

---

# TABLE DES MATIERES

---

Introduction Générale.....	5
Chapitre I : Problème de Bin-Packing.....	6
I. Classification du problème.....	7
II. Bin – Packing à une dimension (1BP).....	7
II.1 Exemples d’application du problème de Bin – Packing à une dimension.....	8
II.1.1 Sauvegarde de fichiers dans des supports informatiques .....	8
II.1.2 Découpe de bois .....	8
II.1.3 Organisation d’une fête .....	8
II.2 Formulation mathématique.....	9
II.2.1 Bin-Packing avec plusieurs bins.....	9
II.2.2 Bin – Packing avec un seul bin.....	10
III. Problème de Bin – Packing à deux dimensions (2BP) .....	12
III.1 Position du problème.....	12
III.2 Modèles Mathématiques.....	14
III.2.1 Modèle de Strip-Packing .....	14
III.2.2 Modèle de Bin-Packing .....	16
III.2.3 Modèle de Knapsack (Sac-à-dos).....	19
Chapitre II : Méthodes de résolution .....	23
I. L’optimisation combinatoire .....	23
II. Méthodes exactes .....	24
II.1 Méthode de Branch & Bound.....	24
II.1.1 Principe.....	24
II.1.2 Fonctionnement .....	24
III. Méthodes approchées .....	27
III.1 Le cas 1BP.....	27
III.1.1 Stratégie Next – Fit (N.F).....	27
III.1.2 Stratégie First – Fit (F.F).....	28
III.1.3 Stratégie Best-Fit (B.F) .....	29
III.1.4 Stratégie Worst-Fit (W.F).....	29
III.2 Le cas 2BP.....	30

III.2.1	Méthode en une phase .....	30
III.2.2	Méthodes en deux phases .....	32
IV.	Métaheuristiques : Algorithmes Génétiques.....	33
IV.1	Principe et déroulement.....	33
IV.2	Codage.....	34
IV.3	Création de la population initiale .....	34
IV.4	Evaluation des individus .....	34
IV.5	Sélection .....	35
IV.6	Reproduction.....	36
IV.6.1	Croisement.....	36
IV.6.2	Mutation .....	37
IV.7	Schéma récapitulatif.....	38
Chapitre III : Application de l'Algorithme Génétique aux problèmes de Bin - Packing .....		39
I.	Application au problème de Sac – à – dos à une dimension .....	40
I.1	Algorithme génétique .....	40
I.1.1	Codage.....	40
I.1.2	Sélection .....	41
I.1.3	Croisement.....	41
I.1.4	Mutation .....	42
I.2	Plate forme .....	42
I.2.1	Les étapes de l'exécution.....	43
I.2.2	Exemple d'exécution .....	45
I.3	Brunch&Bound .....	45
II.	Application au problème de Bin – Packing à une dimension.....	48
II.1	Plat forme .....	49
II.2	Exemple d'exécution :.....	50
Conclusion générale .....		51

---

# Introduction Générale

---



Le secteur industriel est fréquemment confronté à des problèmes de découpe de matières premières. Dans l'intérêt de réduire les coûts et impact environnemental, l'objectif est de minimiser la quantité de matériaux perdue dans les chutes. Dans le secteur de transport, le développement des activités logistiques provoque la multiplication de problèmes de conditionnement. La résolution de ces problèmes est un enjeu économique très important pour de nombreuses entreprises.

Les problèmes de découpage de matériaux et de placement sont des problèmes classiques de la littérature, ce sont des généralisations du problème classique connu sous le nom de « Bin-Packing ». Ces problèmes consistent à placer le maximum d'objets possible dans un conteneur de taille fixe, de telle manière que les objets ne se chevauchent pas et qu'ils ne dépassent pas la taille du conteneur.



Ces problème de Bin – Packing font partie des problèmes combinatoire les plus étudiés, ils peuvent être en une, deux ou trois dimensions suivants les caractéristiques des objets à ranger. Notre travail, qui s'inscrit dans le cadre d'un mémoire de fin d'étude de la licence Calcul Scientifique et Application, et qui a été effectué au sein de la faculté des Sciences et Technique de Fès (FSTF), a pour objectif : « La réalisation de deux applications dont la première permet de déterminer les objets qu'il faut mettre dans un seul bin de manière à maximiser sa valeur totale sans dépasser le poids maximal de ce dernier, alors que la deuxième application permet de donner un rangement économique d'un ensemble d'objets dans plusieurs bins »

Ce mémoire est organisé comme suit : dans le premier chapitre, nous étudions ces problèmes en utilisant la programmation par contraintes, ceci consiste d'abord à proposer une modélisation des problèmes par la détermination de l'ensemble des variables du problème, de leurs domaines et les différentes contraintes qui les lient. Dans le deuxième chapitre, nous proposons quelques méthodes de résolutions qui consistent à trouver la solution optimale et nous définissons aussi des heuristiques qui permettent de déterminer une solution efficace en un temps raisonnable. Alors que dans le troisième chapitre nous nous intéressons à réaliser nos applications qui permettent de donner une solution d'un problème de Bin – Packing à une dimension.

---

# Chapitre I : Problème de Bin-Packing

---

Le problème de Bin-Packing relève de la recherche opérationnelle et de l'optimisation combinatoire. Il s'agit de trouver le rangement le plus économique possible pour un ensemble d'objets dans des boîtes dites « bins ». Ce problème a fait l'objet depuis plusieurs années d'une attention croissante de plusieurs chercheurs, car il a de nombreuses applications dans le domaine industriel, informatique et même de l'édition.

Le problème classique se définit en une dimension, mais il existe de nombreuses variantes en deux et trois dimensions.

En effet, les problèmes de Bin – Packing se distinguent en trois types :

- Bin – Packing à une dimension, où les objets sont caractérisés par une seule mesure comme la hauteur, la largeur ou le poids, qu'il faut ranger sur un axe de taille fixée. On peut citer comme exemple de ces problèmes le rangement de fichiers sur un support informatique, la découpe de câbles, le remplissage de camions ou de containers avec comme seule contrainte le poids ou le volume des articles.
- Bin – Packing à deux dimensions où il faut ranger les objets sur une surface limitée, tels que la découpe de matière première (bois, verre, acier, etc.), le placement de boîtes sur une palette (sans superposition de boîtes), le placement dans un entrepôt (sans superposition de boîtes), placement des articles dans un journal.
- Bin – Packing à trois dimensions où les objets sont rangés dans un espace de volume fixe comme le rangement d'objets dans des boîtes, un entrepôt, des camions, etc. (avec superposition de boîtes, de palette, etc.)

Les problèmes de type Bin – Packing ont été abordés dans la littérature dès les années 1950, essentiellement sur des problèmes à une dimension, mais comme il a fait l'objet depuis plusieurs années d'une attention croissante de beaucoup de chercheurs, il existe actuellement un bon nombre d'articles traitant ces problèmes en deux dimensions. Dans le cas du problème à trois dimensions, il



constitue un nouveau champ de recherche auquel les chercheurs commencent à s'intéresser de plus en plus.

## **I. Classification du problème**

Il existe plusieurs variantes pour le problème de Bin-Packing, que ce soit de type bin-packing à une dimension, deux dimensions ou trois dimensions, et chaque variante présente ses propres spécificités telle que : L'orientation des objets, le nombre de dimensions, le type de tâches, le nombre de bins, les caractéristiques des objets, etc.

Dyckhoff et Finke ont proposé une typologie qui permet d'organiser les problèmes de Bin-Packing en tenant compte de quatre caractéristiques principales :

- 1- le nombre de dimensions du problème ;
- 2- le type de tâche : tous les objets et une sélection de bins, ou bien une sélection d'objets et tous les bins ;
- 3- les caractéristiques des bins : 1 seul bin, des bins de tailles identiques, ou bien des bins de tailles différentes ;
- 4- les caractéristiques des objets : objets identiques, peu d'objets de formes différentes, plusieurs objets de formes différentes ou bien des objets de formes relativement identiques.

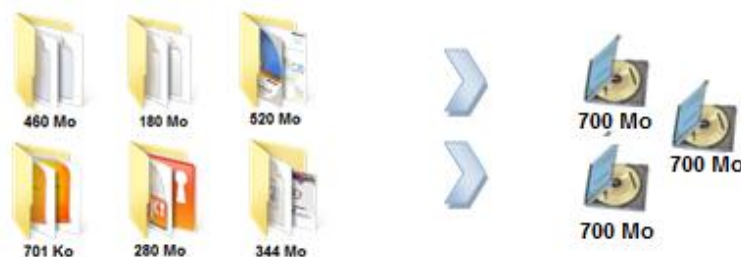
## **II. Bin – Packing à une dimension (1BP)**

Le problème de Bin – Packing à une dimension est l'un des problèmes classiques de l'optimisation combinatoire, il consiste à placer un ensemble d'objets caractérisés par une seule variable (hauteur, largeur, poids, ou autre) dans un ensemble de boîtes appelées « *bins* ». L'objectif de ce rangement peut varier d'un problème à un autre. En effet, on peut distinguer le rangement qui consiste à minimiser le nombre total de bins utilisés et en respectant la taille maximale de chaque bin, et le rangement qui consiste à mettre certains éléments de l'ensemble des objets dans un seul bin, de façon à maximiser la fonction objectif du problème étudié, comme la valeur du sac s'il s'agit d'un problème de Sac-à-dos. Nous notons que le problème de sac à dos constitue l'un de ses applications les plus usuels. Néanmoins, plusieurs autres applications existent dans la littérature.

## II.1 Exemples d'application du problème de Bin – Packing à une dimension

### II.1.1 Sauvegarde de fichiers dans des supports informatiques

Je souhaite sauvegarder le contenu de mon disque dur sur des CD-Rom de 700 Mo. Chaque dossier a une taille donnée, et je ne souhaite pas découper les dossiers. Comme il ne me reste plus beaucoup de CD, je cherche le minimum de disques que je peux utiliser. Notez bien que dans cette version du problème, je ne cherche pas à regrouper spécialement des fichiers par genre. J'ai un problème de bin packing à résoudre !



### II.1.2 Découpe de bois

Un menuisier a reçu une commande d'un ensemble d'étagères de tailles différentes. Dans le but de répondre à cette commande au moindre coût, le menuisier voudrait utiliser le minimum de planches possibles pour découper les rectangles ayant la même largeur nécessaires pour fabriquer les étagères. Il s'agit d'un problème de Bin-Packing à une dimension.



### II.1.3 Organisation d'une fête

J'ai une fête à organiser, et je dois inviter un certain nombre de familles, chaque famille est composée d'un ou plusieurs personnes, Comme il ne me reste pas beaucoup d'argent, je cherche le minimum de tables à utiliser pour ce dîner. Sachant qu'on ne peut pas séparer une personne de sa famille.

## II.2 Formulation mathématique

La modélisation d'un problème réel, consiste à le rendre sous forme d'un ensemble d'équations mathématiques. Dans cette partie, nous présentons la modélisation du problème de Bin-Packing à une dimension sous forme de programme mathématique, dans la première partie, nous modélisons le problème dans le cas de plusieurs bins alors que la deuxième partie sera consacrée au problème dans le cas d'un seul bin. C'est le cas du problème de sac à dos.

La recherche du modèle mathématique d'un problème revient à identifier les composantes suivantes :

1. Les variables
2. L'objectif
3. Les contraintes

### II.2.1 Bin-Packing avec plusieurs bins

Dans cette section, nous présentons la modélisation du problème classique de Bin – Packing à une seule dimension dans le cas de plusieurs bins. Pour cela, nous utilisons les notations suivantes :

#### (i) Notations

Données :

- $n$  : le nombre d'objets à ranger
- $w_j$  : le poids de l'objet  $j$
- $W_i$  : la capacité maximale du bin  $i$

Variables :

- $y_i = \begin{cases} 1 & \text{si le bin } i \text{ est utilisé} \\ 0 & \text{sinon} \end{cases}$
- $x_{ij} = \begin{cases} 1 & \text{si l'objet } j \text{ est dans le bin } i \\ 0 & \text{sinon} \end{cases}$

#### (ii) Contraintes

La première contrainte que nous devons exprimer est le respect de la capacité maximale de chaque bin, cela est donné par l'inégalité suivante :

$$\sum_{j=1}^n w_j \cdot x_{ij} \leq W_i \quad \forall i = 1, \dots, n$$

La deuxième contrainte impose à tous les objets d'être rangés dans une boîte et une seule :

$$\sum_{i=1}^n x_{ij} = 1 \quad \forall j = 1, \dots, n$$

Toute solution pour laquelle la famille d'équations précédente est vérifiée est dite *réalisable*.

### (iii) Objectif

Nous rappelons que l'objectif du problème classique de Bin – Packing est de minimiser le nombre total des bins utilisés, ce qui revient à minimiser la somme totale des variables  $y_i$ .

D'où la fonction objectif :

$$\min \sum_{i=1}^n y_i$$

Le modèle mathématique s'écrit alors sous la forme suivante :

$$\left\{ \begin{array}{ll} \min \sum_{i=1}^n y_i \\ \text{s. c.} \\ \sum_{j=1}^n w_j \cdot x_{ij} \leq W_i & \forall i = 1, \dots, n \\ \sum_{i=1}^n x_{ij} = 1 & \forall j = 1, \dots, n \\ x_{ij}, y_i \in \{0,1\} & \forall i, j = 1, \dots, n \end{array} \right.$$

## II.2.2 Bin – Packing avec un seul bin

Dans cette section, nous traitons le cas particulier du problème de Bin – Packing à une dimension où le nombre de bins est réduit à 1. Le problème de Sac-à-dos est l'une des applications directes de ce type de problème ;



Il peut être énoncé comme suit : « *Etant donné plusieurs objets possédant chacun un poids et une valeur et étant donné un poids maximum pour le sac. Quels objets faut-il mettre dans le sac de manière à maximiser la valeur totale sans dépasser le poids maximal autorisé pour le sac ?* »

Cela doit certainement vous faire rappeler de ces longues journées de préparation de valise, avec cette contrainte « chaque passager a le droit d'avoir avec lui un seul bagage cabine, dont les dimensions maximales sont de 55 cm x 35 cm x 25 cm, avec un poids qui ne doit pas dépasser 12 kg ». La fameuse question qui se pose à chaque voyageur lors de sa préparation des bagages : « Qu'est-ce-que je dois prendre, si je ne doit pas dépasser les 12 Kg ? »



Pour présenter la formulation mathématique de ce problème, nous utilisons les notations suivantes :

#### (i) Notations

Donnés :

- $P$  : la capacité maximal du sac
- $p_i$  : le poids de l'objet  $i$
- $w_i$  : la valeur de l'objet  $i$

Variables :

Dans ce problème il est question de déterminer quels objets faut-il mettre dans le sac, nous allons utiliser dans ce cas une seule variable de décision définie par :

$$x_i = \begin{cases} 1 & \text{si l'objet } i \text{ est mis dans le sac} \\ 0 & \text{sinon} \end{cases}$$

#### (ii) Contraintes

La seule contrainte qu'il faut respecter dans le cas du problème de sac à dos est que la somme des poids de tous les objets dans le sac doit être inférieure ou égale au poids maximal du sac à dos. C'est ce qui est exprimé par l'inégalité ci-dessous.

$$\sum_{i=1}^n x_i p_i \leq P$$

### (iii) Objectif

Contrairement au problème classique de Bin – Packing où il est question de minimiser le nombre de bins utilisés, le problème de sac à dos a pour objectif de maximiser la valeur totale des objets se trouvant dans le sac. Cela est traduit par la fonction objectif suivante :

$$\max \sum_{i=1}^n x_i w_i$$

Le problème mathématique s'écrit donc sous la forme suivante :

$$\left\{ \begin{array}{l} \max \sum_{i=1}^n x_i w_i \\ S. C \\ \sum_{i=1}^n x_i p_i \leq P \\ x_i \in \{0,1\} \end{array} \right.$$

## III. Problème de Bin – Packing à deux dimensions (2BP)

Dans cette partie, nous définissons le problème de Bin-Packing en deux dimensions et présentons deux modèles différents permettant de le modéliser mathématiquement.

### III.1 Position du problème

Etant donné un ensemble d'objets rectangulaires caractérisés par une hauteur et une largeur données, le problème de Bin-Packing à deux dimensions consiste à les placer dans des boîtes de tailles plus grandes appelées « *bins* » de telle sorte que les objets ne se chevauchent pas et ne dépassent pas la taille fixée des bins. L'objectif varie selon le problème étudié, dans certains cas, il est question de minimiser le nombre total des bins utilisés alors que dans d'autres cas, le but est de maximiser le nombre d'objets à mettre dans les bins.

Ce type de problème peut être rencontré dans plusieurs applications industrielles telles que la découpe de matière première (bois, verre, aluminium) dans le but de minimiser les chutes, dans le domaine de la logistique, on est par exemple amené à minimiser le nombre de containers nécessaires au transport de marchandise, etc.

Plusieurs types de Bin-Packing à deux dimensions sont à distinguer, nous citons par exemple :

- **Le problème de strip-packing 2SP** : c'est une variante très étudiée du Bin-Packing en deux dimensions. On dispose d'une liste  $A$  d'articles et d'un conteneur unique de largeur  $W$  et illimité en hauteur. L'objectif est de ranger tous les articles de  $A$  dans le conteneur en minimisant la hauteur totale à utiliser. Ce problème on le trouve rarement dans la réalité, mais il paraît parfois comme un sous-problème à résoudre pour trouver la solution d'un problème de 2BP.
- **Le problème de Subset Sum (la somme des sous-ensembles)** : ce problème est décrit ainsi, étant donné un ensemble d'entiers positifs  $S$  et une somme cible  $t$ , existe-t-il un sous-ensemble de  $S$  dont la somme fait  $D$  ? Ainsi, pour chaque objet  $a_i$ , la valeur optimale de la fonction  $f(A, w_j, D)$  est évaluée.  $f(A, w_i, D)$  représente la largeur maximale inférieure ou égale à  $D$  qui peut être atteinte en plaçant un ensemble d'objets, y compris  $a_i$ , l'un à côté de l'autre dans un bin. L'évaluation de la fonction  $f$  tient compte de toutes les permutations. Le problème de subset-sum n'est pas, proprement parlant, un problème de Bin-Packing mais il paraît parfois comme un sous-problème à résoudre pour obtenir des évaluations par défaut.
- **Le problème de knapsack (Sac-à-dos)** : On dispose d'un ensemble d'articles, ayant chacun une taille précise, ainsi que la taille de bin. Chaque article possède en outre un coût. L'objectif est de ranger dans le bin (sac) un ensemble d'articles qui maximise la somme des coûts associés sans dépasser la taille maximal du bin.

## III.2 Modèles Mathématiques

Dans cette partie, nous décrivons le modèle mathématique de quelques types de Bin-Packing à deux dimensions. Nous notons  $(w_i, h_i)$  les dimensions d'un objet  $a_i$  appartenant à l'ensemble  $A$  des rectangles à ranger, et  $(W, H)$  les dimensions du bin  $B : B = (W, H)$ .

Tout au long de cette section, nous considérons que nos objets sont triés par ordre décroissant par rapport à leurs hauteurs.



### III.2.1 Modèle de Strip-Packing

Pour présenter une formulation complète, nous devons noter que les objets de la liste  $A$  sont placés successivement dans un conteneur de hauteur infinie, en le remplissant couche par couche, formant ainsi des niveaux telle que, le 1<sup>er</sup> Objet placé dans chaque niveau est le plus haut dans ce niveau car les objets sont triés par ordre décroissant, on l'appelle l'objet qui initialise le niveau



#### (i) Notations

Donnés :

- $n$  : le nombre d'objets à ranger
- $w_i$  : la largeur de l'objet  $i$
- $h_i$  : la hauteur de l'objet  $i$
- $W$  : la largeur du conteneur

Variabes :

- $y_i = \begin{cases} 1 & \text{si l'objet } i \text{ initialise le niveau } i \\ 0 & \text{sinon} \end{cases}$

- $x_{ij} = \begin{cases} 1 & \text{si l'objet } j \text{ est dans le niveau } i \\ 0 & \text{sinon} \end{cases}$

## (ii) Contraintes

Dans ce problème nous disposons de plusieurs types de contraintes : Contrainte de rangement d'un objet, contrainte de disponibilité et contraintes de non-négativité et d'intégrité. Prenons en considération tour à tour chacune de ces contraintes.

- *Contrainte de rangement d'un objet* : chaque objet de la liste doit être rangé une et une seule fois dans le bin, on a deux possibilités : soit l'objet initialise le niveau soit il est rangé dans un niveau précédant :

$$\sum_{i=1}^{j-1} x_{ij} + y_j = 1 \quad \forall j = 1, \dots, n$$

- *Contrainte de disponibilité* : le total des largeurs des objets misent sur chaque niveau avec l'objet  $i$  ne doit pas dépasser la largeur disponible dans le bin :

$$\sum_{j=i+1}^n w_j x_{ij} \leq (W - w_i) y_i \quad \forall i = 1, \dots, n-1$$

- *Contrainte de non-négativité et d'intégrité* :

$$y_i, x_{ij} \in \{0,1\} \quad \forall i, j = 1, \dots, n$$

## (iii) Objectif

D'après la définition donnée auparavant, l'objectif du Strip-Packing est de minimiser la hauteur utilisée dans le conteneur. La hauteur de chaque niveau est définie par la hauteur de l'objet qui l'initialise, si l'objet  $i$  initialise le niveau  $i$  ( $y_i = 1$ ) alors la hauteur de ce niveau est  $h_i$ , mais s'il ne l'initialise pas ( $y_i = 0$ ) alors dans ce cas la hauteur de ce niveau sera 0. En général la hauteur de chaque niveau est égale à  $h_i y_i$ , donc notre fonction objectif sera :

$$\min \sum_{i=1}^n h_i y_i$$

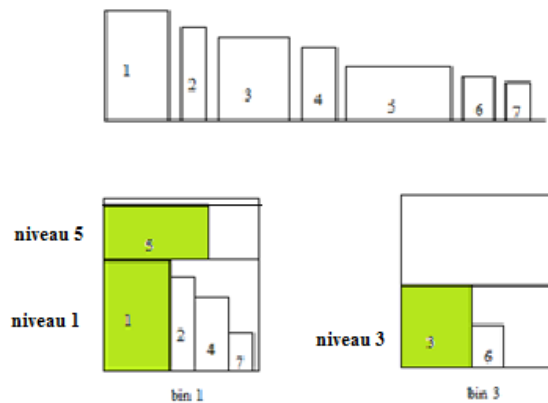
Le modèle mathématique s'écrit alors sous la forme suivante :

$$\left\{ \begin{array}{l} \min \sum_{i=1}^n h_i y_i \\ S.C \quad \sum_{i=1}^{j-1} x_{ij} + y_j = 1 \quad \forall j = 1, \dots, n \\ \sum_{j=i+1}^n w_j x_{ij} \leq (W - w_i) y_i \quad \forall i = 1, \dots, n-1 \\ x_{ij}, y_j \in \{0,1\} \quad \forall i, j = 1, \dots, n \end{array} \right.$$

### III.2.2 Modèle de Bin-Packing

De manière formelle, les objets sont placés successivement, mais dans ce cas dans des bins de largeur  $W$  et de hauteur  $H$  bien définies, en le remplissant couche par couche, formant ainsi des niveaux tels que :

- le 1<sup>er</sup> Objet placé dans chaque niveau est le plus haut dans ce niveau, on l'appelle l'objet qui initialise le niveau.
- le 1<sup>er</sup> niveau de chaque bin est le plus haut dans le bin, de même ce niveau est dit le niveau qui initialise le bin.



#### (i) Notation

Donnés :

- $n$  : le nombre d'objets à ranger

- $w_i$  : la largeur de l'objet  $i$
- $h_i$  : la hauteur de l'objet  $i$
- $W$  : la largeur du conteneur

Variables :

- $y_i = \begin{cases} 1 & \text{si l'objet } i \text{ initialise le niveau } i \\ 0 & \text{sinon} \end{cases}$
- $x_{ij} = \begin{cases} 1 & \text{si l'objet } j \text{ est dans le niveau } i \\ 0 & \text{sinon} \end{cases}$
- $q_k = \begin{cases} 1 & \text{si le niveau } k \text{ initialise le bin } k \\ 0 & \text{sinon} \end{cases}$
- $z_{ki} = \begin{cases} 1 & \text{si le niveau } k \text{ est dans le bin } i \\ 0 & \text{sinon} \end{cases}$

## (ii) Contraintes

La première contrainte que nous décrivons est : chaque objet ne doit être placé que dans un seul niveau, ou il l'initialise ou bien il est placé dans un niveau qui est déjà initialisé par un autre objet. Cela est donné par la formule suivante :

$$\sum_{i=1}^{j-1} x_{ij} + y_j = 1 \quad \forall j = 1, \dots, n$$

La deuxième contrainte signifie que pour chaque niveau utilisé on ne peut dépasser la largeur d'un bin.

$$\sum_{j=i+1}^n w_j x_{ij} \leq (W - w_i) y_i \quad \forall i = 1, \dots, n - 1$$

La troisième contrainte assure que chaque niveau utilisé est rangé exactement dans un seul bin de même, soit il initialise le bin ou bien il est rangé dans un bin déjà initialisé par un autre niveau,

$$\sum_{k=1}^{i-1} z_{ki} + q_i = y_i \quad \forall i = 1, \dots, n$$

Finalement la dernière contrainte assure que dans chaque bin utilisé on ne peut dépasser la hauteur maximal du bin.

$$\sum_{j=i+1}^n h_i z_{ki} \leq (H - h_k) q_k \quad \forall k = 1, \dots, n - 1$$

### (iii) Objectif

L'objectif est de minimiser le nombre de bins utilisés par l'utilisateur. On sait que chaque bin est soit initialisé par le niveau  $i$  ou vide, donc le nombre de bins utilisé est :

$$Z = \sum_{i=1}^n q_i$$

Par suite, la fonction objectif sera exprimée par :

$$\min Z = \sum_{i=1}^n q_i$$

Le modèle mathématique s'écrit sous la forme suivante :

$$\left\{ \begin{array}{ll} \min Z = \sum_{i=1}^n q_i & \\ \text{S. C} & \\ \sum_{i=1}^{j-1} x_{ij} + y_j = 1 & \forall j = 1, \dots, n \\ \sum_{j=i+1}^n w_j x_{ij} \leq (W - w_i) y_i & \forall i = 1, \dots, n - 1 \\ \sum_{k=1}^{i-1} z_{ki} + q_i = y_i & \forall i = 1, \dots, n \\ \sum_{j=i+1}^n h_i z_{ki} \leq (H - h_k) q_k & \forall k = 1, \dots, n - 1 \\ x_{ij}, y_j, z_{ki}, q_k \in \{0, 1\} & \forall i, j, k = 1, \dots, n \end{array} \right.$$



### III.2.3 Modèle de Knapsack (Sac-à-dos)

#### Modélisation en utilisant une position relative des objets

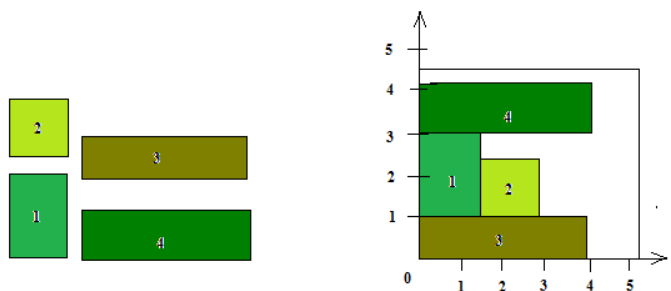
Nous rappelons que le problème de sac-à-dos consiste à ranger dans un seul bin, un ensemble d'objets (ayant chacun une valeur et une taille précise) qui maximise la somme des valeurs associé sans chevauchement des objets les uns les autres, ni dépassement de la taille du bin, de plus la rotation des objets est interdite.

Données :

- $n$  : le nombre d'objets
- $w_i$  : la largeur de l'objet  $i$
- $h_i$  : la hauteur de l'objet  $i$
- $\rho_i$  : la valeur de l'objet  $i$
- $W$  : la largeur du bin
- $H$  : la hauteur du bin

Variables :

- $(x_i, y_i)$  : les coordonnées du coin inférieur-gauche de l'objet  $i$
- $\delta_i = \begin{cases} 1 & \text{si l'objet } i \text{ est mis dans le sac} \\ 0 & \text{sinon} \end{cases}$
- $g_{ij} = \begin{cases} 1 & \text{si l'objet } i \text{ est à gauche de l'objet } j \\ 0 & \text{sinon} \end{cases}$
- $d_{ij} = \begin{cases} 1 & \text{si l'objet } i \text{ est au dessous de l'objet } j \\ 0 & \text{sinon} \end{cases}$



Coordonnées

Objet	(x,y)
1	(0,1)
2	( $\frac{1}{2}$ ,1)
3	(0,0)
4	(0,3)

### (i) Contraintes

La 1<sup>ère</sup> contrainte impose que chaque pair d'objet  $(i,j)$  sélectionnée est soit l'un à gauche de l'autre ; soit l'un en dessous de l'autre. Donc chaque paire d'objets ne peut s'intersecter dans le rangement.

$$g_{ij} + g_{ji} + d_{ij} + d_{ji} = \delta_i \cdot \delta_j \quad \forall i, j = 1, \dots, n / i \neq j$$

La 2<sup>ème</sup> contrainte assure que si l'objet  $i$  est à gauche de l'objet  $j$  alors on ne peut pas dépasser la largeur du bin ;

$$x_i + w_i \leq x_j + W(1 - g_{ij}) \quad \forall i, j = 1, \dots, n / i \neq j$$

La 2<sup>ème</sup> contrainte assure que si l'objet  $i$  est au dessous de l'objet  $j$  alors on ne peut pas dépasser la hauteur du bin ;

$$y_i + h_i \leq y_j + H(1 - d_{ij}) \quad \forall i, j = 1, \dots, n / i \neq j$$

Finalement la 3<sup>ème</sup> et la 4<sup>ème</sup> contraintes imposent que les coordonnées des objets ne peuvent pas dépasser la taille du bin.

$$\begin{aligned} 0 \leq x_i \leq W - w_i & \quad \forall i = 1, \dots, n \\ 0 \leq y_i \leq H - h_i & \quad \forall i = 1, \dots, n \end{aligned}$$

### (ii) Objectif

Nous cherchons à maximiser la valeur du sac, c'est-à-dire rendre la somme des valeurs des objets mis dans le sac aussi grande que possible. Nous écrivons alors :

$$\max \sum_{i=1}^n \rho_i \delta_i$$

Le programme mathématique s'écrit donc de la façon suivante :

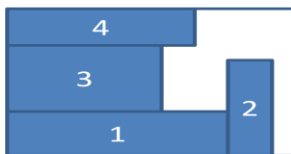
$$\left\{ \begin{array}{ll} \max \sum_{i=1}^n \rho_i \delta_i & \\ \text{S.C} & \\ g_{ij} + g_{ji} + d_{ij} + d_{ji} = \delta_i \cdot \delta_j & \forall i, j = 1, \dots, n / i \neq j \\ x_i + w_i \leq x_j + W(1 - g_{ij}) & \forall i, j = 1, \dots, n / i \neq j \\ y_i + h_i \leq y_j + H(1 - d_{ij}) & \forall i, j = 1, \dots, n / i \neq j \\ 0 \leq x_i \leq W - w_i & \forall i = 1, \dots, n \\ 0 \leq y_i \leq H - h_i & \forall i = 1, \dots, n \\ \delta_i, g_{ij}, d_{ij} \in \{0,1\} & \end{array} \right.$$

### Modélisation en utilisant la théorie des graphes :

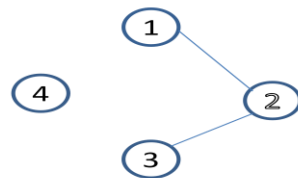
Nous retrouvons dans la littérature une autre modélisation du problème de bin packing avec un seul bin qui s'énonce comme suit : « *Est-il possible de placer un ensemble donné d'objets dans un seul bin ?* ». Cette modélisation est basée sur la théorie de graphes et a été proposée par Fekete & Schepers.

Soient deux graphes  $G_h = (V, E_h)$  et  $G_w = (V, E_w)$ , avec  $V$  l'ensemble des sommets du graphe dont chacun correspond à un objet mis dans le bin, et  $E_h$  (resp.  $E_w$ ) l'ensemble des arêtes du graphe  $G_h$  (resp.  $G_w$ ).

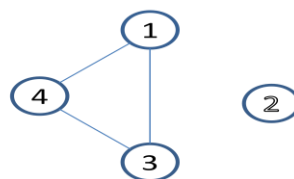
Les deux graphes sont construits de la façon suivante : une arête est rajoutée dans  $G_h$  (resp. dans  $G_w$ ) entre deux sommets  $v_i$  et  $v_j$  si et seulement si les projections des objets  $a_i$  et  $a_j$  sur l'axe vertical (resp. horizontal) se chevauchent.



Le graphe  $G_h$  :



Le graphe  $G_w$  :



Pour qu'une solution associée à un couple de graphes  $(G_h, G_w)$  soit réalisable, il suffit que les trois conditions suivantes soient vérifiées :

- 1 – Pour chaque ensemble stable  $S_w$  de  $G_w$ , on a  $\sum_{v_i \in S_w} w_i \leq W$
- 2 – Pour chaque ensemble stable  $S_h$  de  $G_h$ , on a  $\sum_{v_i \in S_h} w_i \leq H$
- 3 –  $E_h \cap E_w = \emptyset$

### ***Définition***

Un graphe est dit stable si et seulement s'il est un graphe sans arête

Les deux premières conditions signifient que le rangement proposé respecte la largeur et la hauteur du bin qu'on ne doit pas dépasser, et la troisième condition signifie qu'il n'y a pas de chevauchement entre deux objets quelconques rangés dans le bin.

---

# Chapitre II : Méthodes de résolution

---

Nous introduisons dans ce chapitre la définition d'un problème d'optimisation combinatoire, nous décrivons les différentes méthodes qui existent pour résoudre ce type de problèmes. Enfin, nous présentons une description détaillée de quelques méthodes heuristiques, exactes et métaheuristiques connues dans la littérature et appliquées au problème Bin-Packing *IBP* et *2BP*.

## I. L'optimisation combinatoire

L'optimisation combinatoire est une branche de la recherche opérationnelle, dans sa forme la plus générale, un problème qui relève de l'optimisation combinatoire s'écrit sous la forme suivante :

$$\begin{cases} \text{opt } f(x) \\ \text{sc} \\ f_i(x) \leq 0 \quad i = 1, \dots, n \\ x \in C \end{cases}$$

Avec, *opt* signifie minimiser ou maximiser, et *C* un sous ensemble fini de  $\mathbb{R}^n$ .

Un problème d'optimisation combinatoire (on dit aussi d'optimisation discrète) consiste à trouver dans un ensemble discret la meilleure solution réalisable (solution optimale).

Trouver une solution optimale dans un ensemble discret et fini est un problème facile en théorie : il suffit d'essayer toutes les solutions, et de comparer leurs qualités pour voir la meilleure. Cependant, en pratique, l'énumération de toutes les solutions peut prendre trop de temps. Or, le temps de recherche de la solution optimale est un facteur très important et c'est à cause de lui que les problèmes d'optimisation combinatoire sont réputés si difficiles. De plus, comme l'ensemble des solutions réalisables est défini de manière implicite, il est aussi parfois très difficile de trouver ne serait-ce qu'une solution réalisable.

De nombreuses méthodes de résolution ont été développées en recherche opérationnelle (RO) et en intelligence artificielle (IA). Ces méthodes peuvent être classées sommairement en deux grandes catégories : les méthodes exactes (complètes) qui garantissent la complétude de la

résolution et les méthodes approchées (incomplètes) qui perdent la complétude pour gagner en efficacité.

Les méthodes exactes ont permis de trouver des solutions optimales pour des problèmes de taille raisonnable. Malgré les progrès réalisés (notamment en matière de la programmation linéaire en nombres entiers), les méthodes exactes rencontrent généralement des difficultés face aux applications de taille importante, le temps de calcul nécessaire pour trouver une solution risque d'augmenter exponentiellement avec la taille du problème.

Contrairement aux méthodes approchées, elles constituent une alternative très intéressante pour traiter les problèmes d'optimisation de grande taille si l'optimalité n'est pas primordiale. En effet, ces méthodes sont utilisées depuis longtemps par de nombreux praticiens.

## **II. Méthodes exactes**

Parmi les méthodes exactes, on trouve la plupart des méthodes traditionnelles (développées depuis une trentaine d'années) telles la méthode du Simplex qu'on utilise souvent pour résoudre des programmes linéaire en nombres réels, on trouve aussi la procédure de séparation et évaluation (PSE) ou en anglais Branch & Bound qui est très efficace dans le cas d'un programme linéaire en nombres entiers.

### **II.1 Méthode de Branch & Bound**

#### **II.1.1 Principe**

La technique du Branch & Bound est une méthode algorithmique classique pour résoudre un problème d'optimisation. Il s'agit de rechercher une solution optimale dans un ensemble de solutions réalisables, c'est-à-dire qu'il consiste à chercher la meilleure solution parmi toutes les solutions possibles. Cette méthode repose d'abord sur la séparation (branch) de l'ensemble des solutions en sous-ensembles plus petits, puis l'exploration de ces solutions en utilisant une évaluation optimiste (bound) pour trouver la meilleure d'entre eux.

#### **II.1.2 Fonctionnement**

La dénomination séparation et évaluation (B&B) recouvre deux idées :

1. **Séparation** : permet d'énumérer intelligemment toutes les solutions possibles, pour décrire cette opération il suffit de dire comment on divise un ensemble en sous-

ensembles. pour cela nous utilisons un « arbre de recherche » constitué par des nœuds, des arcs et ils sont repartis en niveaux, telle que :

- chaque nœud représente une étape de construction de la solution, ils prennent soit 1 ou 0
- chaque arc représente les choix faits pour construire la solution

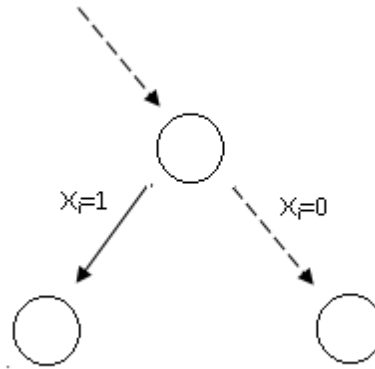


Figure I.1.2

La figure I.1.2 représente une partie de l'arbre de recherche.

L'arbre de recherche commence par un seul nœud de niveau 0 appelé racine de l'arborescence où aucune décision n'a été prise, pour construire le 1<sup>er</sup> niveau on a deux possibilités soit que  $x_i$  prend la valeur 1 soit 0, donc on aura deux sommets dans ce niveau, de la même manière pour construire le niveau 2, on répète la même procédure sur chacun des nouveaux nœuds.

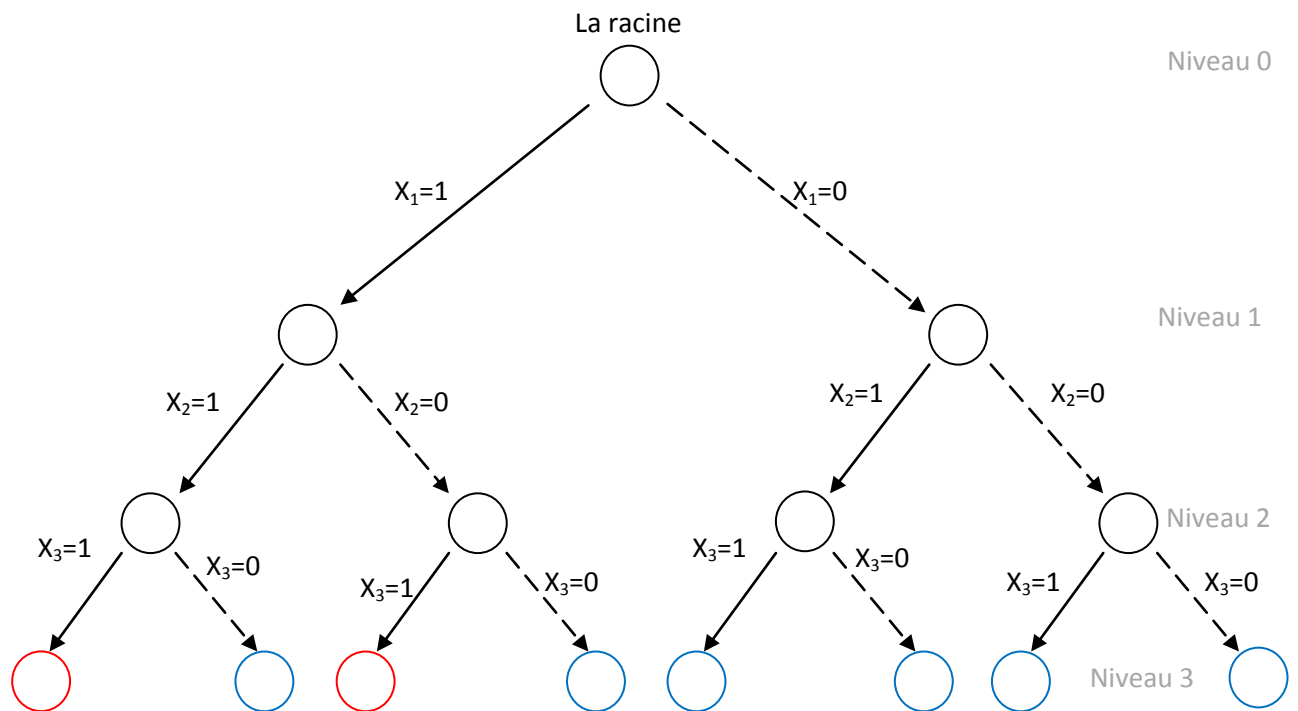


Figure I.1.2.2

Les nœuds en rouge représentent un cas impossible c'est-à-dire un cas où au moins une contrainte n'est plus vérifiée, il n'est donc pas nécessaire de développer cette étape. Les nœuds en bleu sont des nœuds où les contraintes du problème jusqu'à présent sont tous vérifiées.

À la fin de l'algorithme, il suffit de calculer la valeur de la fonction objectif pour chacune des nœuds feuilles, et de prendre la solution avec la plus grande valeur. Mais supposons que le nombre de niveaux est très grand, dans ce cas cette procédure ne sera pas très pratique, car plus la taille de l'arbre augmente plus le nombre de feuilles augmente rapidement. On parle alors de croissance exponentielle.

Pour cela on utilise des techniques pour parcourir ce type d'arbre afin de diminuer la taille de l'arbre et d'augmenter la rapidité.

Un élément essentiel dans la méthode de B&B est le calcul de bornes, inférieures et supérieures, de la fonction objectif.

- **une borne inférieure** est une valeur minimum de la fonction objective. Autrement dit, c'est une valeur qui est nécessairement inférieure à la valeur de la meilleure solution



possible. Dans notre cas, une solution réalisable quelconque fournit une borne inférieure.

- **une borne supérieure** est une valeur maximale de la fonction objective calculé en tous point de l'arbre elle est égale à la somme de toutes les valeurs de tous les objets déjà mis dans le sac plus la somme des valeurs des objets restants dont on ne sait pas encore s'ils seront dans le sac.

Nous allons voir maintenant l'utilité de ces bornes. Supposons que la borne inférieure soit initialisée par l'algorithme vu précédemment, et pendant la recherche à chaque nœud on calcul la borne supérieure de ce nœud, Si jamais, à un nœud donné, la valeur de la borne supérieure est inférieure à la valeur de la borne inférieure, alors il est inutile d'explorer les nœuds descendants de celui-ci. On dit qu'on coupe l'arbre de recherche.

**Remarque : la valeur de la borne inférieure peut être actualisée lorsqu'est trouvée une solution réalisable qui possède une valeur plus grande.**

### III. Méthodes approchées

Dans les problèmes difficiles, comme le cas du problème de bin-packing, il est souvent intéressant d'appliquer des heuristiques qui donnent des solutions de bonne qualité dans un temps raisonnable.

Dans cette section, nous découvrons quelques heuristiques proposées pour le 1BP, elles seront aussi utilisées par la suite pour la résolution de 2BP.

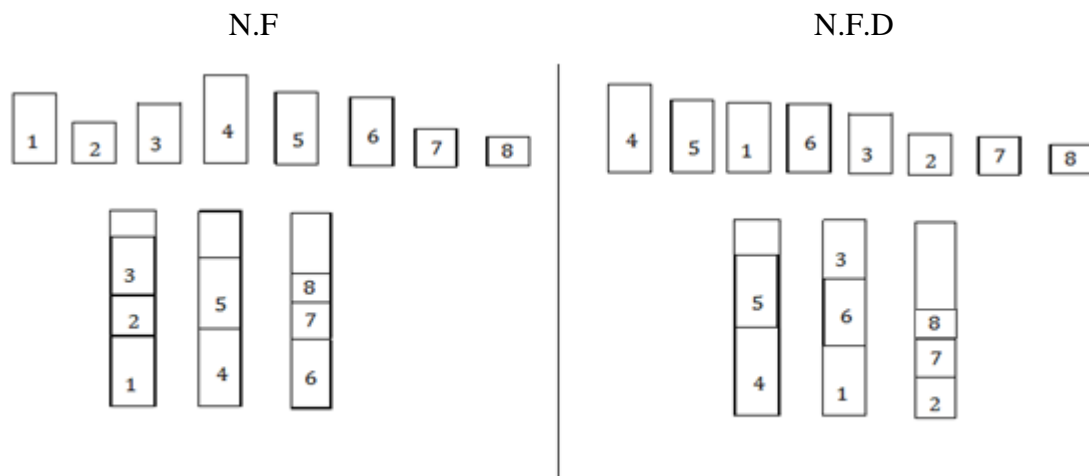
#### III.1 Le cas 1BP

Dans cette section, nous décrivons quelques heuristiques proposées pour le problème de Bin Packing à une dimension (1BP). Les stratégies utilisées dans le 1BP ont inspiré plusieurs approches de résolution proposées plus tard pour des problèmes de bin-packing de dimension plus élevée.

##### III.1.1 Stratégie Next – Fit (N.F)

On ne considère qu'un seul bin ouvert à la fois, les objets sont traités dans un ordre donné, on range les objets successivement dans le bin ouvert tant qu'il y a de la place, sinon on ferme le

bin en cours et on ouvre un nouveau bin. Le Next Fit Decreasing (N.F.D) consiste à trier les objets par ordre décroissant de hauteurs et appliquer la stratégie N.F pour les ranger.

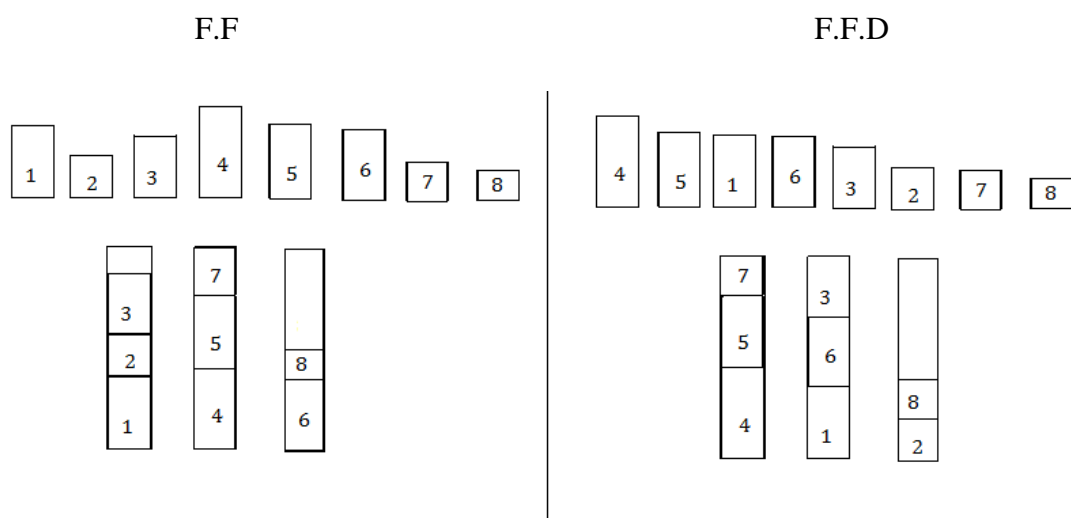


### III.1.2 Stratégie First – Fit (F.F)

Au départ, on ne considère qu'un seul bin, on range les objets successivement dans le ce bin, mais quand il n'y a plus de place dans ce premier bin pour ranger l'objet en cours, un deuxième bin est alors ouvert mais sans fermer le premier.

Dans une étape intermédiaire  $i$  où on dispose de  $k$  bins ouverts, un objet  $a_i$  en cours est rangé dans le bin du plus faible numéro qui peut le contenir, dans le cas où aucun bin ne peut le contenir un nouveau bin  $k+1$  est alors utilisé sans fermer les autres.

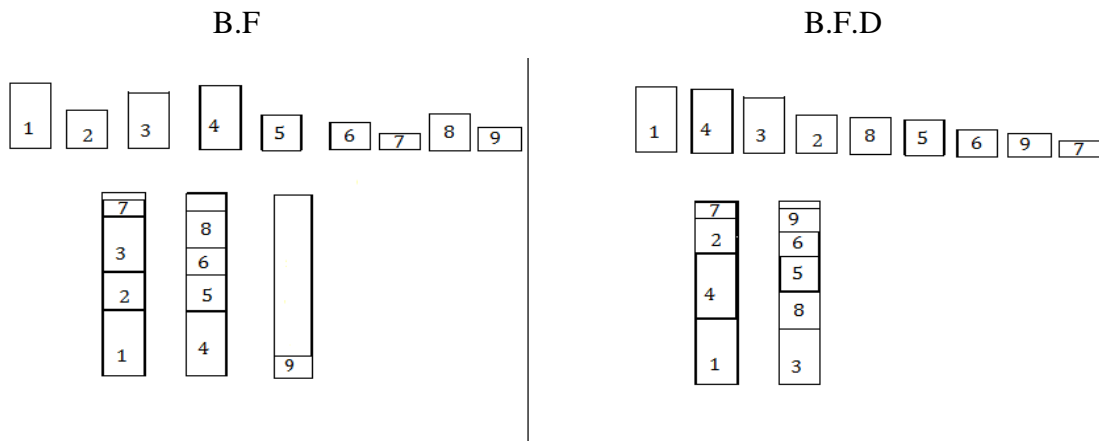
L'ordre selon lequel on traite les objets est crucial pour la qualité de la solution. Un choix heuristique consiste à trier les objets par ordre décroissant de hauteurs, on parle dans ce cas d'heuristiques First Fit Decreasing (F.F.D).



### III.1.3 Stratégie Best-Fit (B.F)

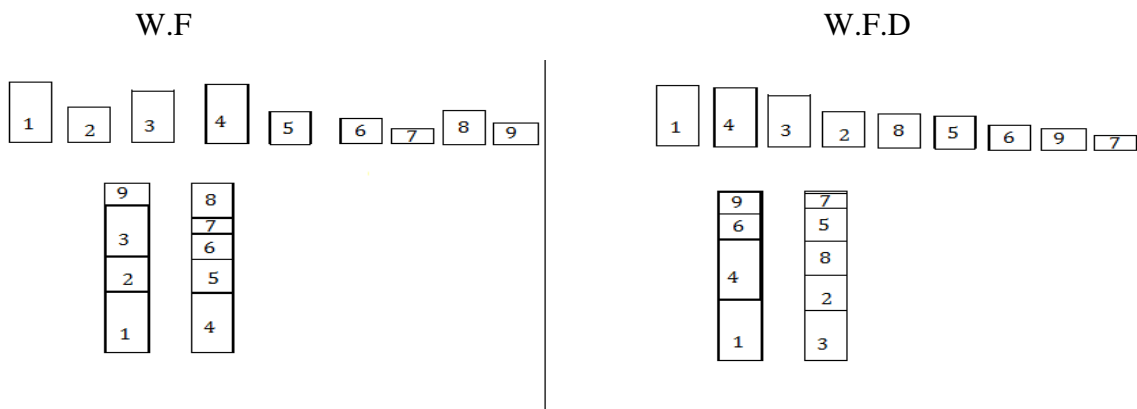
Comme dans la stratégie F.F, les algorithmes B.F laissent les bins toujours ouverts. Cependant, le choix du bin dans lequel l'objet en cours va être placé dépend des valeurs des gaps (hauteurs non utilisées) présentes dans les bins. Un objet  $a_i$  est alors placé dans le bin qui présente le moindre gap (ie hauteurs non utilisées) parmi les bins qui peuvent le contenir.

On parle de Best Fit Decreasing (B.F.D) quand il s'agit de trier les objets à ranger dans l'ordre décroissant de hauteurs avant de les ranger suivant une stratégie B.F.



### III.1.4 Stratégie Worst-Fit (W.F)

C'est une variante du B.F, dans laquelle l'objet en cours est placé dans le bin qui présente le plus grand gap. Aussi surprenant que cela puisse paraître, cette stratégie peut produire de meilleures solutions que BF sur certaines instances. Comme toutes les autres stratégies on peut trier les objets par ordre décroissant avant de les ranger on dit alors qu'on utilise une stratégie Worst-Fit-Decreasing(W.F.D).



## III.2 Le cas 2BP

Le problème de Bin – Packing à deux dimensions est un problème classique pour lequel plusieurs méthodes approchées ont été proposées, ces méthodes sont en général des heuristiques dont la plupart ont été inspirées par les stratégies utilisées dans le 1BP.

On peut diviser ces heuristiques en deux principales familles.

- Les algorithmes en une phase
- Les algorithmes en deux phases

### III.2.1 Méthode en une phase

Les algorithmes en une phase consistent à ranger les objets directement dans les bins. Parmi les algorithmes qui procèdent en une phase, nous trouvons :

#### (i) Finite – First – Fit (F.F.F)

Cette méthode consiste à trier les objets par ordre décroissant par rapport à leurs hauteurs. L'objet en cours est placé dans le niveau le plus bas du premier bin qui peut le contenir, si aucun niveau ne peut le contenir, un nouveau niveau est créé dans le bin ou bien on ajoute un nouveau bin.

#### (ii) Bottom – Left

Elle consiste à placer un objet dans la position la plus en bas à gauche. La procédure est répétée pour chaque objet dans un ordre donné. On place l'objet en cours dans la 1<sup>ère</sup> position la plus bas puis la plus à gauche possible pouvant le contenir, si on ne peut trouver cette position, dans ce cas on ajoute un nouveau bin.

#### (iii) Algorithmes proposés pour le Strip – Packing

Un petit rappel du problème de Strip – Packing :

« Nous avons une collection de rectangles et un conteneur 2D illimité en hauteur, l'objectif est de ranger ces objets dans ce conteneur tout en minimisant la hauteur utilisée. »

Parmi les algorithmes existants pour le problème de Strip – Packing nous trouvons :

- **Algorithme N.F.D.H (Next – Fit – Decreasing - Height)**

Dans l'algorithme NFDH, les rectangles sont triés selon leur hauteur de façon décroissante.

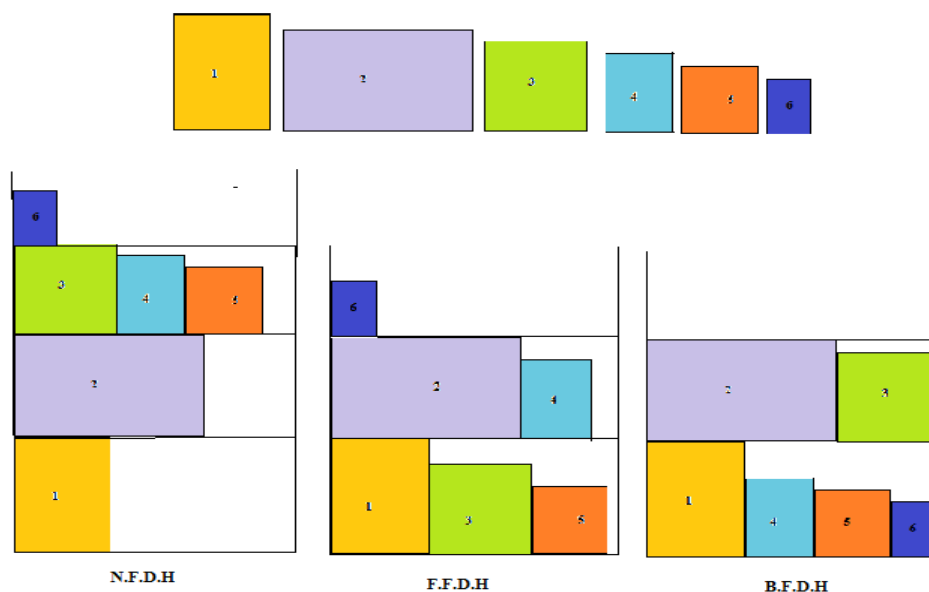
On dépose en bas à gauche le premier rectangle (le plus haut) qui détermine la hauteur de l'étage (définitivement fixé). Puis l'on dépose à côté de celui-ci un autre rectangle (qui est le plus haut rectangle parmi le reste), et l'on continue jusqu'à ce que le rectangle courant ne rentre plus par sa largeur. Dans ce cas, nous déposons ce rectangle au dessus à gauche de l'étagère en créant un nouvel étage. Cette opération est répétée jusqu'à ne plus avoir de rectangle.

- **Algorithme F.F.D.H (First – Fit – Decreasing - Height)**

Le principe de FFDH est très similaire à celui de NFDH. Cependant avant de placer le prochain rectangle sur le niveau courant, nous essayons de le placer dans le niveau inférieur (le plus bas possible), ayant suffisamment d'espace pour l'accueillir. Dans FFDH, nous pouvons donc revisiter un niveau inférieur, ce qui n'est pas permis dans NFDH.

- **Algorithme B.F.D.H (Best – Fit – Decreasing - Height)**

L'algorithme BFDH agit de la même façon que FFDH, sauf que la recherche d'un emplacement du rectangle courant se fait de façon exhaustive sur les étagères. Ainsi l'espace horizontal perdu par le placement d'un rectangle est calculé pour toutes les étagères. Le rectangle est effectivement placé dans l'étagère pour laquelle cet espace perdu est minimal. On peut ainsi logiquement s'attendre à ce que cet algorithme soit plus lent que FFDH qui place les rectangles dans la première étagère trouvée.



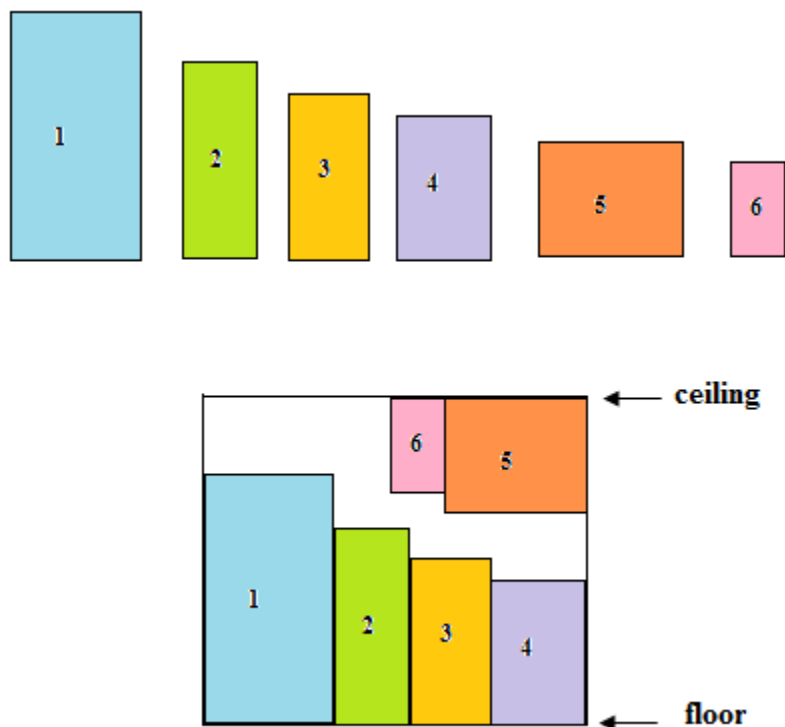
### III.2.2 Méthodes en deux phases

Les Méthodes en deux phases fonctionnent de la manière suivante : La première étape (strip packing) consiste à trier les objets suivant leurs hauteurs décroissantes (Decreasing Height) et les placer successivement dans un conteneur de hauteur infinie, en le remplissant couche par couche, formant ainsi des niveaux définis par la hauteur du plus grand objet par couche, puis ranger ces niveaux dans les bins.

#### (i) Floor – ceiling (F.C)

La 1<sup>ère</sup> phase consiste à placer les objets par couche suivant la stratégie BFDH avec la modification suivante : si un objet ne tient plus sur le floor (étage) d'un niveau, il peut être placé, décalé à droite, et accroché au ceiling (plafond) du niveau, ou on ajoute un nouveau niveau.

Dans la 2<sup>ème</sup> phase, les niveaux sont placés dans des bins finis selon une stratégie B.F.D, ou bien en utilisant une méthode exacte.



## **(ii) Hybrid – First – Fit (H.F.F)**

Dans une première phase, l'algorithme BFDH est effectué pour créer une collection de niveaux de différentes hauteurs. Puis dans la 2<sup>ème</sup> phase, ils emballent les différents niveaux dans les bins. Pour la résolution du problème 1BP de la deuxième phase (ranger les niveaux obtenus dans des bins finis) la stratégie FFD est employée.

## **IV. Métaheuristiques : Algorithmes Génétiques**

Depuis une dizaine d'années, des progrès importants ont été réalisés avec l'apparition d'une nouvelle génération de méthodes approchées puissantes et générales, souvent appelées métaheuristiques.

Une métaheuristique est constituée d'un ensemble de concepts fondamentaux, qui permettent d'aider à la conception de méthodes. Parmi les métaheuristiques les plus connues, on trouve les algorithmes évolutionnaires ou les algorithmes génétiques, les algorithmes de colonies de fourmis, ...

Dans cette section, nous nous intéressons aux algorithmes génétiques, où nous présentons le principe ainsi que les différentes étapes de l'algorithme.

### **IV.1 Principe et déroulement**

Le principe de l'algorithme génétique est basé sur la théorie de Darwin qui pousse les individus les plus adaptés à survivre le plus longtemps pour se reproduire. Un algorithme génétique va faire évoluer une population dans le but d'améliorer les individus.

Le déroulement d'un algorithme génétique peut être découpé en cinq parties :

1. La création de la population initiale.
2. Codage des individus.
3. Evaluation des individus.
4. Sélection
5. La reproduction
6. Retour à l'étape (3) jusqu'à l'arrêt de l'algorithme.

Nous allons analyser plus en détail le principe de fonctionnement étape par étape.

## IV.2 Codage

Le codage consiste à représenter chaque solution du problème sous forme d'un individu. Chaque individu de la population est codé par un chromosome ou plusieurs chromosomes, chaque chromosome comportant plusieurs gènes (le nombre de gène est égal au nombre objets).

Il existe 3 différents choix de codage :

- Codage binaire :

Chaque chromosome est représenté par une chaîne de bits (pouvant prendre comme valeur 0 ou 1)

- Codage à caractères multiples :

Chaque chromosome est représenté par une suite numérique ou de caractères.

- Codage sous forme d'arbre :

Ce codage en structure arborescente part d'une racine, à partir de laquelle peuvent être issus un ou plusieurs fils. L'arbre se construit alors au fur et à mesure, en ajoutant des branches à chaque nouvelle génération. Un des avantages est que ce codage peut être utilisé dans les cas où les solutions n'ont pas de taille finie. En revanche, les solutions peuvent parfois être difficiles à interpréter, surtout lorsqu'on se retrouve avec des arbres de taille très importante.

## IV.3 Création de la population initiale

La population initiale sera créée de manière aléatoire à condition que chaque individu de la population créée soit une solution du problème. La population doit être non homogène et servira de base pour les générations futures.

## IV.4 Evaluation des individus

L'étape d'évaluation consiste à attribuer une valeur d'adaptation ou une 'note' à chaque individu selon sa performance. Il faudrait donc créer en fonction du problème étudié une fonction d'évaluation. Cette fonction, appelée « Fitness », permet de déterminer la qualité de



la solution représentée par chaque individu, si la valeur numérique trouvée est grande alors cet élément a plus de chance d'être choisi l'or de l'étape de sélection, notre but étant de rechercher les meilleurs individus de la population.

## IV.5 Sélection

L'opérateur de sélection va permettre d'identifier les meilleurs individus de la population générée et ainsi d'éliminer les plus mauvais.

Il existe plusieurs méthodes de sélection, les plus couramment utilisées sont les suivantes :

- **Sélection par roulette (wheel)**

Il faut imaginer une sorte de roulette de casino sur laquelle sont placés tous les individus de la population, la place accordée à chacun des individus étant en relation avec sa valeur d'adaptation. Ensuite, la bille est lancée et s'arrête sur un individu. Les meilleurs individus peuvent ainsi être tirés plusieurs fois et les plus mauvais ne jamais être sélectionnés.

- **Sélection par rang**

Ce mode de sélection choisit toujours les individus ayant les meilleures notes d'adaptation, sans laisser intervenir le hasard.

- **Sélection par tournoi**

Cette sélection consiste à prendre aléatoirement un nombre  $K$  d'individus et choisir le meilleur d'entre eux. On répète ce processus jusqu'à atteindre le nombre nécessaire d'individus à reproduire.

- **Sélection d'échantillonnage**

Consiste à associer à chaque individu un segment de longueur relative à sa note, les individus sélectionnés sont choisis par un ensemble de points équidistants.

- **Sélection Uniforme**

La sélection se fait aléatoirement, uniformément et sans intervention de la valeur d'adaptation. Chaque individu a donc une probabilité  $\frac{1}{P}$  d'être sélectionné, où  $P$  est le nombre total d'individus dans la population.

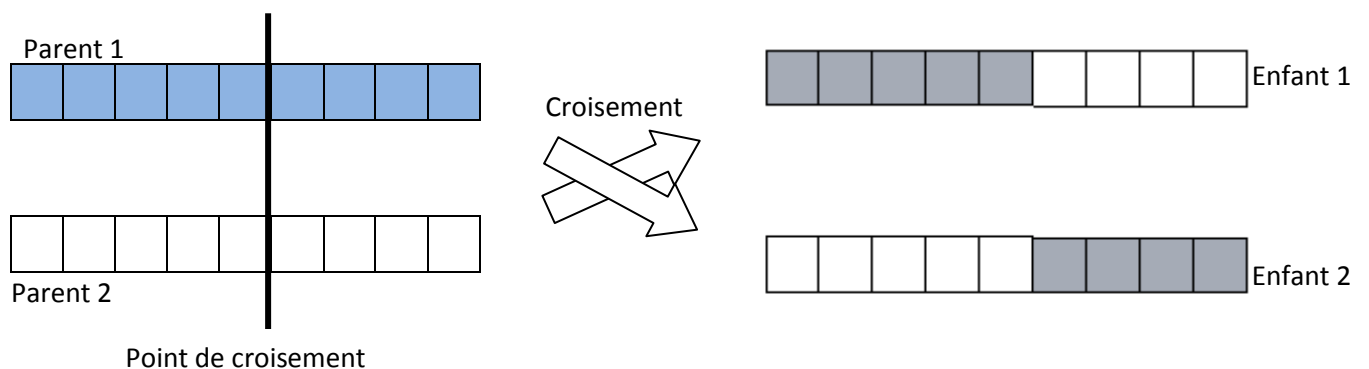
## IV.6 Reproduction

L'étape de reproduction est constituée de deux opérateurs permettant l'exploration et la diversité de la population. Ces opérateurs sont respectivement le croisement et la mutation.

### IV.6.1 Croisement

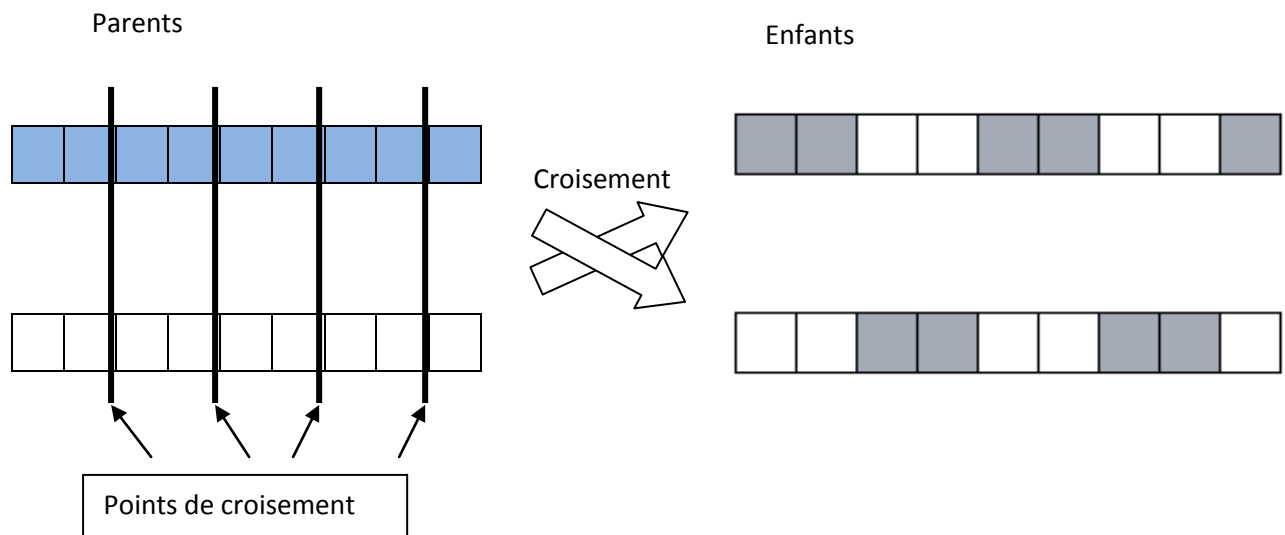
Lorsque la sélection est terminée, on prend deux individus (parents) à croiser pour obtenir deux nouveaux individus qu'on appelle enfants.

Les individus enfants sont créés comme montré dans la figure suivante :



La première étape consiste à choisir un point au hasard qui va servir au point de croisement. Le premier enfant est créé en commençant par la première partie du parent 1 et en terminant par la deuxième partie du parent 2. De même pour le deuxième enfant, il est créé en copiant la première partie du parent 2 et la deuxième partie du parent 1.

Dans l'exemple montré sur la figure précédente, nous n'avons choisi qu'un seul point de croisement, mais il est fréquent d'effectuer ce que l'on appelle le croisement à points multiples comme l'illustre la figure suivante :



Cet opérateur permet la création de deux nouveaux individus. Toutefois, un individu sélectionné lors de la reproduction ne subit pas nécessairement l'action d'un croisement. Ce dernier ne s'effectue qu'avec une certaine probabilité appelée probabilité de croisement. Plus cette probabilité est élevée et plus la population subira de changement. La probabilité de croisement  $pc$ , est généralement donnée par le programmeur. Les valeurs généralement admises sont comprises entre 0,5 et 0,9.

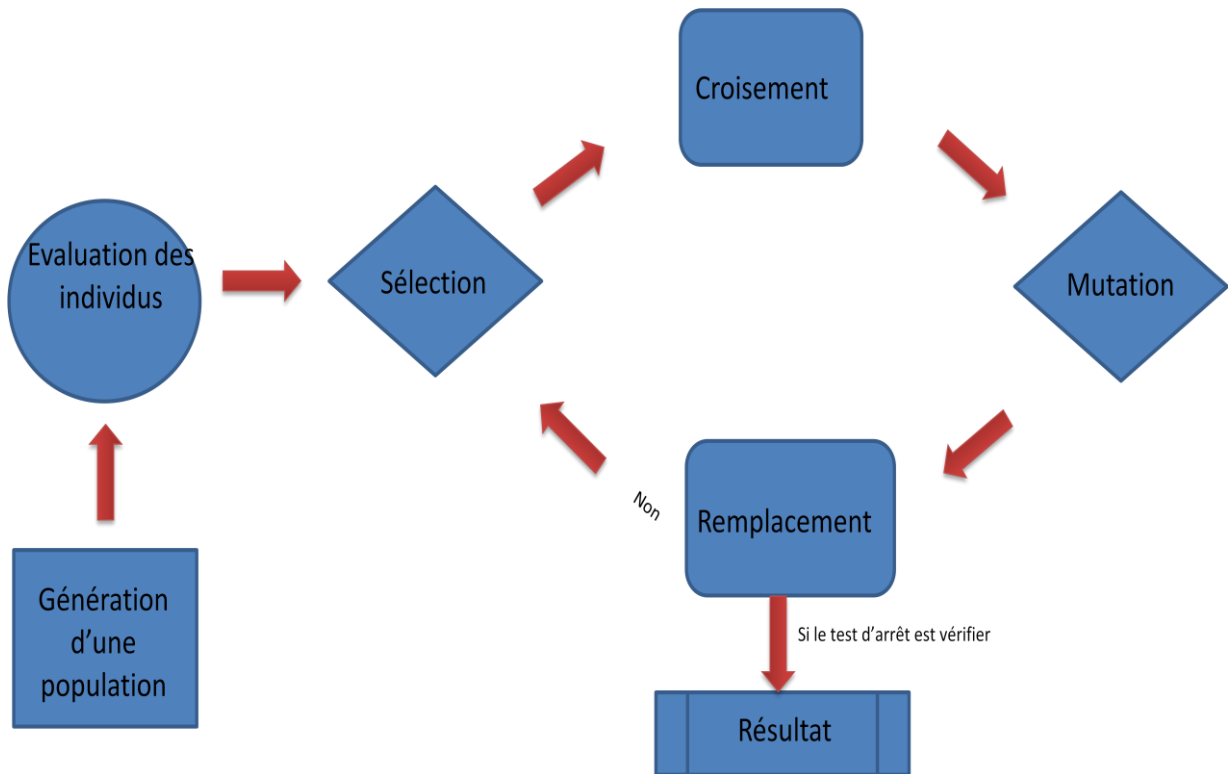
#### IV.6.2 Mutation

Le rôle de l'opérateur de mutation est de modifier aléatoirement, avec une certaine probabilité, la valeur d'un composant de l'individu. Dans le cas du codage binaire, chaque gène est remplacé ou permuté avec un autre gène selon une probabilité  $P_m$ . Tout comme plusieurs lieux de croisement peuvent être possibles, nous pouvons très bien admettre qu'une même chaîne puisse subir plusieurs mutations.

La probabilité de mutation  $P_m$  est généralement faible puisqu'un taux élevé risque de conduire à une solution sous-optimale.

## IV.7 Schéma récapitulatif

On peut résumer l'algorithme génétique par le schéma récapitulatif suivant :



---

# Chapitre III : Application de l'Algorithme Génétique aux problèmes de Bin - Packing

---

Dans ce chapitre, nous présentons deux applications de l'algorithme génétique : dans la première application, nous traitons le problème de sac à dos alors que la deuxième application concerne le problème de Bin Packing dans le cas d'une seule dimension. Nous avons programmé ces deux méthodes en utilisant le langage C, et avons créé une plateforme sous Visual C++. Nous avons testé ces programmes sur des données que nous avons générées aléatoirement. Pour le premier problème traité, nous présentons également la méthode de Brunch & Bound avec laquelle nous comparons les résultats obtenus par l'algorithme génétique.

La fenêtre principale de notre application nous donne le choix du problème à résoudre, pour résoudre le problème de sac à dos, il suffit de cliquer sur le bouton 1, et pour résoudre le problème de Bin – Packing, il suffit de cliquer sur le bouton 2.



- 1- Pour accéder au programme de Sac –à-dos
- 2- Pour accéder au programme de Bin-Packing

# **I. Application au problème de Sac – à – dos à une dimension**

Le problème du sac à dos fait partie des problèmes d'optimisation combinatoire les plus étudiés ces cinquante dernières années, en raison de ces nombreuses applications dans le monde réel. En effet, ce problème intervient souvent comme sous-problème à résoudre dans plusieurs domaines : la logistique comme le chargement d'avions ou de bateaux, l'économie comme la gestion de portefeuille ou dans l'industrie comme la découpe de matériaux.

De nos jours le problème du sac à dos se résout de manière assez efficace. Les approches proposées dans la littérature, pour résoudre ce problème, sont des méthodes exactes capables de résoudre un problème à l'optimalité ou des heuristiques qui fournissent une solution approchée de bonne qualité dans des temps de résolution très raisonnables.

Pour résoudre ce problème, nous proposons une application sous forme d'un programme en visual C++, qui étant donné la taille et la valeur d'un ensemble d'objets en plus de la taille maximale du sac, nous spécifie les objets à sélectionner.

Notre programme est basé sur l'algorithme génétique qui d'après notre observation, a donné un résultat proche de l'optimalité. Ce résultat a été comparé avec le résultat exact trouvé en appliquant la méthode de Branch & Bound.

## **I.1 Algorithme génétique**

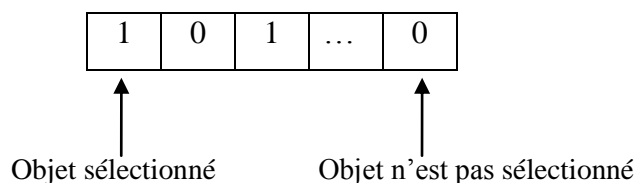
L'algorithme génétique que nous avons appliqué est hybride. Il est basé sur des opérateurs de croisement et de mutation spécifiques et bien adaptés au problème de Bin-Packing. Nous présentons dans la suite la description de toutes les procédures composant cet algorithme.

### **I.1.1 Codage**

Dans le cas général, il n'y a pas de représentation absolue. L'encodage se fait selon les caractéristiques du problème. Ainsi l'encodage d'un problème peut aller de la simple chaîne binaire, à un arbre en passant par les listes chaînées.

Dans notre algorithme, nous utilisons un codage binaire. Nous représentons une solution par une chaîne de bits, dont chaque élément représente un objet avec la notation suivante :

- 1 si l'objet est mis dans le sac
- 0 sinon



### I.1.2 Sélection

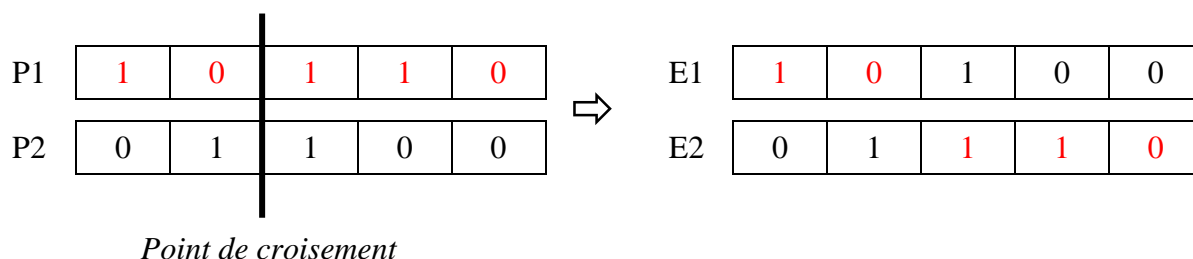
L'opérateur de sélection a pour mission de choisir dans la population présente les futurs parents nécessaires à l'étape de reproduction.

Dans notre algorithme, nous avons choisi d'utiliser la sélection uniforme, qui consiste à choisir aléatoirement et uniformément les individus à croiser.

### I.1.3 Croisement

Le croisement des individus peut être toute opération permettant d'obtenir un nouvel individu à partir de plusieurs individus existants. Nous avons choisi d'effectuer le croisement à un point en reprenant des parties de chaque individu parent pour créer un nouvel individu enfant.

Exemple : Considérons les deux parents suivant :



Dans certains cas, il se peut que le croisement fournisse des enfants qui représentent des solutions non réalisables pour notre problème, autrement dit, le poids total des objets mis dans le sac dépasse sa capacité. Pour remédier à cela, nous proposons un croisement à un point avec correction. La démarche de ce codage est la suivante :

Nous commençons par copier la première partie du parent 1 dans le premier enfant, et on complète par les gènes du deuxième parent en calculant à chaque insertion le poids total des

objets. Si à l'ajout d'un nouvel objet on dépasse la capacité du sac, alors dans ce cas, le gène correspond prendra la valeur 0. Et inversement pour le deuxième enfant.

### I.1.4 Mutation

La mutation d'un individu peut être toute opération aléatoire. On peut par exemple permuter un ou plusieurs couples d'objets, ou tout simplement changer la valeur d'un ou plusieurs gènes. Dans notre algorithme, nous utilisons un type de mutation, qui consiste à changer aléatoirement les gènes des individus ayant une probabilité de mutation très faible (inférieur à 0,2).

## I.2 Plate forme

La plate forme que nous présentons a été réalisée sous Visual Basic. Cette boîte de dialogue est composée de plusieurs éléments que nous décrivons un par un.

The image shows a Visual Basic dialog box titled "Sac-à-dos" with a yellow star icon. It contains several input fields and buttons, each labeled with a circled number:

- 1: "Taille de l'objet :" text label above a text box.
- 2: "Valeur de l'objet :" text label above a text box.
- 3: "liste des objets :" text label above a list box.
- 4: "Ajouter" button.
- 5: "Capacité du Sac :" text label above a text box.
- 6: "Enregistrer" button.
- 7: "Résultat" button.

There is also a "Valeur" label next to the list box and a "Résultat" label next to the "Résultat" button.

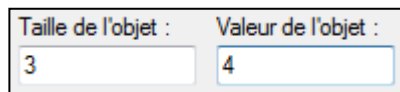
- 1- Champ consacré à la saisie de la taille de l'objet
- 2- Champ consacré à la saisie de la valeur de l'objet



- 3- La liste des objets, elle contient deux colonnes (la colonne à gauche : pour afficher les tailles des objets, alors que la colonne à droite est pour leurs valeurs)
- 4- Pour ajouter un autre objet à la liste
- 5- Champ consacré à la saisie de la taille, la capacité du sac
- 6- Pour enregistrer les données dans un fichier, afin de trouver une solution approché au problème
- 7- Pour afficher le résultat donné par le programme

### I.2.1 Les étapes de l'exécution

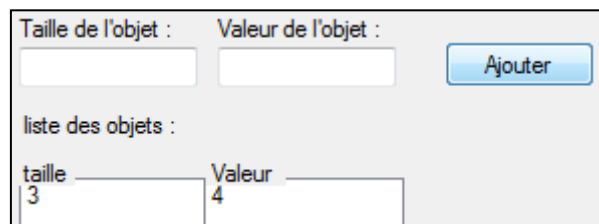
**Etape 1** - Donner la taille de l'objet dans le champ 1 nommé « Taille de l'objet », puis juste à coté dans 2, le champ suivant « Valeur de l'objet », donner la valeur du même objet.



A screenshot of a form with two input fields. The first field is labeled 'Taille de l'objet :' and contains the number '3'. The second field is labeled 'Valeur de l'objet :' and contains the number '4'.

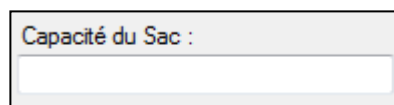
Dans cet exemple nous avons donné un objet qui possède une taille de 3 est une valeur de 4.

**Etape 2** - Cliquer sur le « Ajouter ». Dans ce cas, l'objet saisi sera afficher dans la liste des objets, pour ajouter un autre objet il suffit de répéter la même opération en revenant à la première étape,



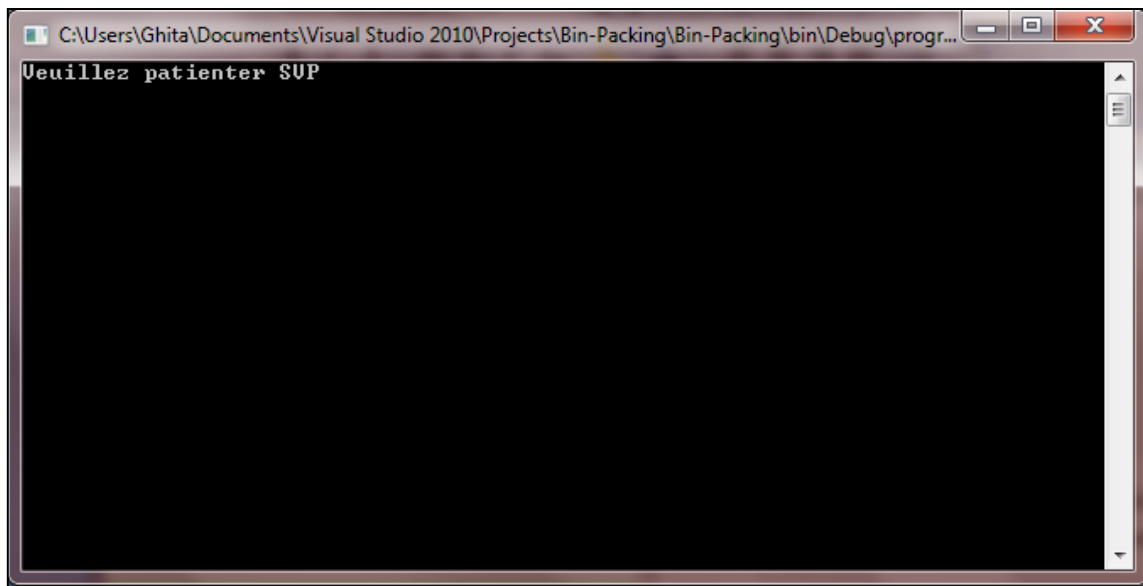
A screenshot of the application interface. At the top, there are two input fields for 'Taille de l'objet :' and 'Valeur de l'objet :', both empty. To the right of these fields is a blue button labeled 'Ajouter'. Below the input fields is a section titled 'liste des objets :'. Under this title, there is a table with two columns: 'taille' and 'Valeur'. The first row of the table contains the values '3' and '4' respectively.

**Etape 3** – Une fois que la saisie de tous les objets est terminée, il faut saisir la capacité maximale du sac à ne pas dépasser.

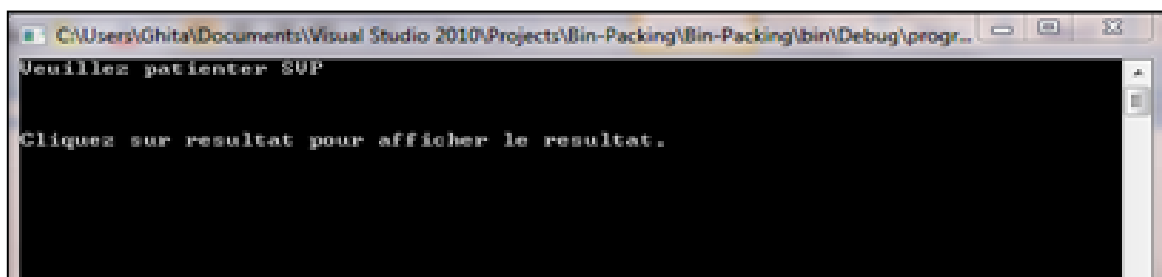


A screenshot of a form with a single input field labeled 'Capacité du Sac :'. The field is empty.

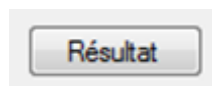
**Etape 4** – Appuyer sur le bouton « Enregistrer », pour lancer la recherche de la solution, cela se fait dans un autre programme alors une autre fenêtre (noir) s'ouvrera.



**Etape 4** - Attendre jusqu'à ce que le message suivant « Cliquer sur résultat pour afficher le résultat » s'affichera à l'écran.

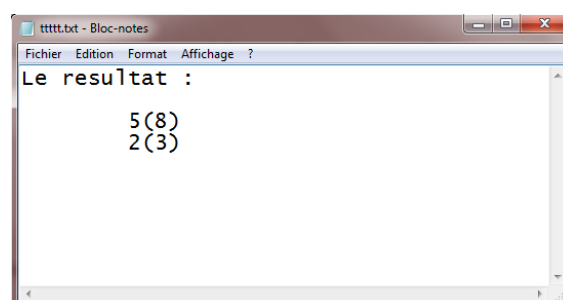


**Etape 5** - Pour afficher le résultat, il suffit de cliquer sur le bouton « Résultat »



et un fichier texte s'ouvrira, qui affichera seulement les objets sélectionnés sous la forme suivante :  $P(V)$ , où  $P$  est le poids de l'objet et  $V$  sa valeur.

Exemple :



C'est-à-dire que dans notre sac on a deux objets :

le 1<sup>er</sup> est l'objet qui possède une taille de 5 et une valeur de 8

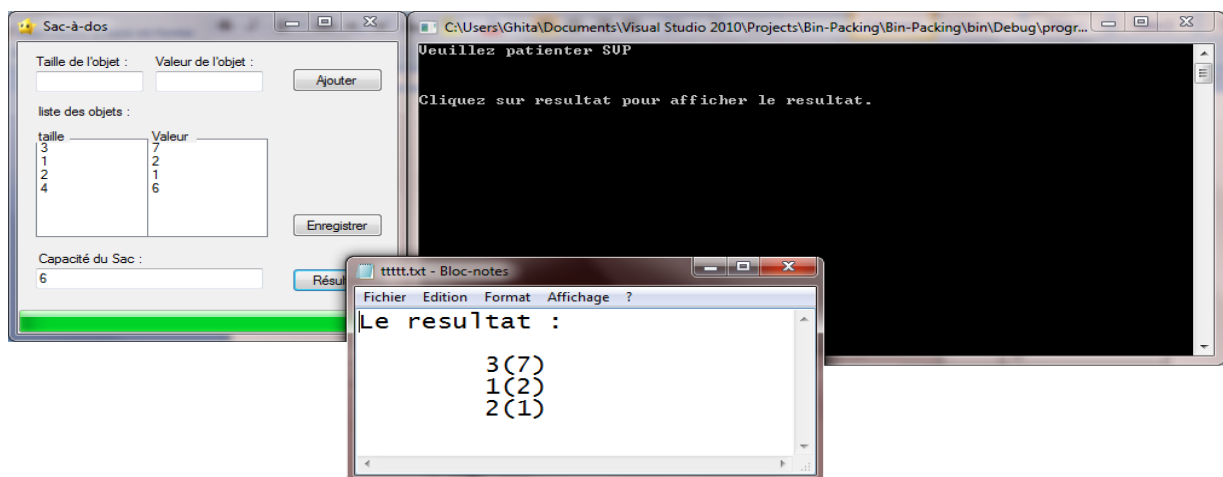
Le 2<sup>ème</sup> est l'objet qui a la taille 2 et la valeur 3

### I.2.2 Exemple d'exécution

Considérons le problème de Sac-à-dos suivant, où les poids et les valeurs des objets sont présentés au tableau suivant :

	1	2	3	4
Pi (poids)	3	1	2	4
Wi (valeur)	7	2	1	6

Avec  $P = 6$  le poids maximal du sac



Pour vérifier la performance de ce programme, nous allons comparer ce résultat trouver par ce dernier, avec le résultat exact qu'on trouvera en appliquant la méthode de Branch & Bound.

### I.3 Brunch&Bound

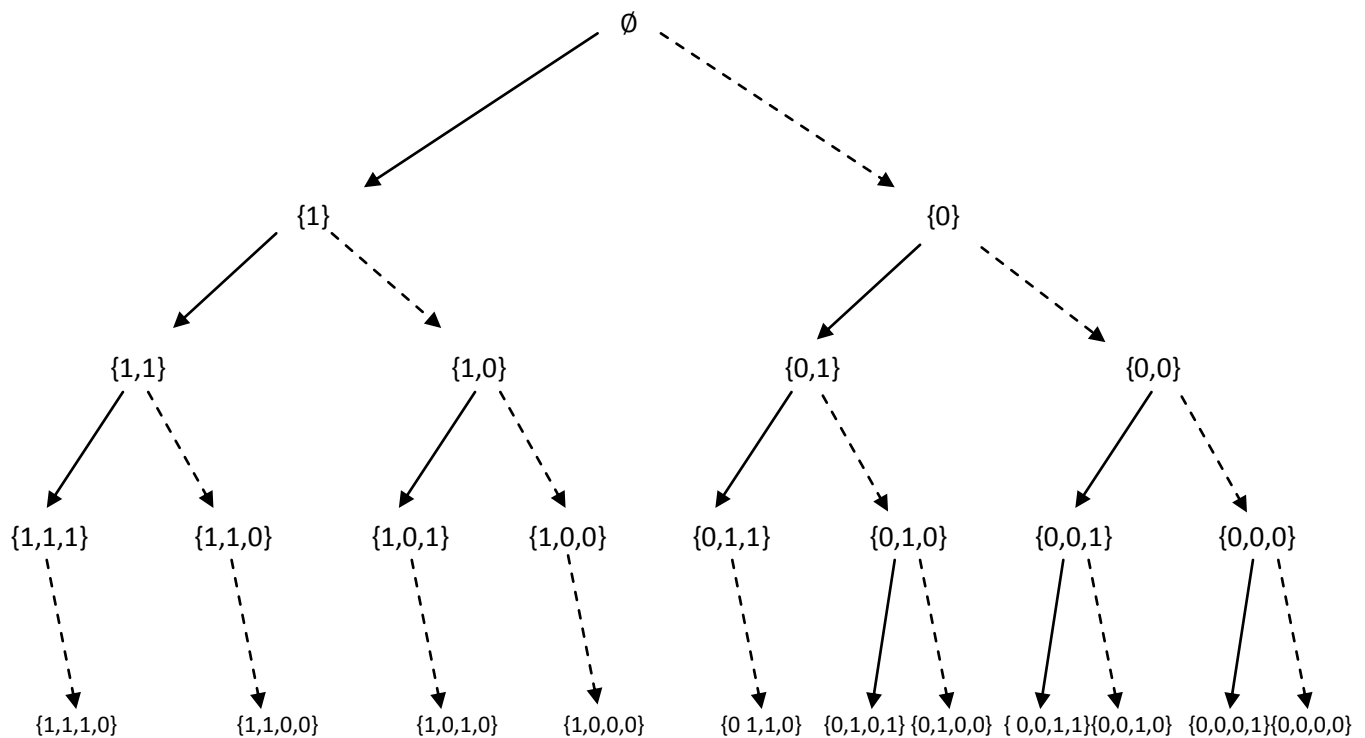
Branch and Bound est un terme anglais qui veut dire « Séparation et Evaluation », c'est une méthode générique de résolution de problèmes d'optimisation et plus particulièrement d'optimisation combinatoire ou discrète. C'est une méthode d'énumération implicite, toutes les solutions possibles du problème peuvent être énumérées, mais l'analyse des propriétés du problème permet d'éviter l'énumération de larges classes de mauvaises solutions.

En effet, Dans les méthodes par séparation et évaluation, la séparation permet d'obtenir une méthode générique pour énumérer toutes les solutions tandis que l'évaluation évite l'énumération systématique de toutes les solutions.

Etape 1 : Séparation (Création de l'arbre de recherche)

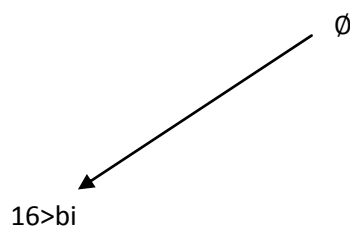
Notre arbre de recherche sera représenté de la façon suivante :

- Chaque niveau correspond à un objet
- Chaque nœud représente une étape de construction de la solution
- Chaque arc représente les choix faits pour construire la solution (arc peut prendre la valeur 1 si l'objet est sélectionné, et 0 sinon)



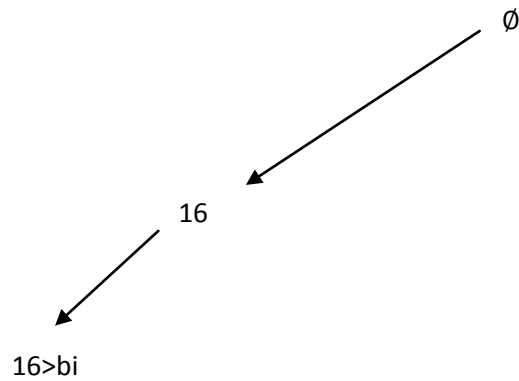
Etape 2 : parcours de l'arbre et le calcul des bornes (la borne inférieure est initialisée par 0  $b_i=0$ ),

Itération 1 :

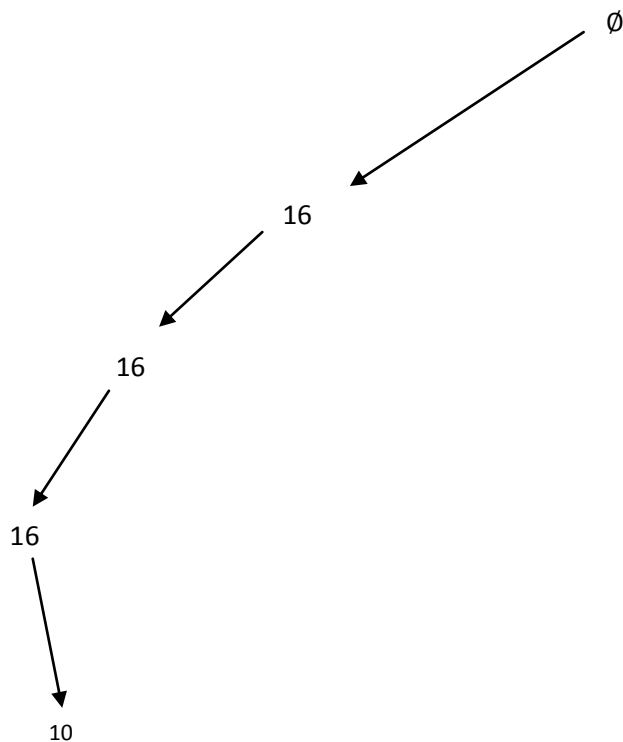


La borne supérieure des deux nœuds est supérieure à la borne inférieure, et on passe au sommet fils, et on refait le même travail,

Itération 2 :

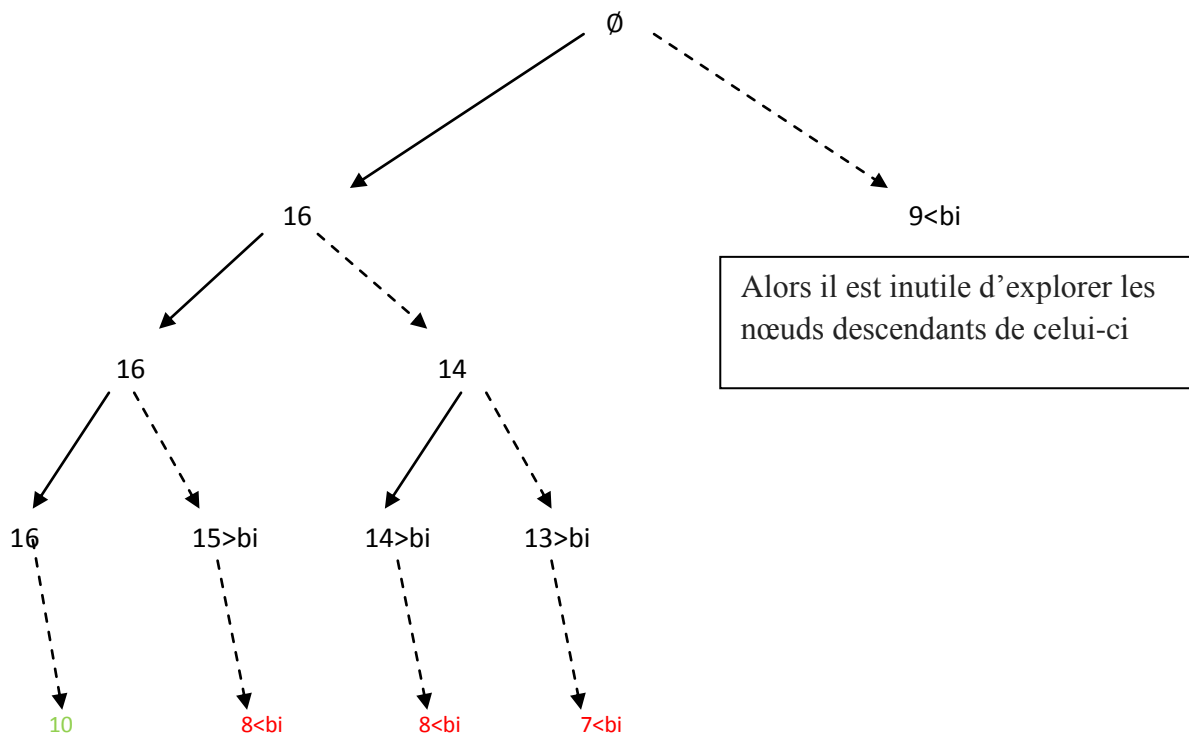


Et on continue de la même manière jusqu'à ce qu'on arrive à un nœud feuille



La borne supérieure de ce nœud feuille est supérieure à la valeur de la borne inférieure actuelle, donc la valeur de la borne inférieure devient 10 ( $b_i=10$ )

Itération k :



La solution optimale est {1, 1, 1, 0}

Nous remarquons que les résultats trouvés sont les mêmes, de là on peut dire que notre objectif est atteint.

## II. Application au problème de Bin – Packing à une dimension

De nos jours le problème du Bin - Packing à une dimension, se résout de manière assez efficace. Beaucoup d'approches sont proposées dans la littérature, pour résoudre ce problème, que se soit des méthodes exactes capables de fournir une solution optimale du problème, ou des heuristiques qui fournissent une solution approchée de bonne qualité dans des temps de résolution très raisonnables.

Pour résoudre ce type de problème, nous proposons une application sous forme d'un programme en visual C++, qui étant donné la taille d'un ensemble d'objets et la taille des bins, spécifie le rangement qu'il faut adopter.

Notre programme est une généralisation du problème de sac – à – dos défini dans la première partie de ce chapitre, les objets seront rangés dans les bins en deux étapes :

- 1<sup>ère</sup> étape : On cherche la solution du problème de sac – à – dos suivant :

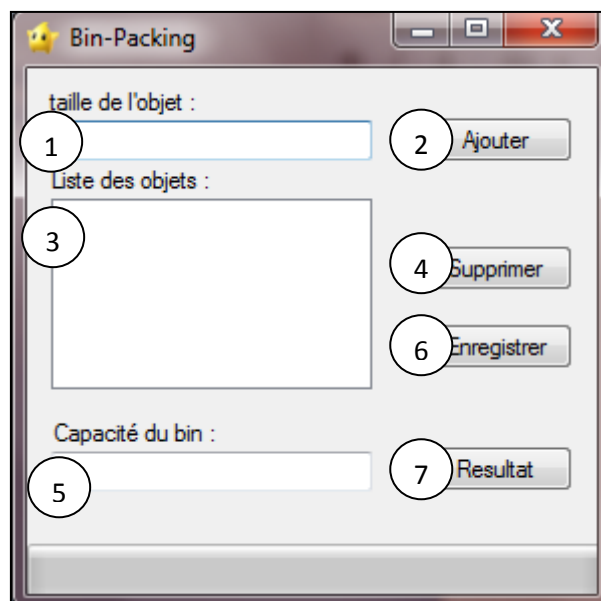
Etant donné un ensemble d'objets qui possèdent des tailles différentes et des bins de tailles identiques, notre objectif est de mettre le maximum d'objets possible dans un bin, autrement dit, on veut maximiser l'espace utilisé dans chaque bin. Dans ce cas la valeur de chaque objet est sa taille.

- 2<sup>ème</sup> étape : Lorsqu'on trouve le résultat de ce problème en utilisant le programme précédant, on range les objets sélectionnés dans le bin en cours et on supprime ces objets de la liste initiale des objets. On revient à l'étape précédente jusqu'à ce que tous les objets de notre liste soient rangés dans les bins.

De manière formelle notre programme est en visual C++, il est basé sur les algorithmes génétiques, qui donnent des résultats approchés.

## II.1 Plat forme

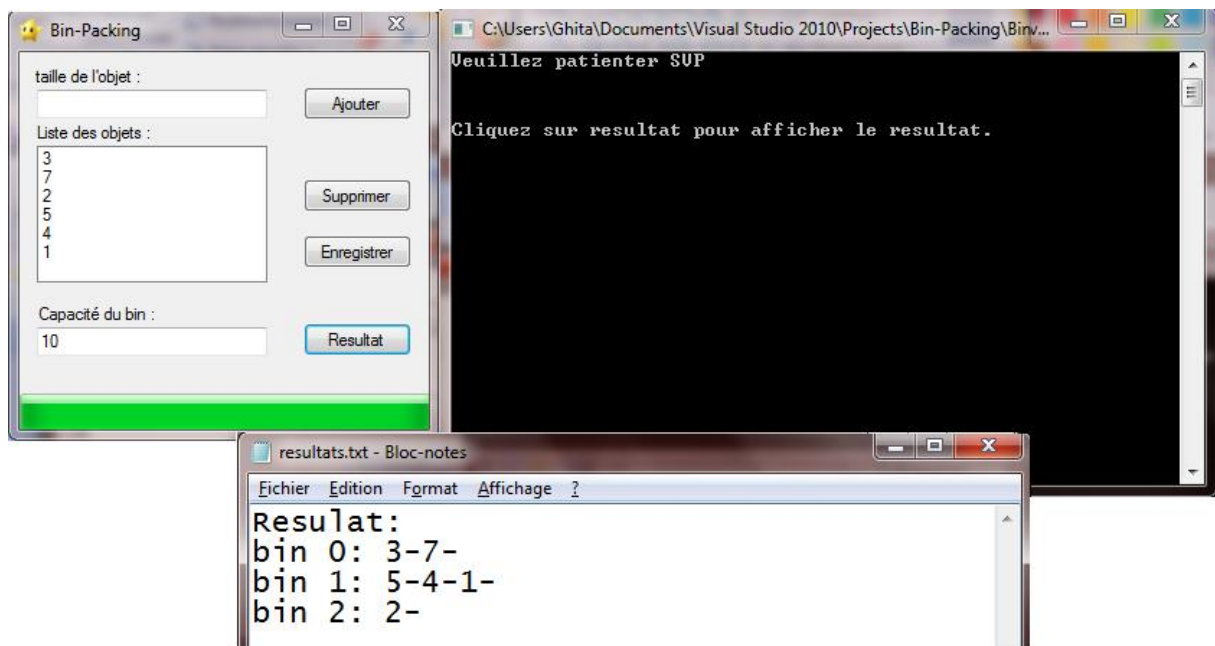
La plate forme que nous présentons a été réalisée sous Visual Basic. Cette boite de dialogue est composée de plusieurs éléments que nous décrivons un par un.



- 1- Pour donner la taille de l'objet
- 2- Pour ajouter l'objet dans la liste
- 3- La liste des objets
- 4- Pour supprimer un objet
- 5- Pour donner la taille du bin
- 6- Pour enregistrer les données

7- Pour afficher le résultat

## II.2 Exemple d'exécution :





---

## Conclusion générale

---

Dans ce mémoire nous avons traité le problème de bin-packing à une et à deux dimensions. L'importance de ce problème réside dans le domaine vaste de ses applications concrètes en transport, logistique et dans le monde industriel.

Nous avons étudié quelques modèles mathématiques correspondant à ces problèmes, puis un aperçu de certains travaux aboutis de la littérature, sur les problèmes de Bin – Packing à une et à deux dimensions. Nous avons ensuite adopté deux méthodes de résolution du problème de bin packing à une dimension. La première méthode est une application des algorithmes génétiques qui est une méthode approchée alors que la deuxième est une méthode exacte à savoir la méthode de Brunch & Bound.

Nous avons appliqué et programmé notre approche génétique en Langage C et nous avons créé deux plates formes sous Visual Basic permettant la résolution du problème de Sac—à-dos où l'objectif est de maximiser la valeur totale des objets mis dans le sac tout en respectant la capacité du sac, et nous avons généralisé ce programme de Sac – à – dos afin de résoudre des problèmes de Bin – Packing.





---

# BIBLIOGRAPHIE

---

- 1- BAUMGARTNER Lukas, Verena Schmid, and Christian Blum. Solving the Two-Dimensional Bin Packing Problem with a Probabilistic Multi-start Heuristic. 2011
- 2- BEN MOHAMED Ahmed. Résolution approchée du problème de bin-packing, 2009
- 3- YOUSSEF BENADADA – Ahmed EL HILALI ALAOUI. Programmation mathématique de la modélisation à la résolution, 2012
- 4- BENCHEIKH Ghizlane. Problèmes de transport Modélisation et résolution par méthaheuristiques. 2009
- 5- DREO Johann – PÉTROWSKI Alain – SIARRY Patrick – TAILLARD Eric. Métaheuristiques pour l'optimisation difficile
- 6- EL HAYEK Joseph, Le problème de bin-packing en deux-dimensions, le cas non-orienté : résolution approchée et bornes inférieures, 2007
- 7- EL HILALI ALOUI Ahmed – EL KHOUKI Fatima– BENCHEIKH Ghizlane. Introduction à la recherche opérationnelle. 2009
- 8- ESSEGHIR LALAMI Mohammed. Contribution à la résolution de problèmes d'optimisation combinatoire : méthodes séquentielles et parallèles. 2012
- 9- GUERET Christelle – PRINS Christian – SEVAUX Marc. Programmation linéaire.
- 10- JONCOUR Cédric. Problèmes de placement 2D et application à l'ordonnancement modélisation par la théorie des graphes et approches de programmation mathématique, 2010
- 11- KEBE Mohamed. Résolution d'un problème de tournées de véhicules avec contraintes de placement, 2009
- 12- KHANAFER Ali. Algorithmes pour résoudre des problèmes de Bin Packing mono- et multi – objets. 2010
- 13- Andrea LODI, Silvano MARTELLO, Daniele VIGO. Recent advances on two-dimensional bin packing Problems. 2001
- 14- Lodi Andrea. Algorithms for Two-Dimensional Bin Packing and Assignment Problems. 1999
- 15- LODI Andrea, Silvano MARTELLO, Daniele VIGO. Models and Bounds for Two-Dimensional Level Packing Problems,

- 16- MINOUX Michel. Progammmation mathématique théorie et algorithmes, 1991
- 17- PUCHINGER Jakob and R. Raidl GUNTHER. Models and algorithms for three- stage two dimensional Bin Packing. 2004
- 18- WATTEBLED Paméla, Résolution heuristique d'un problème de conception de modules automatiques de rangement, 2010