République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohamed El Bachir El Ibrahimi Bordj Bou Arréridj

Faculté de Mathématique et Informatique

Département de Recherche Opérationnelle

Mémoire de fin d'études

En vue de l'obtention du Diplôme de Master

Domaine : de Recherche Opérationnelle

Filière : Mathématiques appliquées

Spécialité : méthodes et outils pour la recherche Opérationnelle

Thème

Résolution du problème Bin Packing en utilisant L'algorithme FireFly

Présenté par :

- **TAYEB HAMMANI** ***Kenza***
- **BELFERKOUS** ***Fatima***

Président :	M. Zouache Djaafer		Université de BBA
Examinateur :	M. Saifi AbdelHamid		Université de BBA
Rapporteur :	Naili MAKHLOUF	MCB	Université de BBA

Année universitaire : 2020/2021

Remerciements

*Je tiens à remercier en premier lieu **ALLAH,** le tout Puissant de m'avoir donné courage, santé et patience pour achever ce travail (**ELHAMDOU LILLAH**).*

Nous tenons à exprimer nos remerciements à notre promoteur monsieur

***NAILI MAKHLOUF** qui a mis toute sa compétence à notre disposition et pour son suivi régulier à l'élaboration de ce modeste travail.*

*Nous désirons exprimer notre profonde et vive reconnaissance à M **ZOUACHE DJAAFER** d'avoir accepté de présider le jury de soutenance.*

*Nous adressons un grand merci à M^me^ **SAIFI ABDELHAMID** pour l'honneur qu'il nous a fait en acceptant à examiner ce mémoire.*

C'est avec un réel plaisir que j'adresse mes sincères reconnaissances et ma profonde gratitude à tous ceux qui ont m'aidé de près ou de loin pour réaliser ce travail.

Enfin, on tiens aussi à remercier mes camarades qui ont participé au bon déroulement de ce mémoire.

Dédicace

Avant tout : louanges à ***Dieu****, élément et miséricordieux pour nous avoir donné la force de mener à terme ce travail.*

Je dédie cet humble travail à mes chers parents, mais aucune dédicace ne serait témoin de mon profond amour, mon immense gratitude et mon plus respect ; À mon très cher père «BRAHIM »qui m'a toujours soutenu, et a été toujours présent pour moi.

À la plus chère au monde, ma mère «HAYETTE» la lumière de ma vie pour son amour, qui m'a toujours encouragé durant toutes mes années d'études.

À mes fleurs sœurs AHLEM pour leur disponibilité, leur soutien moral, leur encouragement incessant, d'être coopératif et d'assumer à ma place certaine de mes responsabilités familiales.

À mon très cher petit frère YOUNES.

À mes chers enseignants sans exception.

À toutes les promotions de 2^{éme} année Master Recherche Opérationnelle LMD, ainsi que toute personne qui m'aime et me respecte.

KENZA

Dédicace

Je dédie ce travail

À ma mère « Ghania ». Pour son amour, ses encouragements et ses sacrifiées

À mon père « Aziz ».Pour son soutien, son affection qu'il m'accordé

À mon mari « moussa ». Pour son soutien moral et leurs conseils précieux tout au long de mes études

À tous les membres de ma famille

À tous mes amis et surtout mon collègue « Kenza » et ma sœur« Amina »

Et tous ceux qui m'aiment

FATIMA

Liste des tableaux

Liste des algorithmes

Listes des figures

Liste d’abréviation

OC : Optimisation Combinatoire.

POC : Problème Optimisation Combinatoire.

PBP : Problème Bin Packing.

POMO : Problème d'Optimisation Multi-Objectifs.

PVC : Problème du Voyageur de Commerce.

TSP: Traveling Salesman Problem.

KP: Knapsack Problem.

POMO : Problème d'Optimisation Multi-Objectifs.

MD: Multi-Objectifs Dominance.

B&B: Branch-and-Bound.

AG : Algorithmes Génétiques.

SA : Recuit Simulé.

PSO : Particules Optimisation d'essaim.

HS : Harmonie Recherche.

AF : Algorithme FireFly.

PBP : Problème Bin Packing.

PBP-1D : Problème Bin Packing Une Dimensions.

PBP-2D : Problème Bin Packing Deux Dimensions.

PBP -3D: Problème Bin Packing Trio Dimensions.

PSE : Procédure de Séparation et Evaluation.

FF: First-Fit.

NF: Next Fit.

BF: Best Fit.

FFD: First-Fit Decreasing.

BFD: Best-Fit Decreasing.

FNF: Finite Next-Fit.

FFF: Finite First-Fit.

BL: Bottom-Left.

NFDH: Next-Fit Decreasing Height.

FFDH: First-Fit Decreasing Height.

BFDH: Best-Fit Decreasing Height.

La recherche opérationnelle est la discipline des mathématiques appliquées liée à l'informatique, qui traite des questions d'utilisation optimale des ressources dans l'industrie et dans le secteur public.

L'optimisation combinatoire occupe une place très importante en recherche opérationnelle. Son importance se justifie d'une part par la grande difficulté des problèmes d'optimisation et d'autre part par de nombreuses applications pratiques pouvant être formulées sous la forme d'un problème d'optimisation combinatoire. Les problèmes d'optimisation sont utilisés pour modéliser de nombreux problèmes dans différents secteurs de l'industrie (télécommunications, électronique, mécanique, chimie, transport, …).

Problématique

Problème Bin Packing il y a des objets ayant des volumes différents, qui doivent être emballés dans des boîtes. L'objectif est de remplir le nombre fini de boîtes avec les objets donnés d'une manière qui minimise le nombre de boîtes utilisés. L'objectif principal du problème est d'utiliser efficacement l'espace et le temps.

Étant donné un ensemble des objets avec des poids différents et un nombre illimité de boîtes avec une capacité de boîtes fixe, l'objectif du problème est d'emballer ces objets au nombre minimum de boîtes de telle sorte que le poids total des objets affectés à une boîte ne dépasse pas la capacité des boîtes. Il existe de nombreuses variantes de ce problème, telles que bin packing 1D, bin packing 2D, bin packing 3D, etc.

Objectif

Modélisation et résolution d'un problème de bin packing qui concerne et d'emballer les objets de poids différents dans un nombre fini de boîtes sans dépasser sa capacité, et Résolvez-le en utilisant l'algorithme de FireFly puis de comparer les résultats obtenues.

Organisation du mémoire

Afin de bien présenter notre travail nous l'avons structuré de la manière suivante

Dans le premier chapitre, donne un aperçu général sur les notions liées à l'optimisation (définition, classification et exemples) et Concept optimisation multi-objectif.

Dans le deuxième chapitre sera consacré à la présentation des méthodes de résolution des problèmes d'optimisation combinatoire.

Dans le troisième chapitre est consacré aux problèmes de bin packing à uni- et bi- et tri-dimensionnel, notamment leurs modélisations et les méthodes de résolutions.

Dans le quatrième chapitre, nous donnants l'implémentation et les résultats numérique de la programmation de bin packing par l'algorithme FireFly.

Chapitre I:

Etat de l’art

I.1. Introduction

L'optimisation combinatoire (OC) occupe une place très importante en recherche opérationnelle et en informatique. De nombreuses applications pouvant être modélisées sous la forme d'un problème d'optimisation combinatoire (POC) telles que le problème du voyageur de commerce, Problème de sac à dos, le problème de transport, etc. Le but de ce domaine est de résoudre plusieurs problèmes d'optimisation combinatoire difficiles à résoudre.

I.2. Problèmes d'optimisation combinatoire

I.2.1. Définition

Un problème d'optimisation se définit comme la recherche du minimum ou du maximum (de l'optimum donc) d'une fonction donnée. On peut aussi trouver des problèmes d'optimisation pour lesquels les variables de la fonction à optimiser sont contraintes d'évoluer dans une certaine partie de l'espace de recherche. **[1]**

L'optimisation peut signifier beaucoup de choses différentes. Il est possible d'écrire un problème d'optimisation sous la forme générique :

$$\begin{cases} \min f(x), & & (\text{I}.1) \\ g_j(x) = 0 & j \in I_1 & (\text{I}.2) \\ g_k(x) \leq 0 & k \in I_2 & (\text{I}.3) \end{cases}$$

x : Le vecteur constitué des variables de décision, avec $x = (x_1, \ldots, x_n) \in R_n$,

f : La fonction objectif réelle,

g : Le vecteur des fonctions intervenant dans les contraintes.

Où I_1, I_2 sont les ensembles d'indices des contraintes de types égalités et inégalités respectivement. Soit $X = \{x \in R_n : g_j(x) = 0, i \in I_1, g_k(x) \leq 0, k \in I_2\}$.

I.2.2. Vocabulaire et définitions

Fonction objectif : C'est le nom donné à la fonction f (on l'appelle encore **fonction de coût** ou **critère d'optimisation).** C'est cette fonction que l'algorithme d'optimisation va devoir « optimiser » (trouver un optimum). [1]

Variables de décision : Elles sont regroupées dans le vecteur x. C'est en faisant varier ce vecteur que l'on recherche un optimum de la fonction f. [1]

Minimum global : Un « point » est x^* un minimum global de la fonction f si on a :

$f(x^*) < f(x)$ quel que soit x tel que $x^* \neq x$. Cette définition correspond au point M_3 de la **Fig 1.** [1]

Minimum local fort : Un « point » x^*est un minimum local fort de la fonction f si on a :

$f(x^*) < f(x)$ quel que soit $x^* \in V(x^*)$ et $x^* \neq x$, où $V(x^*)$ définit un « voisinage » de x^*. Cette définition correspond aux points M_2 et M_4 de la **Fig 1.** [1]

Minimum local faible : Un « point » x^*est un minimum local faible de la fonction f si on a :

$f(x^*) \leq f(x)$ quel que soit $x^* \in V(x^*)$ et $x^* \neq x$, où $V(x^*)$ définit un « voisinage » de x^*. Cette définition correspond au point M_1 de la **Fig 1.** [1]

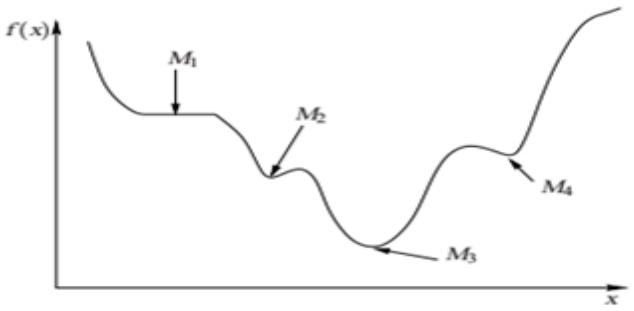


Figure I.1. Les différents minimaux.

I.2.3. Classification des problèmes d'optimisation

En gros, la classification peut être effectuée en nombre d'objectifs, nombre de contraintes, formes fonctionnelles, paysage des fonctions objectifs, type de variables de conception, incertitude dans valeurs et l'effort de calcul (voir la **Fig. 2**).

En nombre d'objectifs

Si nous essayons de classer les problèmes d'optimisation selon le nombre d'objectifs, alors il y a deux catégories : objectif unique M= 1 et multi-objectifs M > 1. L'optimisation multi-objectif est également appelée multicritère ou même l'optimisation multi-attributs dans la littérature. Optimisation la plus réelle les problèmes sont multi-objectifs. Par exemple, lors de la conception d'un moteur de voiture, nous vouloir maximiser

son rendement énergétique, minimiser les émissions de dioxyde de carbone, et d'abaisser son niveau sonore. Ici, nous avons trois objectifs. Parfois, ces les objectifs sont souvent contradictoires et un certain compromis est nécessaire. Par exemple, lorsque nous louons ou achetons une maison, nous la voulons dans le meilleur emplacement possible avec un grand espace alors que nous avons l'intention de dépenser le moins possible.

En nombre des contraintes

De même, nous pouvons également classer l'optimisation en termes de nombre de contraintes J+K S'il n'y a aucune contrainte $J = K = 0$, alors on l'appelle un problème d'optimisation sans contrainte. Si $K = 0$ et $J \geq 1$, on l'appelle un problème d'égalité, alors que $J = 0$ et $K \geq 1$ devient un problème d'inégalité problème contraint. Il convient de souligner que dans certaines formulations de littérature d'optimisation, les égalités ne sont pas explicitement incluses et seules les inégalités sont incluses. C'est parce qu'une égalité peut être écrite comme deux inégalités. Par exemple $g(x) = 0$ est équivalent à $g(x) \leq 0$ et $g(x) \geq 0$.

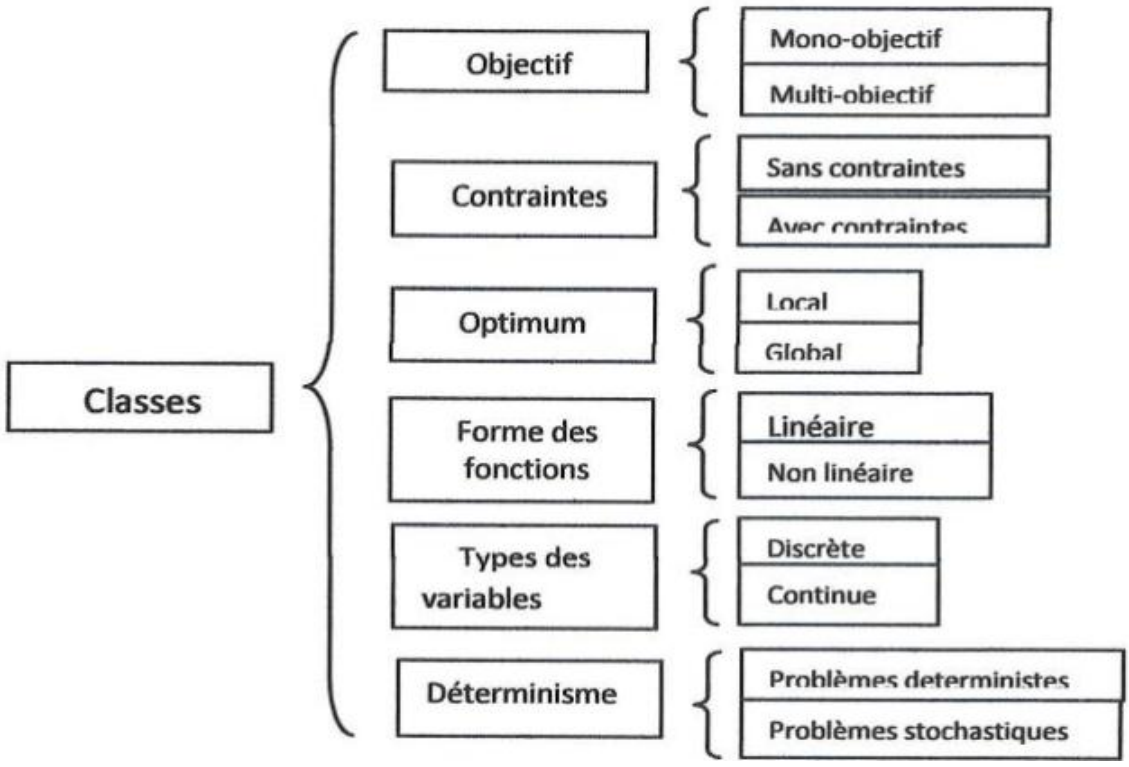


Figure I.2. Classification des problèmes d'optimisation. [2]

Forme des fonctions

Nous pouvons également utiliser les formes de fonction réelles pour la classification. L'objectif les fonctions peuvent être linéaires ou non linéaires. Si les contraintes g_j et g_k sont tout linéaire, alors cela devient un problème linéairement contraint. Si les deux contraintes et les fonctions objectifs sont toutes linéaires, Cela devient un problème de programmation linéaire. Ici, la « programmation » n'a rien à voir avec le programme informatique; cela signifie planification et/ou optimisation. Cependant, d'une manière générale, tous les f_i, $g_j$et g_k sont non linéaires, nous devons faire face à un problème d'optimisation non linéaire.

Optimum

Nous pouvons aussi classer l'optimisation en termes de le paysage de fonctions objectifs. Pour une seule fonction objectif, la forme ou le paysage peut varier considérablement pour le cas non linéaire. Si il n'y a qu'une seule vallée ou pic avec un optimum global unique, alors le problème d'optimisation est dit uni-modal. Dans ce cas, l'optimum local est le seul optimum, qui est aussi son optimum global. Par exemple, $f(x, y) = x^2 + y^2$ est unimodale, car le minimum local à (0, 0) est le global minimum. Une classe spéciale et très importante d'optimisation uni-modale est l'optimisation convexe, lorsque les fonctions objectives ont une certaine convexité et que la solution globale optimale est garantie. Dans l'optimisation de l'ingénierie, nous avons l'intention de concevoir ou de reformuler le problème en termes de formes matricielles quadratiques souvent convexes, et donc leurs solutions optimales sont les meilleures au monde. Cependant, la plupart des objectifs les fonctions ont plus d'un mode, et de telles fonctions multimodales sont beaucoup plus difficiles à résoudre. Par exemple $f(x, y) = \cos(x)\cos(y)$ est multimodal.

Types des variables

Le type de valeur des variables de conception peut également être utilisé pour la classification. Si les valeurs de toutes les variables de conception sont discrètes, alors l'optimisation est appelée optimisation discrète. Si toutes ces valeurs sont des nombres entiers, alors cela devient un problème de programmation en nombres entiers. En gros, optimisation discrète est également appelée optimisation combinatoire, bien que certaines publications préfère déclarer que l'optimisation discrète consiste en une programmation entière et une optimisation combinatoire. L'optimisation combinatoire est de loin le type d'optimisation le

plus populaire et elle est souvent liée à la théorie des graphes et au routage, comme le problème du voyageur de commerce, le problème de l'arbre couvrant minimum, le routage des véhicules, la planification des compagnies aériennes et le problème du sac à dos. De plus, si toutes les variables de conception sont continués ou prennent des valeurs réelles dans certains intervalle, l'optimisation est appelée problème d'optimisation continue. Cependant, les variables de conception dans de nombreux problèmes peuvent être à la fois discrètes et continue, nous appelons alors une telle optimisation le type mixte.

Déterministe

Dans le monde réel, il y a aussi une incertitude et du bruit dans les variables de conception et fonctions objectif et/ou contraintes, alors l'optimisation devient un problème d'optimisation, ou un problème d'optimisation robuste avec bruit. Pour la plupart problèmes stochastiques, nous devons les redéfinir de manière à ce qu'ils deviennent significatifs lors de l'utilisation de techniques d'optimisation standard [2].

I.2.4. Exemples de problème d'optimisation combinatoire

Il y a plusieurs problèmes d'optimisation combinatoire, parmi ces problèmes on peut citer :

- Problème de voyageur de commerce.
- Problème de sac à dos.
- Problèmes de coloration de graphes.
- Problème de bin packing.
- Problème de propagation de virus.
- Problème d'affectation.
- Problème de couverture d'ensemble.
- Problème de localisation.

I.2.4.1. Le problème du voyageur de commerce ou PVC

Le PVC (Traveling Salesman Problem ou TSP) se formule ainsi : étant donné un ensemble I de n villes, dans quel ordre un voyageur de commerce doit-il les parcourir pour que, partant et arrivant dans la même ville, il minimise la distance totale parcoure ?

On écrit le problème du voyageur de commerce comme suit :

Données :

$I = \{1, \dots, n\}$: Ensemble des villes

$h_{i,j}, \forall\ (i,j) \in I \times I$: Coût relatif au déplacement de la ville *i* vers la ville *j*

Variables :

$$x_{i,j} = \begin{cases} 1\ si\ la\ ville\ i\ précède\ la\ ville\ j \\ 0\ sinon \end{cases}, \forall\ (i,j) \in I \times I$$

Minimiser :

$$\sum_{i \in I} \sum_{j \in I} h_{i,j} \times x_{i,j} \qquad (I.4)$$

Sous les contraintes :

$$\sum_{j \in I} x_{i,j} = 1, \quad \forall i \in I \qquad (I.5)$$

$$\sum_{i \in I} x_{i,j} = 1, \quad \forall j \in I \qquad (I.6)$$

$$\sum_{i \in S} \sum_{j \in S} x_{i,j} \leq Card(S) - 1, \forall\ S \subset I, 1 \leq Card(S) \leq \left\lfloor \frac{n}{2} \right\rfloor \qquad (I.7)$$

La fonction objectif (I.4) qui est la minimisation de la longueur totale parcourue par le voyageur, les contraintes (I.5) et (I.6) indiquent que le voyageur passe une et une seule fois par chaque sommet (client). Dans ce modèle, $h_{i,j}$ représente le coût (en termes de distance) pour aller de la ville i vers la ville j. En résolvant le problème d'affectation et en visualisant les solutions obtenues, on s'aperçoit de la présence de sous-cycles (en général en grands nombres). Les contraintes(I.7), qui sont propres au TSP et qui rendent ce problème non polynomial, permettent d'éliminer ces sous-cycles. [3]

Figure I.3. Le problème du voyageur de commerce.

I.2.4.2. Problème du sac à dos (Knapsack Problem ou KP)

Ce problème a l'avantage d'avoir une formalisation mathématique très simple, ce qui en fait un bon exemple d'introduction à la programmation linéaire en nombres entiers. On dispose d'un ensemble d'objets I. Chaque objet i est caractérisé par sa masse m_i et sa valeur v_i. Nous devons décider lesquels nous emportons dans notre sac à dos, sachant que la masse totale que nous pouvons cherchons à maximiser la valeur des objets contenus dans le sac .Concernant le programme linéaire, chaque objet i est associé à une variable x_i qui stipule si oui ou non l'objet est dans le sac. La fonction objectif $(I.8)$maximise la valeur globale des objets contenus dans le sac. La contrainte $(I.9)$exprime que la capacité du sac est respectée. [3]

On écrit le problème du sac à dos comme suit :

Données :

$I = \{1, \dots, n\}$: Ensemble des n objets

$m_i, \forall\, i \in I$: Masse de l'objet i

$v_i, \forall\, i \in I$: Valeur de l'objet i

C : capacité du sac

Variables :

$$x_i = \begin{cases} 1\ si\ l'objet\ i\ est\ dans\ le\ sac \\ 0\ sinon \end{cases} \qquad \forall i \in I$$

Maximiser :

$$\sum_{i \in I} v_i x_i \qquad (I.8)$$

Sous les contraintes :

$$\sum_{i \in I} m_i x_i \leq C \qquad (I.9)$$

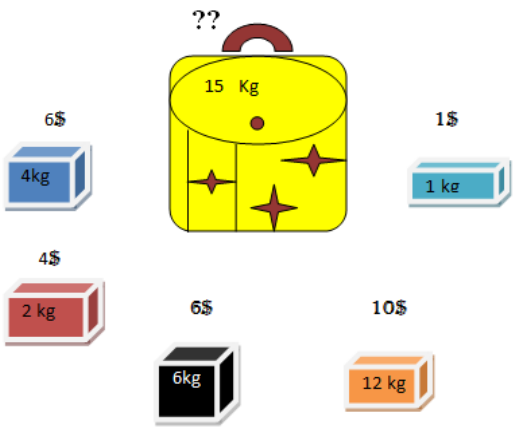


Figure I.4. Exemple d'un sac à dos.

I.2.4.3. Problème d'affectation

Le problème d'affectation consiste à établir des liens entre les éléments de deux ensembles distincts, de façon à minimiser un coût et en respectant des contraintes d'unicité de lien pour chaque élément.

Supposons que, dans une entreprise, n ouvriers puissent travailler indifféremment sur n machines, mais avec plus ou moins d'efficacité ; l'efficacité peut se mesurer par le revenu provenant de la vente des produits fabriqués par les divers ouvriers travaillant sur les différentes machines.

Répartir les ouvriers sur les machines de la façon la plus efficace est un problème d'affectation.

Soit n unités à affecter à n tâches ; une unité ne peut être affectée qu'à une tâche et une tâche ne peut employer qu'une unité. L'unité i exécute la tâche j avec un coût ou un profit c_{ij}.

On défini $x_{ij} = 1$ si i est affecté à j, $x_{ij} = 0$ autrement.

$$min\ Z = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \qquad (I.10)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i = 1,2,\dots,n$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j = 1,2,\dots,n$$

$$x_{ij} = 1\ ou\ 0 \qquad \forall i,j$$

I.3. Optimisation multi objectif

I.3.1. Définition

Le problème d'optimisation multi-objectifs (POMO) peut être défini comme le problème de trouver un vecteur de variables décision x, qui optimise un vecteur de M fonctions $f_i(x)$, où i = 1, 2, . . ., M; soumis à contraintes d'inégalité $g_j(x) \geq 0$, et contraintes d'égalité $g_k(x) = 0$, où $j = 1,2,\dots,J$ et $k = 1,2,\dots,K$. Les fonctions objectifs forment une description mathématique de la performance critères qui sont généralement en conflit les uns avec les autres. [4]

Sans perte de généralité, un POMO peut être défini comme suit :

$$\begin{cases} Maximiser\ \{f_1(x), f_2(x), \dots, f_M(x)\} & (\text{I}.11) \\ g_j(x) \geq 0;\ \ j = 1,2, \dots, J & (\text{I}.12) \\ g_k(x) = 0;\ \ k = 1,2, \dots, K & (\text{I}.13) \end{cases}$$

où x est le vecteur des variables de décision ; $f_i(x)$ est le i-ième fonction objectif; et $g_j(x)$) et $g_k(x)$ sont des vecteurs de contraintes. Ces fonctions objectifs constituent un ensemble multidimensionnel espace en plus de l'espace décisionnel habituel. Ce supplément l'espace est appelé l'espace objectif Z. Pour chaque solution x dans l'espace des variables de décision, il existe un point dans l'espace objectif :

$$f(x) = Z = (z_1, z_2, \dots, z_M)^T$$

Dans un problème d'optimisation multi-objectif, nous souhaitons trouver un ensemble de valeurs pour les variables de décision qui optimise un ensemble de fonctions objectifes. Un vecteur de décision x est dit dominer un vecteur de décision y (aussi écrit $x \succ y$) si et seulement si:

$$f_i(x) \geq f_i(y) \quad \forall i \in \{1,2, \dots, M\} ;$$

Et

$$\exists\ i \in \{1,2, \dots, M\}\ \ |f_i(x) \geq f_i(y)|.$$

Tous les vecteurs de décision qui ne sont dominés par aucun autre vecteur de décision sont dits non dominés ou Pareto-optimaux et constituent le Front Pareto-optimal. Ce sont des solutions pour lesquelles aucun objectif ne peut être amélioré sans s'écartant d'au moins un autre objectif.

I.3.2. Dominance et notion d'optimalité

Dans le domaine de l'optimisation multi objectif, le décideur évalue généralement une solution par rapport à chacun des critères, et se positionne donc naturellement dans l'espace objectif. Néanmoins, contrairement au cas mono-objectif où il existe un ordre total sur $\mathbb{R}$ parmi les solutions réalisables, il n'existe généralement pas de solution qui serait à la fois optimale pour chaque objectif, étant donnée la nature conflictuelle de ces derniers. Ainsi, une relation de dominance, d'ordre partiel, est généralement définie.

Définition 3.2.1. (Dominance Pareto) Un vecteur objectif $z \in Z$ domine un vecteur objectif $z' \in Z$ si et seulement si $\forall i \in \{1,2,\ldots,n\}$, $z_i \leq z'_i$ et $\exists j \in \{1,2,\ldots,n\}$ tel que $z_j < z'_j$. Cette relation sera notée $z \succ z'$. Par extension, une solution $x \in X$ domine une solution $x' \in X$, Noté $x \succ x'$, ssi $f(x) \succ f(x')$.

Définition 3.2.2. (Vecteur non-dominé) Un vecteur objectif $z \in Z$ est non-dominé si et seulement si $\forall z' \in Z, z' \not\succ z$.

Définition 3.2.3. (Solution Pareto optimale) Une solution $x \in X$ est Pareto optimale (ou non-dominée) si et seulement si $\forall x' \in X, x' \not\succ x$.

Définition 3.2.4. (Dominance faible) Un vecteur objectif $z \in Z$ domine faiblement un vecteur objectif $z' \in Z$ si et seulement si $\forall i \in \{1,2,\ldots,n\}$, $z_i \leq z'_i$. Cette relation sera notée $z \succcurlyeq z'$.

Définition 3.2.5. (Point idéal) Le point idéal $z^* = (z^*_1, z^*_2, \ldots, z^*_n)$ est le vecteur qui minimise chaque fonction objectif individuellement, $z^*_i = min_{x \in X} f_i(x)$ pour $i \in \{1,2,\ldots,n\}$.

I.3.3. Bute de multi objective

Le but de l'optimisation multi-objectifs est de trouver la solution unique donnant le meilleur compromis entre plusieurs objectifs. Comme il n'existe généralement pas de solution unique qui optimise simultanément tous les objectifs, la sélection de la meilleure solution de compromis nécessite de prendre en compte les préférences du MD. Hypothèses très faibles et généralement acceptées sur les préférences du MD la meilleure solution de compromis appartient à l'ensemble des solutions dites efficaces [5]. Ainsi, de nombreuses méthodes d'optimisation multi-objectif réduisent l'espace de recherche à l'ensemble des solutions efficaces. Notez que cette approche n'est pas valide si le MD recherche un échantillon des meilleures solutions car les deuxièmes meilleures solutions et d'autres bonnes solutions n'ont pas besoin d'être efficaces sous les mêmes hypothèses sur les préférences du MD.

Conclusion

Dans ce chapitre nous avons présenté le concept fondamental d'optimisation combinatoire et les classifications dans ce problème. Ensuite Nous présentons des problèmes classiques d'optimisation combinatoire : le problème du voyageur de commerce, le problème du sac-à-dos, le problème d'affectation. En fin on a définir l'optimisation multi-objectif consiste à optimiser simultanément plusieurs fonctions et on a vu la notion l'ensemble de solutions Pareto optimales les problèmes d'optimisation multi-objectif.

Chapitre II:

Les méthodes de résolutions

II.1. Introduction

Un grand nombre de méthodes ont été développées pour tenter d'apporter une réponse satisfaisante aux problèmes optimisation. Parmi celles-ci, nous distinguons les méthodes dédiées à un problème spécifique et les méthodes plus générales, pouvant s'appliquer à un ensemble de problèmes. Parmi ces méthodes, nous relevons deux grandes classes de méthodes: les méthodes exactes et les méthodes approchées.

Dans ce chapitre, nous présentons en particulier La méthode séparation et évaluation pour les méthodes exactes et FireFly pour les méthodes approchées**.**

II.2. Méthodes de résolutions

En ingénierie, les problèmes d'optimisation combinatoire consistent à sélectionner la meilleure méthode de résolution, il existe deux classes de méthodes pour résoudre ces problèmes.

Les méthodes exactes : sont les méthodes qui garantissent l'optimalité de la solution obtenue. Si la taille des données est importante, les méthodes exactes ne permettant pas d'avoir un temps de calcule raisonnable. Parmi les méthodes exactes sont branch-and-bound (B&B), la programmation dynamique, Méthodes de relaxation lagrangiennes et programmation linéaire et entière méthodes basées, telles que branche-et-coupe, branche-et-prix, et branche-et-coupe-et prix [6].

Les méthodes approchées : ne garantissent pas que la solution qu'elles fournissent soit optimale, elles sont fréquemment utilisées pour résoudre de problèmes qui sont souvent trop difficiles pour être résolus avec des méthodes exactes. Parmi ces méthodes, on trouve les méthodes heuristiques /méta-heuristiques comprennent, entre autres, le recuit simulé [7], la recherche tabu [8], recherche locale itérative [9], recherche de quartier variable [10], et diverse population basée modèles tels que les algorithmes évolutifs [11], la recherche par dispersion [12], et diverses estimations d'algorithmes de distribution [13].

II.2.1. Méthodes de résolutions exacte

Nous présentons d'abord quelques méthodes de la classe des algorithmes complets ou exacts, ces méthodes donnent une garantie de trouver la solution optimale pour une instance de taille finie dans un temps limité et de prouver son optimalité**.** [14]

II.2.1.1. La méthode de séparation et évaluation (Branch and Bound)

L'algorithme de séparation et évaluation, plus connu sous son appellation anglaise Branch and Bound (B&B) [15], repose sur une méthode arborescente de recherche d'une solution optimale par séparations et évaluations, en représentant les états solutions par un arbre d'états, avec des nœuds, et des feuilles. Le branch-and-bound est basé sur trois axes principaux :

- L'évaluation,
- La séparation,
- La stratégie de parcours.

L'évaluation

L'évaluation permet de réduire l'espace de recherche en éliminant quelques sous-ensembles qui ne contiennent pas la solution optimale. L'objectif est d'essayer d'évaluer l'intérêt de l'exploration d'un sous-ensemble de l'arborescence. Le branch-and-bound utilise une élimination de branches dans l'arborescence de recherche de la manière suivante : la recherche d'une solution de coût minimal, consiste à mémoriser la solution de plus bas coût rencontré pendant l'exploration, et à comparer le coût de chaque nœud parcouru à celui de la meilleure solution. Si le coût du nœud considéré est supérieur au meilleur coût, on arrête l'exploration de la branche et toutes les solutions de cette branche seront nécessairement de coût plus élevé que la meilleure solution déjà trouvée.

La séparation

La séparation consiste à diviser le problème en sous-problèmes. Ainsi, en résolvant tous les sous-problèmes et en gardant la meilleure solution trouvée, on est assuré d'avoir résolu le problème initial. Cela revient à construire un arbre permettant d'énumérer toutes les solutions. L'ensemble de nœuds de l'arbre qu'il reste encore à parcourir comme étant susceptibles de contenir une solution optimale, c'est-à-dire encore à diviser, est appelé ensemble des nœuds actifs.

La stratégie de parcours

- **La largeur d'abord :** Cette stratégie favorise les sommets les plus proches de la racine en faisant moins de séparations du problème initial. Elle est moins efficace que les deux autres stratégies présentées.

- **La profondeur d'abord :** Cette stratégie avantage les sommets les plus éloignés de la racine (de profondeur la plus élevée) en appliquant plus de séparations au problème initial. Cette voie mène rapidement à une solution optimale en économisant la mémoire.

- **Le meilleur d'abord :** Cette stratégie consiste à explorer des sous problèmes possédant la meilleure borne. Elle permet aussi d'éviter l'exploration de tous les sous-problèmes qui possèdent une mauvaise évaluation par rapport à la valeur optimale.

II.2.2. Heuristiques

En optimisation combinatoire, une heuristique est un algorithme approché qui permet d'identifier en temps polynomial au moins une solution réalisable rapide, pas obligatoirement optimale. L'usage d'une heuristique est efficace pour calculer une solution approchée d'un problème et ainsi accélérer le processus de résolution exacte. Généralement une heuristique est conçue pour un problème particulier, en s'appuyant sur sa structure propre sans offrir aucune garantit quant à la qualité de la solution calculée. Les heuristiques peuvent être classées en deux catégories :

- **Méthodes constructives** qui génèrent des solutions à partir d'une solution initiale en essayant d'en ajouter petit à petit des éléments jusqu'à ce qu'une solution complète soit obtenue.
- **Méthodes de fouilles locales** qui démarrent avec une solution initialement complète (probablement moins intéressante), et de manière répétitive essaie d'améliorer cette solution en explorant son voisinage.

II.2.3. Méta-heuristiques

Si certaines heuristiques sont spécifiques à un problème, d'autres ont pour vocation de pouvoir être adaptées à divers problèmes. On appelle parfois ces dernières des méta-heuristiques ou encore heuristiques générales. En fait, il s'agit de principes algorithmiques permettant d'obtenir une solution en respectant certains principes de construction.

Les principales sont les méthodes gloutonnes ; les méthodes de recherche locale (ou d'améliorations itératives) telle que la méthode tabou (ou ses améliorations comme Grasp) ou les méthodes de descentes (recuit simulé) ; les méthodes évolutives (algorithmes génétiques) ; et la simulation (objet ou continue).

Algorithmes gloutons

Les algorithmes gloutons (greedy algorilhms en anglais) sont des algorithmes pour lesquels, à chaque itération, on fixe la valeur d'une (ou plusieurs) des variables décrivant le problème sans remettre en cause les choix antérieurs.

Le principe est donc de partir d'une solution incomplète (éventuellement totalement indéterminée) que l'on complète de proche en proche en effectuant des choix définitifs : à chaque étape, on traite (on « mange ») une partie des variables sur lesquelles on ne revient plus.

Exemples les algorithmes de méta-heuristique

En gros, les algorithmes méta-heuristiques modernes pour l'ingénierie optimisation comprennent les algorithmes génétiques (AG), le recuit simulé (SA), les particules optimisation d'essaim (PSO), algorithme de colonie de fourmis, algorithme d'abeille, harmonie recherche (HS), algorithme FireFly (AF) et bien d'autres.

II.2.3.1. Algorithme FireFly

Inspiration

Les lucioles(en anglais FireFly) sont de petits coléoptères ailés capables de produire une lumière clignotante froide pour une attraction mutuelle. Les femelles peuvent imiter les signaux lumineux d'autres espèces afin d'attirer des mâles qu'elles capturent et dévorent. Les lucioles ont un mécanisme de type condensateur, qui se décharge lentement jusqu'à ce que certain seuil est atteint, ils libèrent l'énergie sous forme de lumière. Le phénomène se répète de façon cyclique. L'Algorithme de lucioles développé par [16] est inspiré par l'atténuation de la lumière sur la distance et l'attraction mutuelle mais il considère toutes les lucioles comme unisexes.

II.2.3.1.1. Les lucioles naturelles

Il y a près de deux mille espèces connues de lucioles dans nature, dont la plupart émettent des éclairs de lumière avec un certain afin d'attirer un partenaire ou un appât. En outre pour ces raisons, les lucioles peuvent se protéger contre les attaquants utilisant les flashs qui peuvent également attirer le contraire sexe. La distance entre les lucioles et l'environnement, où la lumière est émise, est en quelque sorte efficace sur l'intensité de la lumière reçue par les lucioles. Comme l'intensité de la lumière obéit à la loi carrée inverse à une distance particulière$r$ ($I \propto 1/r^2$), et parce que la lumière est absorbée par l'air, la plupart des lucioles peuvent juste être visible à une distance limitée.

Figure II.5. Une luciole naturelle.

II.2.3.1.2. Algorithme FireFly

L'algorithme FireFly (AF) a été développé pour la première fois par Xin-She Yang fin 2007 et 2008 à l'Université de Cambridge [16, 17], qui était basé sur les modèles de clignotement et le comportement des lucioles. La littérature AF s'est considérablement développée au cours des 5 dernières années avec plusieurs centaines d'articles publiés sur les algorithmes de luciole, et Fister et al. Ont fourni un examen complet [18]. En substance, AF utilise les trois règles idéalisées suivantes :

1) Les lucioles sont unisexes de sorte qu'une luciole sera attirée par d'autres lucioles quel que soit leur sexe.
2) L'attractivité est proportionnelle à la luminosité, et ils diminuent tous les deux à mesure que leur distance augmente. Ainsi, pour n'importe quelles deux lucioles clignotantes, la moins brillante se déplacera vers la plus brillante. S'il n'y en a pas de plus brillant qu'une luciole en particulier, elle se déplacera au hasard.
3) La luminosité d'une luciole est déterminée par le paysage de la fonction objectif.

Initialisation :
u_i: i Lucioles, $i \in [1, n]$
n : le nombre de Lucioles
MaxGeneration : le maximum nombre de génération
γ: coefficient d'absorption
r : la distance entre deux firefly
d: la dimension

Debut

Définir une fonction objective $f(x), x = (x_1, x_2, \dots, x_d)^T$
Générer une population de Lucioles x_i $(i = 1, 2, \dots, n)$
Définir l'intensité de lumière I à un point x_i par la fonction objective $f(x_i)$
Déterminer le coefficient d'absorption γ
Tant que ($t < MaxGeneration$)

Pour $i = 1$ **jusqu'à** n

Pour $j = 1$ **jusqu'à** n

Si ($I_i < I_j$)

Déplacer la Lucioles i vers la Lucioles j
Fin Si
Varier l'attraction en fonction de la distance r via $exp[-\gamma r2]$
Evaluation des nouvelles solutions et mettre à jour l'intensité de lumière

Fin Pour *j*

Fin Pour *i*
Classer les Lucioles et trouver la meilleure solution

Fin Tant que

Fin

Algorithme II.1. Algorithme Original de luciole.

Dans l'algorithme FireFly, la variation de l'intensité lumineuse et la formulation de l'attractivité jouent un rôle essentiel. L'intensité de la lumière ou luminosité $I(r_{ij})$ est inversement proportionnelle au carré de la distance r_{ij} [19,20] et la luminosité relative de chaque luciole est exprimée sous la forme gaussienne suivante :

$$I(r_{ij}) = I_0 e^{-\lambda r_{ij}^2}$$

Où, $I(r_{ij})$ est l'intensité lumineuse à une distance r_{ij},I_0 est la luminosité maximale (la luminosité absolue au point source r_{ij}= 0) qui est lié à la valeur de la fonction objectif. La valeur la plus élevée de la fonction objectif est la valeur I_0 et λ est le coefficient d'absorption de la lumière, qui est réglé pour refléter que la luminosité augmente progressivement avec l'augmentation de la distance et l'absorption du milieu r_{ij} est la distance euclidienne entre la luciole i et la luciole j. L'attractivité de chaque luciole [21] s'exprime sous la forme :

$$\beta(r_{ij}) = \beta_0 e^{-\lambda r_{ij}^2}$$

Où, β_0 est l'attractivité maximale (l'attractivité à r_{ij}= 0, la plus grande valeur de la luciole pour en attirer une autre, est généralement fixée à 1). Cependant, sur le plan informatique, le calcul de $1 / (1 + \lambda r_{ij}^2)$ est plus facile que $e^{-\lambda r_{ij}^2}$ [19] et l'intensité peut s'écrire :

$$I(r_{ij}) = \frac{I_0}{1 + \lambda r_{ij}^2}$$

De même, l'attractivité d'une luciole peut être approximée comme suit :

$$\beta(r_{ij}) = \frac{\beta_0}{1 + \lambda r_{ij}^2}$$

Distance et mouvement

On suppose une luciole située en $x_i = (x_1^i, x_2^i, \dots, x_k^i)$ est plus brillant qu'une autre luciole située à $x_j = (x_1^j, x_2^j, \dots, x_k^j)$, la luciole située à x_i se déplacera vers x_j. La distance entre deux lucioles i et j en x_i et x_j est la distance euclidienne donnée par [19,20] comme suit :

$$r_{ij} = \| x_i - x_j \| = \sqrt{\sum_k^d (x_{i,k} - x_{j,k})^2}$$

Où, d est la dimension, $x_{i,k}$ est la $k^{\text{ème}}$ composante de la coordonnée spatiale x_i de la $i^{\text{ème}}$ luciole le mouvement d'une luciole i est attiré par une autre luciole plus attrayante j et l'emplacement de mise à jour est déterminé par :

$$x_{i+1} = x_i + \beta_0 e^{-\lambda r_{ij}^2}(x_j - x_i) + \alpha(rand - \frac{1}{2})$$

Le premier terme est la position actuelle d'une luciole [20], le second est utilisé pour considérer l'attractivité d'une luciole à l'intensité lumineuse vue par les lucioles adjacentes et le troisième est utilisé pour le mouvement aléatoire d'une luciole au cas où il n'y en aurait pas de plus lumineuse. Le coefficient α est un paramètre de randomisation déterminé par le problème d'intérêt, tandis que rand est un nombre aléatoire tiré d'une distribution gaussienne ou d'une distribution uniforme à l'instant t, si $\beta_0 = 0$ cela devient une simple marche aléatoire. Dans l'implémentation de l'algorithme nous utiliserons $\beta_0 = 0$, $\alpha = 0.25$ et le coefficient d'attractivité ou d'absorption $\lambda = 1$ qui garantit une convergence rapide de l'algorithme vers la solution optimale. Le concept de l'algorithme à base de luciole est présenté à la **Fig 6.**

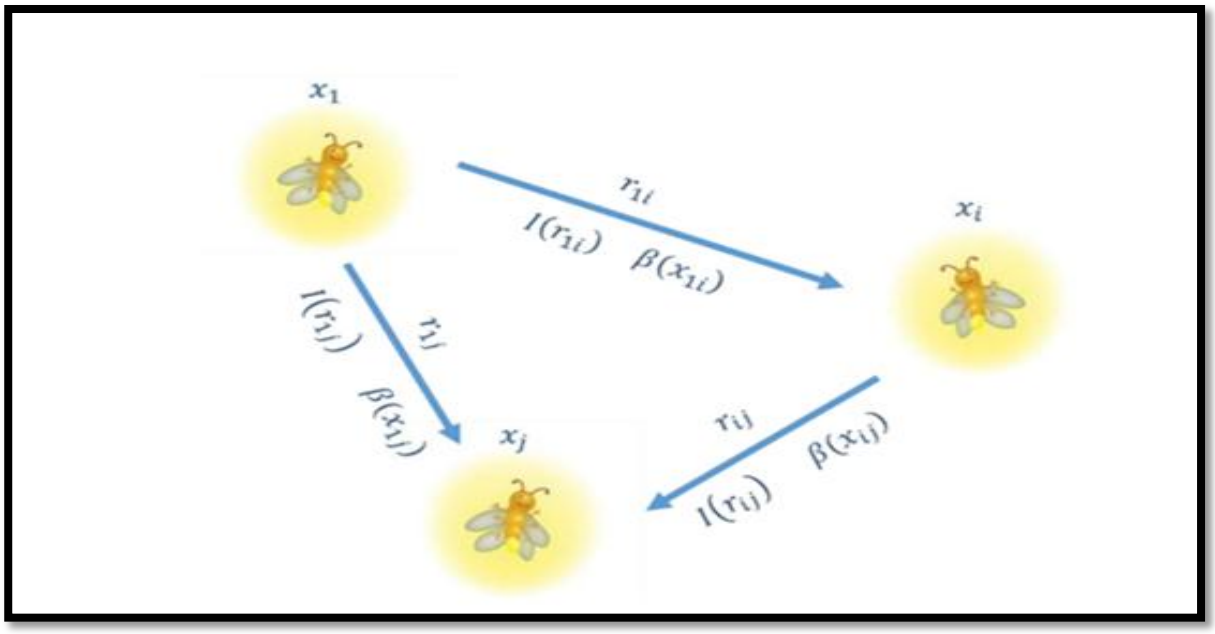


Figure II.6. Une vue conceptuelle des relations de l'algorithme FireFly, y compris les emplacements x, distance r, luminosité $I(r)$, et l'attractivité $\beta(r)$.

Conclusion

A travers ce chapitre, nous avons présenté les méthodes de résolution que nous allons utiliser durant la réalisation de ce travail, nous nous sommes focalisés sur une méthode de résolution exacte qui est la méthode séparation et évaluation en donnant son principe, ensuite nous avons expliqué les méthodes de résolution approchée précisément les méta-heuristiques en donnant une définition claire et précise de algorithme le FireFly.

Chapitre III :

Problème de Bin packing

III.1. Introduction

Le problème bin packing (PBP) est un problème d'optimisation NP-complet très connu. Il existe trois variantes principales des problèmes BP : Problèmes bin packing à une, deux et trois dimensions. Ils ont plusieurs applications réelles telles que le chargement de conteneurs, la coupe de stock, la conception d'emballages et l'allocation de ressources, etc. Le problème de bin packing consiste d'une manière générale à trouver le rangement le plus économique possible pour un ensemble d'objets dans des boîtes dites « bins ».

III.2. Problème de Bin packing

Supposons que nous avons n objets, chacun d'une taille donnée, et quelques boîtes de capacité égale. Nous voulons assigner les objets aux boîtes, en utilisant le moins de boîtes possible. En concédèrent que La taille totale des objets assignés à une boîte ne doit pas dépasser la capacité des boîtes, car leur capacité est égale à 1. Le problème peut être formulé comme suit : [22]

PROBLÈME D'EMBALLAGE DE BIN. (bin-packing problem)

Instance : Une liste de nombres non négatifs $a_1 \ldots . a_n < 1$.

Tâche : Trouver un $k \in N$ et une tâche $f : \{1, \ldots, n\} \rightarrow \{1, \ldots, k\}$ avec

$\sum_{i=f(i)=j} a_i \leq 1$ Pour tous $j \in \{1, \ldots, k\}$ tel que k soit le minimum.

III.2.1. Le problème de bin packing uni-, bi- et tri-dimensionnel

Les problèmes de bin packing se distinguent, selon la dimension, en :

- **Bin packing uni-dimensionnel**

Le problème de bin packing à une dimension peut être décrit par les données de n objets et m boîtes ($m \leq n$) avec w_i = la taille de l'objet i; $C =$ capacité de chacun des boîtes. Le but étant d'assigner chaque objet dans une boîte sans excéder la capacité des boîtes, sans fractionner les objets et en minimisant le nombre de boîtes utilisés. Dans le ref. [23] donne quelques résultats important sur la complexité des boîtes packing.

Exemple 1 :

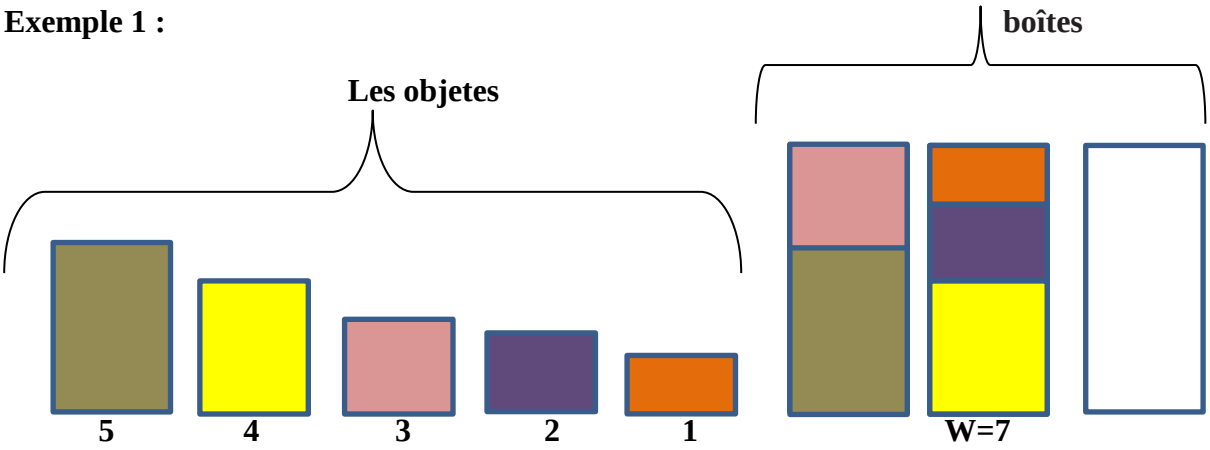


Figure III.7. Une instance de PBP-1D et une solution possible.

Données

N : ensemble de n objet avec w_i longueur du objet et W longueur des boîtes.

Objectif : minimiser le nombre de boîtes utilisés.

Contraintes: ne pas dépasser la capacité d'une boîte.

Cas particuliers du problème de partitionnement

N : ensemble des objets.

M : ensemble des remplissages possibles des boîtes.

- **Bin packing bi-dimensionnel :**

Dans le problème de bin packing bidimensionnel (2BP) on nous donne un ensemble de n éléments rectangulaires $j \in J = \{1, \dots, n\}$, chacun ayant largeur w_j et hauteur h_j, et un nombre illimité de boîtes rectangulaires identiques finis, ayant largeur W et hauteur H.Le problème est d'allouer, sans chevauchement, tous les objets au nombre minimum de boîtes, avec leurs bords parallèles à ceux des boîtes. On suppose que les éléments ont une orientation fixe, c'est-à-dire qu'ils ne peuvent pas être tournés.

Problème 2BP a de nombreuses applications industrielles, notamment dans la coupe (industries du bois et du verre) et l'emballage (transport et entreposage). [24]

Exemple 1 :

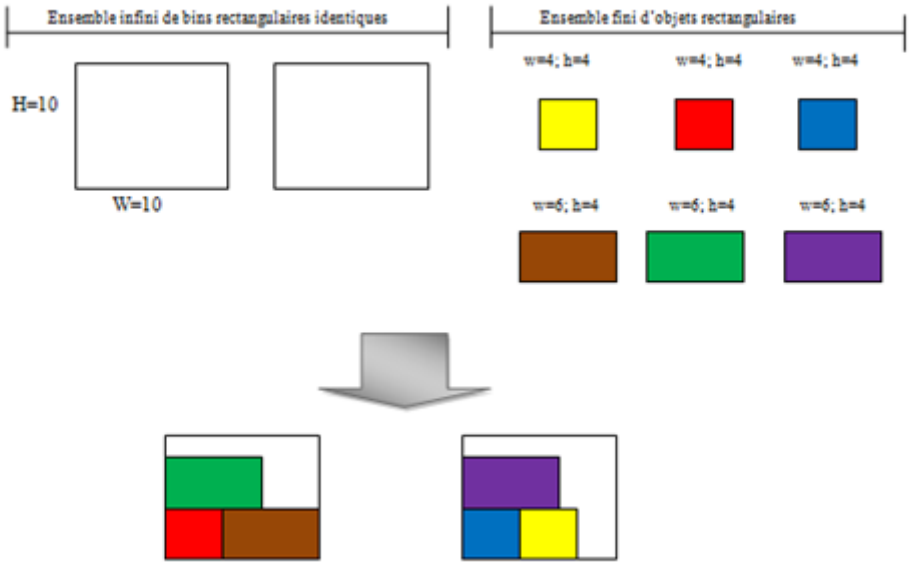


Figure III.8. Une instance de PBP-2D et une solution possible.

- **Bin packing tri-dimensionnel**

On nous donne un ensemble n des objets rectangulaires, chacune caractérisée par la largeur w_j, hauteur h_j et profondeur d_j $(j \in J = \{1, \dots, n\})$, et un nombre illimité de conteneurs tridimensionnels identiques (boîtes) ayant la largeur W, la hauteur H et la profondeur D. Le problème de bin packing en trois dimensions (3D-PBP) consiste à emballer orthogonalement toutes les objets dans le nombre minimum de boîtes. Nous supposons que les objets ne peuvent pas être tournées, c.-à-d. qu'elles sont emballées avec chaque bord parallèle au bord correspondant des boîtes. On suppose également, que toutes les données d'entrée sont des nombres entiers positifs satisfaisant $w_j \leq W, h_j \leq H$ et $d_j \leq D$ $(j \in J)$.

Problème 3D-PBP est fortement NP-complet, car il s'agit d'une généralisation du problème bien connu (unidimensionnel) Problème Bin Packing (1D-PBP), dans lequel un ensemble de n valeurs positives w_j doit être partitionné dans le nombre minimum de sous-ensembles de sorte que la valeur totale dans chaque sous-ensemble ne dépasse pas une capacité de boîtes donnée W. Il est clair que 1D-PBP est le cas particulier de 3D-PBP survenant lorsque $h_j = H$ et $d_j = D$ pour tous $j \in J$. Un autre problème connexe important se pose lorsque $d_j = D$ pour tous $j \in J$: nous

avons dans ce cas le problème d'emballage en binôme bidimensionnel (2D-PBP), demandant la détermination du nombre minimum de boîtes rectangulaires identiques de taille $W \times H$ nécessaires pour emballer un ensemble donné de rectangles de tailles $w_j \times h_j (j \in J)$.[25]

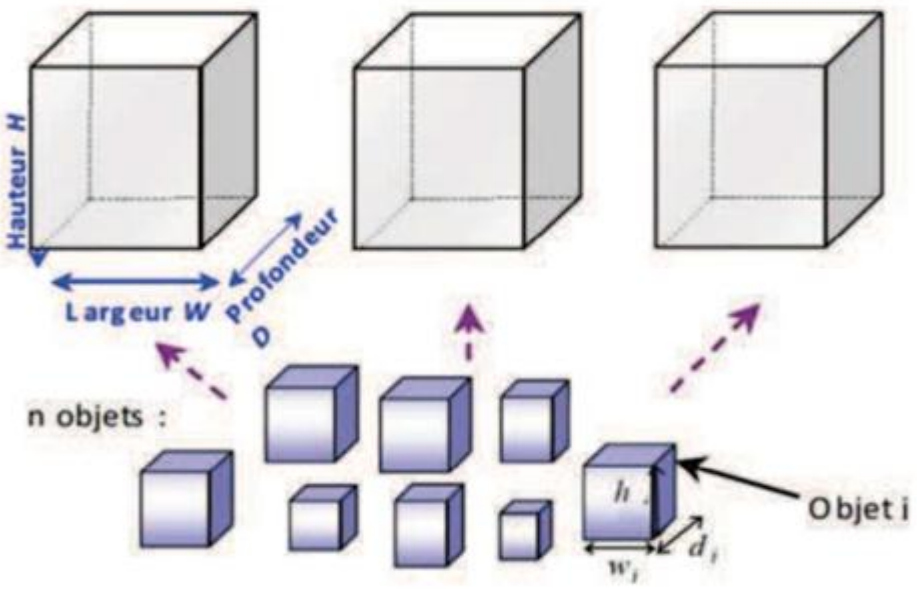


Figure III.9. Illustration graphique du PBP-3D.

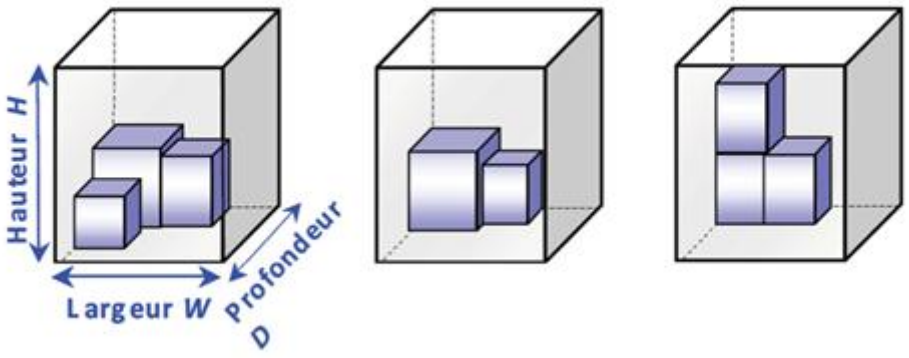


Figure III.10. Illustration d'une solution réalisable pour PBP-3D.

Il s'agit d'un problème lié au problème de chargement, des contraintes supplémentaires sont imposées sur chaque chargement:

1) éviter le débordement,
2) respecter le tonnage (volume intérieur) du support,
3) et définir des précédences et des préférences de regroupement entre certains objets.

Le problème de base revient donc à placer tous les objets de sorte que:

- les objets ne se chevauchent pas.
- le nombre de supports utilisés soit minimum.
- les contraintes soient respectées.

III.2.2. Modèle mathématique

✓ **Bin packing uni-dimensionnel**

Le problème d'emballage unidimensionnel peut être formulé comme suit [26] :

$$\min(z) = \sum_{j=1}^{n} y_j \qquad \text{(III.14)}$$

$$\sum_{i=1}^{n} w_i x_{ij} \leq C y_j \ , \forall j \qquad \text{(III.15)}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \ , \forall j \qquad \text{(III.16)}$$

$$x_{ij}, y_j = 0,1 \quad , \forall i, j \qquad \text{(III.17)}$$

Où

$$y_j = \begin{cases} 1, & \text{si le bin j est utilisé} \\ 0, & \text{sinon} \end{cases} \qquad \text{(III.18)}$$

$$x_{ij} = \begin{cases} 1, & \text{si l'objet i est assigné au bin j} \\ 0, & \text{si non} \end{cases} \qquad \text{(III.19)}$$

L'objectif consiste à réduire au minimum le nombre total de boîtes utilisés. Les contraintes (III.15) font en sorte que le poids des objets placés dans les boîtes j ne dépasse pas la capacité des boîtes. On n'oublie pas que toutes les boîtes ont la même capacité C, mais en général ce n'est pas nécessairement le cas. Enfin, les contraintes (III.16) font en sorte que chaque objet soit placé dans une seule boîte.

✓ **Bin packing bi-dimensionnel**

Il n'existe pas de modèle linéaire efficace lorsque le nombre des objets est très grand. Nous présentons donc deux modèles pour 2BP. Le premier modèle a été proposé par Gilmore et Gomory [27] comme une extension de leur approche 1BP [28]. Ce modèle est basé sur l'énumération de tous les sous-ensembles d'objets qui peuvent être rangés dans une même boîte. Chacun de ces ensembles est représenté par une vecteur colonne binaire S_j composé de n éléments $s_{ij} = 1, \dots, n$ tels que :

$$s_{ij} = \begin{cases} 1 & si\ a_i\ appartient\ au\ sous-ensemble\ j\ ; \\ 0 & sinon, \end{cases} \qquad \text{(III.20)}$$

Soit S la matrice composée des vecteurs S_k pour $k = 1, \dots, M, M$ étant le nombre de sous-ensembles qui peuvent être placés dans une même boîte. La matrice S représente donc l'ensemble de toutes les configurations de rangements réalisables. On cherche donc à minimiser la fonction objectif suivante :

$$\min \sum_{k}^{M} x_k \qquad \text{(III. 21)}$$

Sous les contraintes suivantes :

$$\min \sum_{i}^{M} s_{ij}\, x_j \qquad (i = 1, \dots, n) \qquad \text{(III.22)}$$

$$x_j \in \{0,1\} \qquad (j = 1, \dots, m) \qquad \text{(III. 23)}$$

La faiblesse de cette modélisation réside dans le très grand nombre de colonnes à générer. Gilmore et Gomory [27] ont présenté pour le 1BP, une méthode pour générer dynamiquement les colonnes au cas de besoin. Elle consiste en la résolution d'un problème de sac à dos unidimensionnel.

Le deuxième modèle a été proposé par Fekete et Schepers [29]. Il est basé sur la théorie de graphes pour le problème de décision.

- ✓ **Bin packing tri-dimensionnel**

Pour trouver la solution pour boîtes b, on suppose sans perte de généralité que $\sum_{i \in b} w_i \leq W$, $\sum_{i \in b} h_i \leq H$ et $\sum_{i \in b} d_i \leq D$. En tant que tel, il est juste de conclure que $\sum_{i \in b} w_i \,.\, h_i \,.\, d_i \leq W.H.D$. Donc pour chaque boîtes b nous l'intention de minimiser le volume gaspillé donné par $W.H.D - \sum_{i \in b} w_i \,.\, h_i \,.\, d_i$.

L'emballage en boîtes étant un problème NP-difficile, suggère qu'une recherche exhaustive de la solution optimale est en général insoluble du point de vue informatique, et aussi qu'il n'y a donc pas de méthode de solution optimale connue et réalisable du point de vue informatique pour le problème. Il faut donc trouver d'autres moyens d'obtenir une solution. [30]

III.2.3. Méthodes des résolutions du problème de bin packing

III.2.3.1. Méthodes exactes : Plusieurs méthodes exactes ont été proposées dans la littérature pour le problème de bin packing et ses variantes.

Parmi les méthodes exactes, on trouve la plupart des méthodes traditionnelles telles la méthode du simplex souvent utilisée 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. [31, 32, 33, 34]

III.2.3.2. 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. Les heuristiques proposées pour le PBP-1D, sont généralement adaptées et utilisées par la suite pour la résolution du PBP-2D. [35]

III.2.3.2.1. Résolutions pour le problème de bin packing uni-dimensionnel

Stratégies First-Fit (FF) : Dans Stratégies First-Fit (FF) les objets doivent être alloués dans les boîtes dans un ordre donné. Placez d'abord chaque objet dans la première boîte disponible, placez les objets suivant dans la boîte numérotée le plus bas dans lequel il tient. Si la boîte est complètement pleine, fermez-le définitivement. Si la boîte n'a pas assez d'espace pour contenir les objets, ouvrez une nouvelle boîte et allouez les objets [36, 37, 38]. L'algorithme de premier ajustement fournit une solution rapide mais souvent pas optimale, impliquant le placement de chaque élément dans la première boîte dans lequel il s'adaptera. Elle nécessite un temps, où n est le nombre d'articles à emballer [38, 39].

Exemple 1 : utilisez l'algorithme First-Fit pour emballer les poids 0.5, 0.7, 0.5, 0.2, 0.4, 0.2, 0.5, 0.1 et 0.6 dans des boîtes de capacité 1.

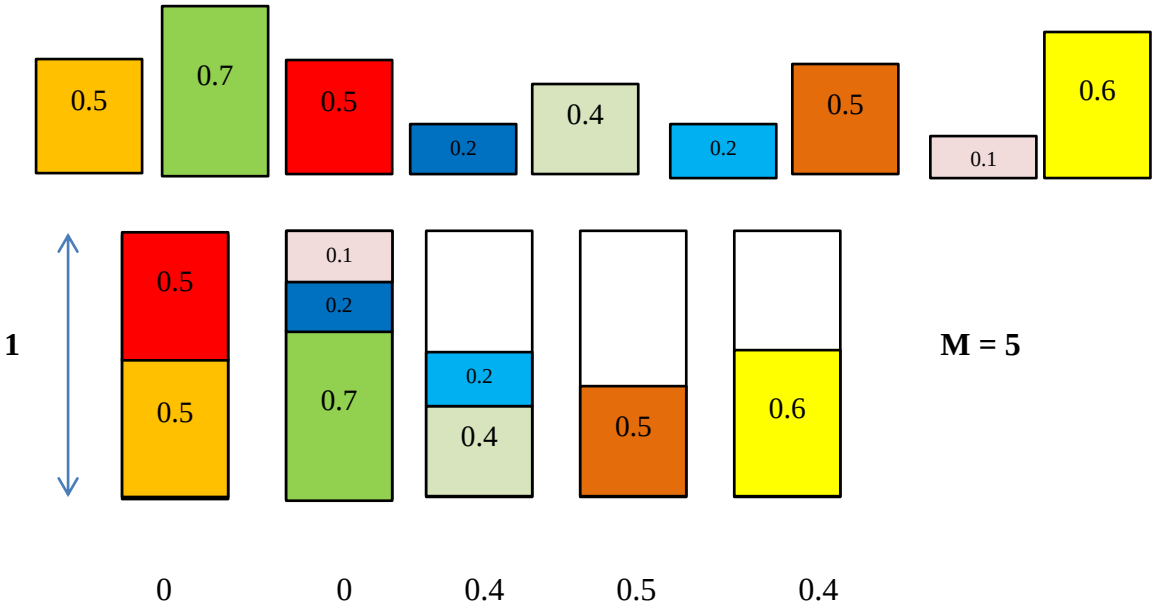


Figure III.11. Algorithme de First-Fit.

Stratégies Next-Fit (NF) : Dans Stratégies Next-Fit (NF) les éléments doivent être alloués dans les boîtes dans un ordre donné. NF vérifie s'il y a un espace dans une boîte actuelle, s'il y a un espace, alloue l'élément suivant dans la boîte actuelle. S'il ne rentre pas, fermez cette boîte et ouvrez une nouvelle boîte et attribuez les objets. L'algorithme Next-Fit est plus rapide que l'algorithme First-Fit, mais il utilise plus de boîtes. NF nécessite un temps. [40-39]

Exemple 1 : utilisez l'algorithme Next-Fit pour emballer les poids 0.5, 0.7, 0.5, 0.2, 0.4, 0.2, 0.5, 0.1 et 0.6 dans des boîtes de capacité 1.

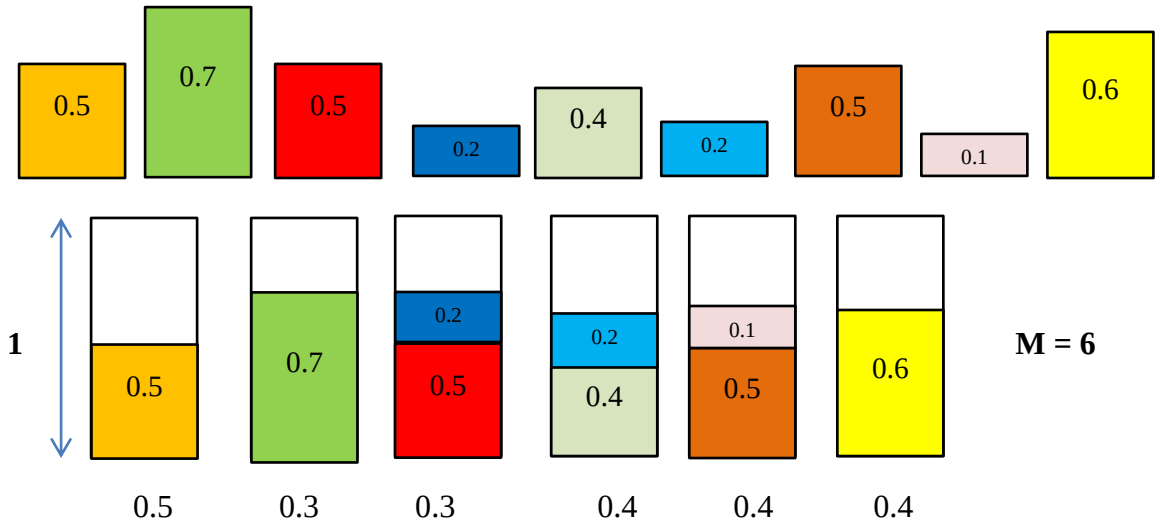


Figure III.12. Algorithme de Next -Fit.

Stratégies Best Fit (BF) : Dans Stratégies Best-Fit (BF) les objets doivent être alloués dans les boîtes dans un ordre donné. Placez les objets suivant dans cette boîte qui laissera le moins d'espace de déchets après avoir placé les objets dans les boîtes. S'il ne rentre dans aucun boîtes, ouvrez une nouvelle boîte et placez les objets. BF place les objets dans l'endroit le plus étroit parmi toutes les boîtes. Il fonctionne mieux pour les entrées aléatoires et nécessite un temps [38, 39].

Exemple 1 : utilisez l'algorithme Best-Fit pour emballer les poids 0.5, 0.7, 0.5, 0.2, 0.4, 0.2, 0.5, 0.1 et 0.6 dans des boîtes de capacité.

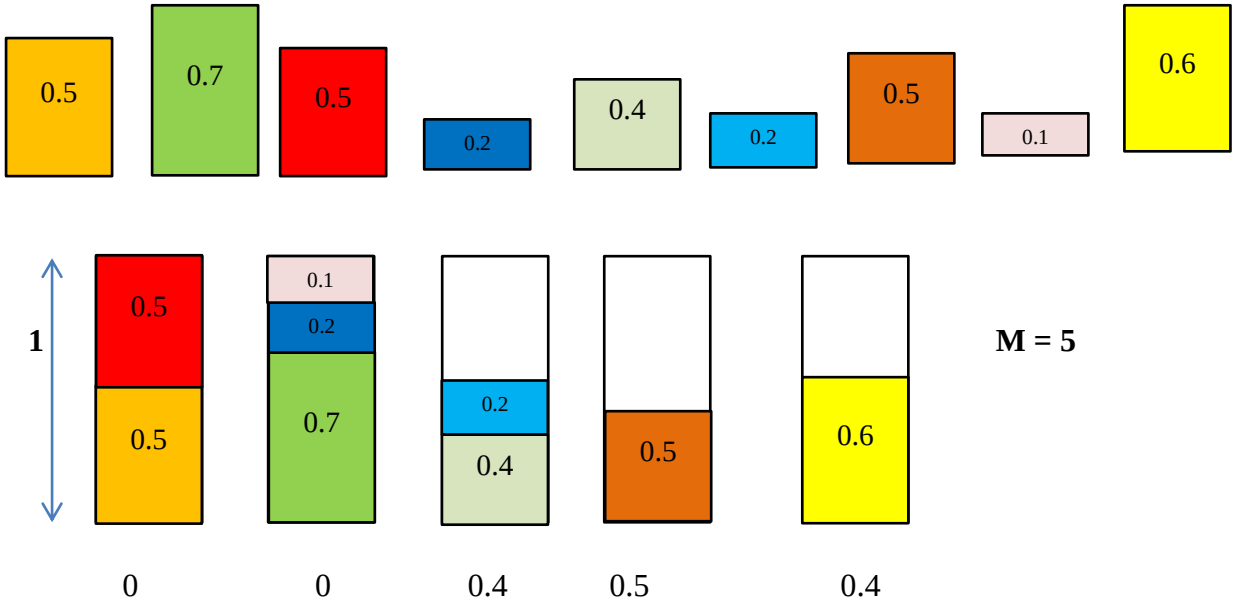


Figure III.13. Algorithme de Best-Fit.

Stratégies First-Fit Decreasing (FFD) et de Best-Fit Decreasing (BFD): Les heuristiques de First Fit décroissant et de Best Fit décroissant sont créées en réorganisant la liste des éléments, par poids, dans l'ordre décroissant avant d'être transmises à l'heuristique séquentielle respective.

Exemple 1 : utilisez l'algorithme First-Fit et Best-Fit décroissant pour emballer les poids 0.5, 0.7, 0.5, 0.2, 0.4, 0.2, 0.5, 0.1 et 0.6 dans des boîtes de capacité 1.

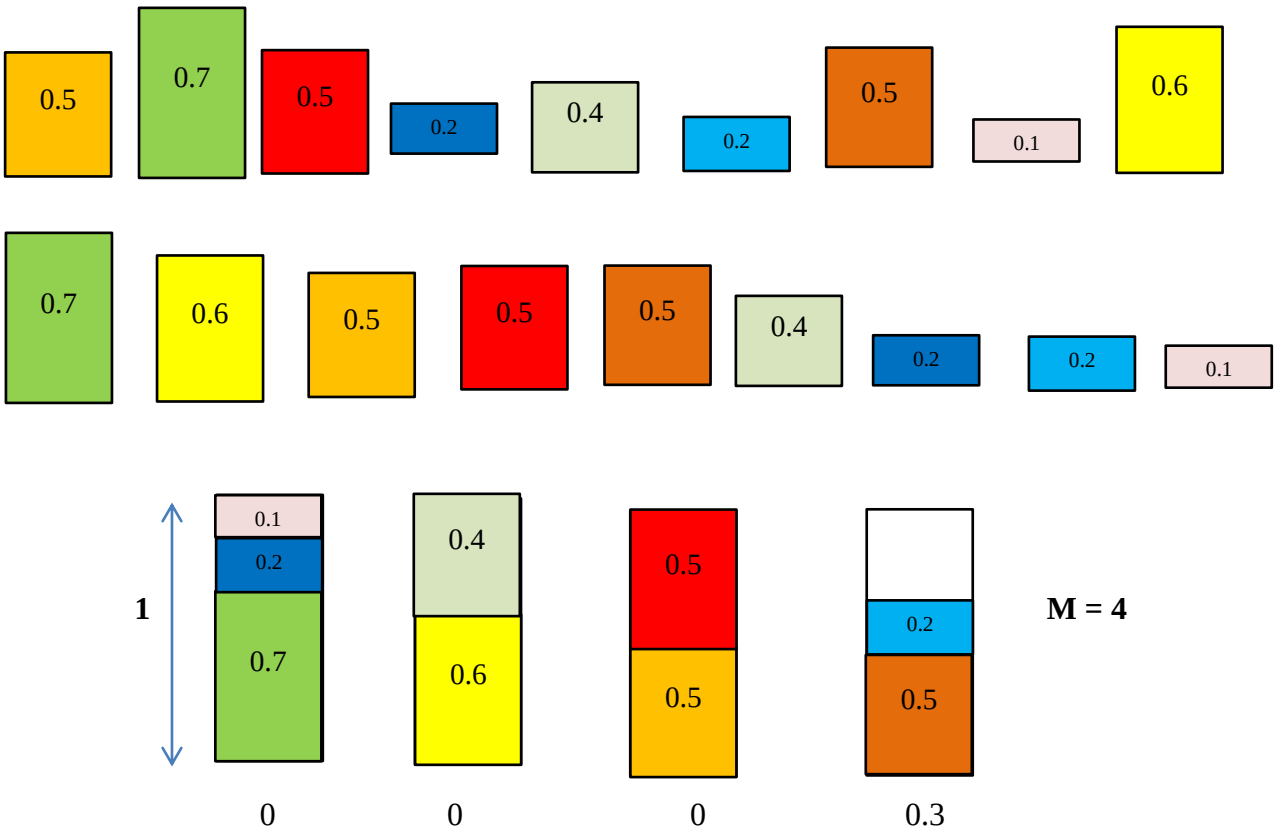


Figure III.14. Algorithme de First-Fit et Best-Fit décroissant.

III.2.3.2.2. Résolutions pour le problème de bin packing bi-dimensionnel

Le problème de bin packing en deux dimensions PBP-2D 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 PBP-1D. Ces heuristiques peuvent être divisées en deux principales familles :

- Des algorithmes à une phase emballent directement les objets dans les boîtes finis ;
- Les algorithmes à deux phases commencent par emballer les objets dans une seule bande, c'est-à-dire une boîte ayant une largeur W et une hauteur infinie.

Dans la deuxième phase, la solution de bande est utilisée pour construire un emballage dans des boîtes finis.

III.2.3.2.2.1. Les algorithmes en une phase

Deux algorithmes à une phase ont été présentés et évalués expérimentalement par Berkey et Wang [41].

L'algorithme Finite Next-Fit (FNF) : emballe directement les objets dans des boîtes finis exactement ([41,42]).

L'algorithme Finite First-Fit (FFF) : les objets en cours est emballé au niveau le plus boîtes de la première boîte où il tient ; si aucun niveau ne peut l'accueillir, un nouveau niveau est créé soit dans la première boîte approprié, soit en initialisant une nouvelle boîte (si aucun boîtes n'a suffisamment d'espace vertical disponible). Un exemple d'application de FFF est donné dans la Figure 15.

Remarque : Les deux algorithmes peuvent être implémentés de manière à nécessiter du temps.

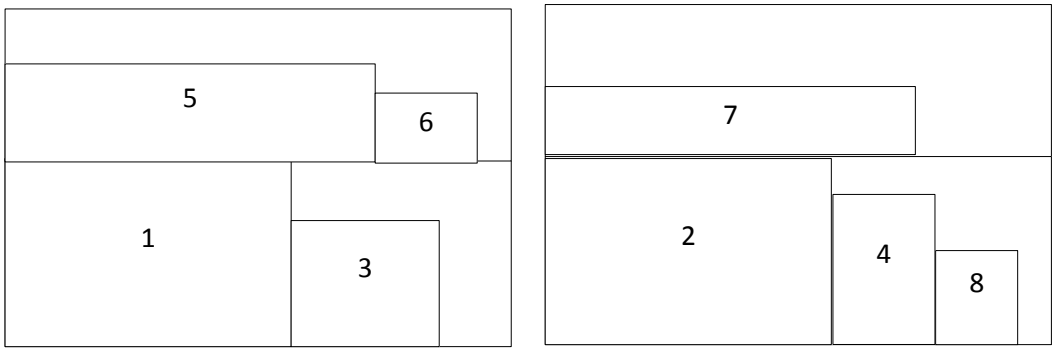


Figure III.15. Algorithme FFF.

Parmi les algorithmes qui ne rangent pas les objets par niveau on trouve **la stratégie Bottom-Left (BL) :** les objets sont considérés dans un ordre donné, et l'objet en cours est placé dans la position la plus en bas puis la plus à gauche possible.

III.2.3.2.2.2. Les algorithmes à deux phases

Dans l'algorithme à deux phases cas, on commence par les placer dans une bande (strip) de même largeur que les plaques, puis on utilise la solution obtenue pour construire une solution pour le problème initial en utilisant une heuristique

dédiée au problème de Bin Packing à une dimension. Beaucoup de ces approches sont dites par couches. Dans ces approches, le rangement des objets est obtenu en plaçant les objets de gauche à droite, dans des rangées formant des couches. Les niveaux sont définis en fonction de la taille de l'objet le plus grand dans chaque couche. Dans ce genre d'algorithmes, les objets sont initialement rangés dans l'ordre décroissant de leur taille. On rencontre ensuite plusieurs stratégies de placement comme Next-Fit Decreasing Height (NFDH), First-Fit Decreasing Height (FFDH) ou encore Best-Fit Decreasing Height (BFDH) [43]. Pour le problème du Bin-packing à deux dimensions, dans ce tableau 1 montre le travail de ces stratégies :

Abréviation	Nom	Placement de l'objet courant
NFDH	Next-Fit Decreasing Height	Le plus à gauche possible dans la couche courante si possible, sinon, la couche est fermée et une nouvelle est créée
FFDH	First-Fit Decreasing Height	Le plus à gauche possible dans la première couche où il rentre sinon, une nouvelle est créée
BFDH	Best-Fit Decreasing Height	Le plus à gauche possible dans la couche où l'espace restant horizontalement est le plus petit sinon, une nouvelle couche est créée

Tableau III.1. Différentes stratégies de placement.

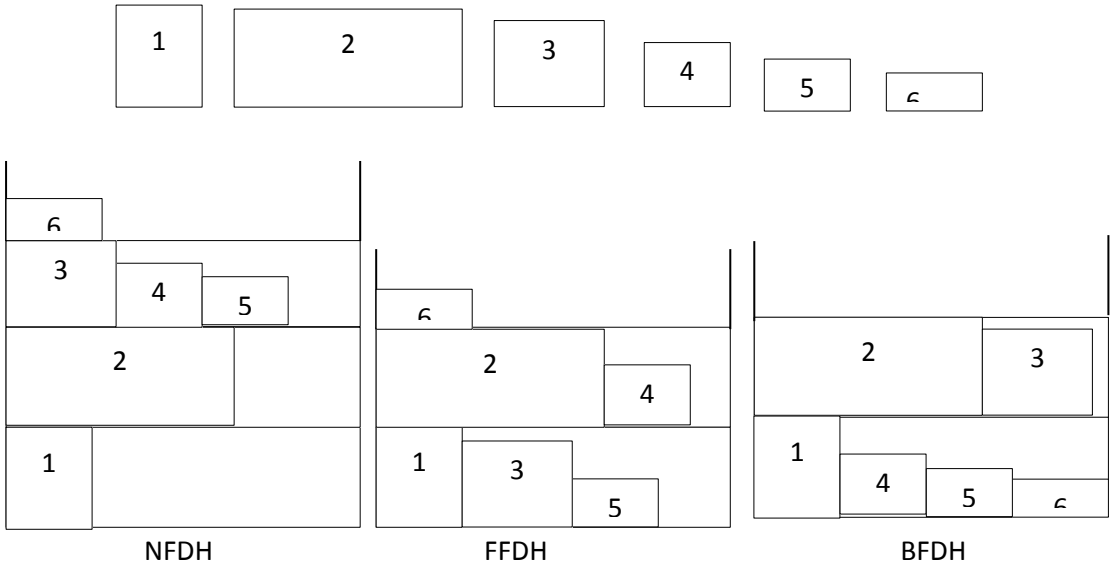


Figure III.16. Illustration d'exemples pour NFDH, FFDH, BFDH.

III.2.3.2.3. Résolutions pour le problème de bin packing tri-dimensionnel

First Fit [44,45] **:** Emballez les objets non affecté dans la première boîte qui a suffisamment d'espace. S'il n'y a pas de telles boîtes, affectez les objets dans une nouvelle boîte.

First Fit Decreasing : Presque la même chose que First Fit sauf que les objets est d'abord triée par ordre décroissant avant d'être emballés.

Last Fit : Emballez les objets non affecté dans la dernière boîte avec suffisamment d'espace. La recherche est similaire à First Fit mais dans l'ordre inverse des boîtes. S'il n'y a pas une telle boîte, affectez les objets dans une nouvelle boîte.

Best Fit : L'algorithme Best Fit place un article dans une boîte, qui est le plus plein parmi les boîtes dans lesquels les objets tient. Plus précisément:

- Les objets sont emballés un à la fois dans un ordre donné.
- Pour déterminer la boîte d'un objet, déterminez d'abord l'ensemble B de conteneurs dans lequel objet tient.
- Si B est vide, alors commencez une nouvelle boîte et placez objet dans ce nouveau boîtes.

Conclusion

Dans ce chapitre nous avons décrit les différents de PBP uni-, bi- et tri-dimensionnels, les modèles mathématiques .et concernant les approches de résolution, nous pouvons conclure que les méthodes exactes ne peuvent être utilisées que pour des exemples de très petite taille ou dans des cas très particuliers. Dans le domaine de bin packing, les algorithmes utilisés sont essentiellement des méthodes approchées.

Chapitre IV :

Proposition et implémentation

IV.1. Introduction

Au cours d'une décennie, de nombreux nouveaux algorithmes méta-heuristiques ont été introduits par les chercheurs problèmes d'optimisation discrets à contraintes multiples. Parmi ces algorithmes, les algorithmes inspirés de la nature sont devenu striés intéressant et distingué, car il imite le comportement de la nature ou des animaux qui résolvent leurs problèmes depuis longtemps et ont trouvé une solution presque optimale à leurs problèmes. Ces comportements de résolution de problèmes ont été incorporés dans les problèmes en temps réel en développant des modèles mathématiques de la nature. Certains de ces algorithmes basés sur la nature communément connus, par exemple, les réseaux neuronal artificiel, génétique algorithme, algorithme de colonie de fourmis, algorithme de luciole, algorithme abeille, etc. qui imitent le comportement de l'homme cerveau dans le traitement des données, génération panure, fourmi dans la recherche de nourriture, luciole dans l'accouplement, abeille dans l'identification du miel respectivement.

IV.2. Contexte et objectifs du travail

Les chercheurs ont proposé ces types d'algorithmes pour résoudre l'optimisation continue discrète problèmes en satisfaisant les contraintes qu'il implique. Tout au plus la performance de tous les algorithmes sont tout à fait satisfaisant seulement en considérant correctement les deux principaux critères au moment de l'application de ces algorithmes, C.-à-d. convertir les paramètres en temps réel aux paramètres de l'algorithme et ajuster l'algorithme pour le but, qui est appelé respectivement encodage et discrétisation. Mais parfois la formulation peut être représentée comme permutation des entiers, des tableaux complexes, etc., selon les variables et les contraintes impliquées dans le problème. Dans ce travail de recherche, l'algorithme firefly (AFF) a été utilisé pour identifier la meilleure solution optimale pour les problèmes d'emballage des bacs en 2 dimensions.

AFF est un algorithme méta-heuristique inspiré de la nature, qui imite le comportement des lucioles et peut résoudre les problèmes de calcul discrets complexes NP de meilleure façon. En 2008, Xin-She Yang a introduit ce modèle d'algorithme Firefly, qui imite le comportement social de la luciole

IV.3. Analyse et conception du modèle

IV.3.2. Organigramme de l'algorithme FireFly

L'algorithme FireFly peut être illustré davantage dans l'organigramme suivant :

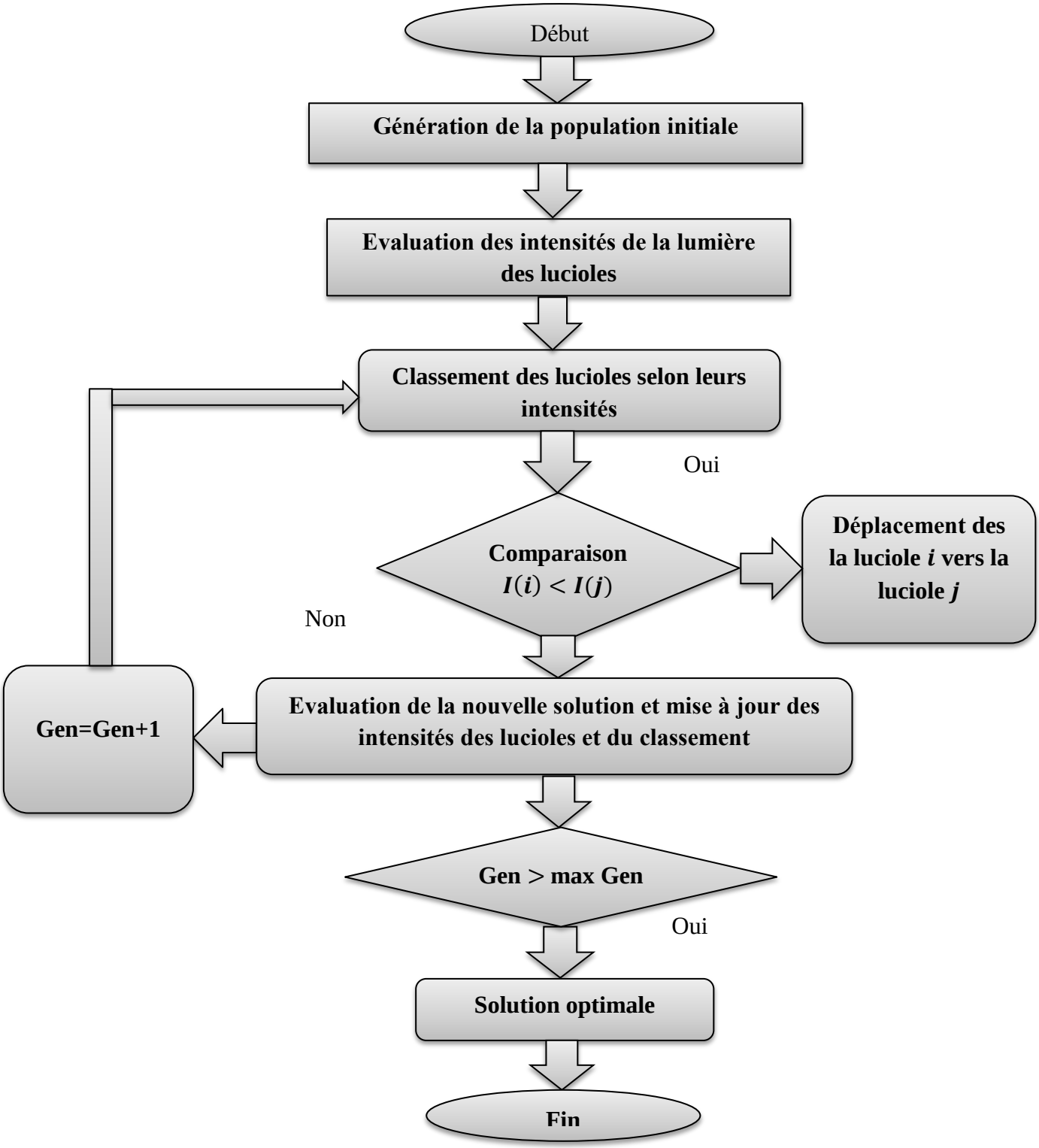


Figure IV.17. Organigramme algorithme FireFly.

IV.3.2.1. Paramètre pour l'algorithme FireFly

Les notions de paramètre utilisées dans l'algorithme sont

Paramètre	Notation de l'algorithme
Luminosité	Fonction objectif
Alpha (α)	Paramètre de randomisation
Beta (β)	Attraction
Gamma (γ)	Coefficient d'absorption
Nombre de générations	Itérations
Nombre de FireFly	Population
Dimension	Dimension du problème

IV.4. Les algorithmes utilisés pour la résolution de ce modèle

- **Algorithme First-Fit**

First-Fit (FF) garde tous les bacs vides ouverts. Il place le prochain article dans le bac numéroté le plus bas dans lequel l'article tient. S'il ne rentre dans aucun bac, un nouveau bac est ouvert. First-Fit ne pourrait pas atteindre un meilleur temps d'exécution car il garde toutes les poubelles non vides actives et tente d'emballer chaque article dans ces poubelles avant d'en ouvrir une nouvelle. Dans cet algorithme, la règle suivie est :

Placez d'abord un élément dans le premier bac, appelé bac indexé le plus bas dans lequel il s'insère, c'est-à-dire que s'il y a un bac partiellement rempli, placez l'élément dans le bac indexé le plus bas sinon, commencez un nouveau bac [46].

PSEUDO CODE OF FIRST-FIT ALGORITHM

Procedure First-Fit ()
Begin
1: for *all objects* $i = 1, 2, \ldots, n$ *do*
2: for *all bins* $j = 1, 2, \ldots$ *do*
3: if *Object* i *fits in bin* j *then*
4*: Pack objects* i *in bin*j*.*
5*: Break the loop and pack the next object*
6: end if
7: end for
8: if *Object* i *did not fit in any available bin then*
9: *Create new bin and pack object* i
10: end if
11: end for
End procedure

- **Algorithme Best-Fit**

Best-Fit (BF) est l'algorithme le plus connu pour le problème de Bin Packing. Il est simple et se comporte bien dans la pratique. Best Fit choisit (parmi les bacs possibles pour l'article) celui où la quantité d'espace libre est minimale. Il choisit le bac avec le moins d'espace libre dans lequel il peut encore tenir l'élément courant. Cet algorithme essaie de choisir le bac le plus complet possible avec suffisamment d'espace chaque fois qu'un élément est assigné. Toutes les poubelles inoccupées sont gardées ouvertes jusqu'à la fin. Il place le prochain article dans le bac dont le contenu actuel est le plus grand, mais ne doit pas dépasser sa capacité. S'il ne rentre dans aucun bac, un nouveau bac est ouvert [47].

PSEUDO CODE OF BEST-FIT ALGORITHM

Procedure Best-Fit ()
Begin
1: for *all objects* $i = 1, 2, \ldots, n$ *do*
2: for *all bins* $j = 1, 2, \ldots$ *do*
3: if *Object* i *fits in bin* j *then*
4: *Calculate remaining capacity after the object has been added*
5: end if
6: end for
7: *Pack object* i *in bin*j*, where* j *is the bin with minimum remaining capacity after adding the object (i.e. the object "fits best")*
8: If *no such bin exists, open a new one and add the object*
9: end for
End procedure

IV.5. Outils de développement

Le nom Matlab signifie MATRIX LABORATORY. Matlab a été écrit à l'origine pour fournir un accès facile aux logiciels matriciels développés par les projets LINPACK (Linear System Package) et EISPACK (Eigen System Package).

Matlab est un langage haute performance pour l'informatique technique. Il intègre le calcul, la visualisation et l'environnement de programmation. De plus, Matlab est un langage de programmation moderne. Il a une structure de données sophistiquée, contient des outils intégrés d'édition et de débogage, et prend en charge

la programmation orientée objet. Ces facteurs font de MATLAB un excellent outil d'enseignement et de recherche.

Matlab pour résoudre des problèmes techniques (par exemple, C, FORTRAN). Matlab est un système interactif dont l'élément de données de base est un tableau qui ne nécessite pas de dimensionnement. Le progiciel est disponible sur le marché depuis 1984 et est maintenant considéré comme un outil standard dans la plupart des universités et des industries du monde entier.

Figure IV.18. Matlab logo.

IV.6. Résultats d'implémentation

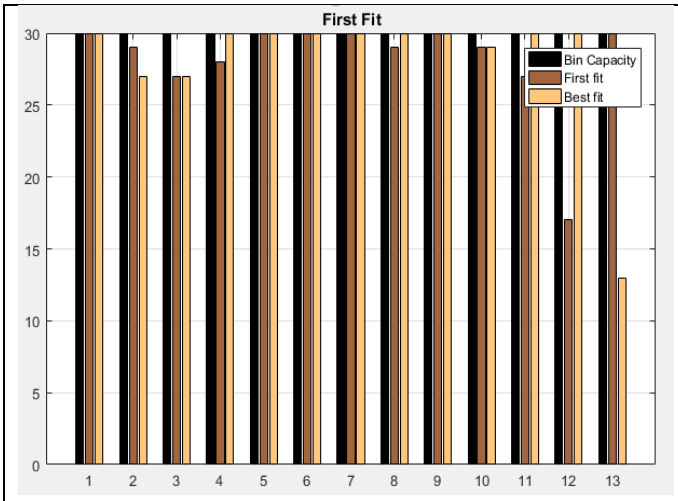
Les résultats obtenus avec l'algorithme FireFly ont été comparés avec l'algorithme First-Fit et l'algorithme Best-Fit. Les résultats expérimentaux et l'étude comparative des algorithmes sont abordés dans cette section.

A. Problèmes de test de bin packing

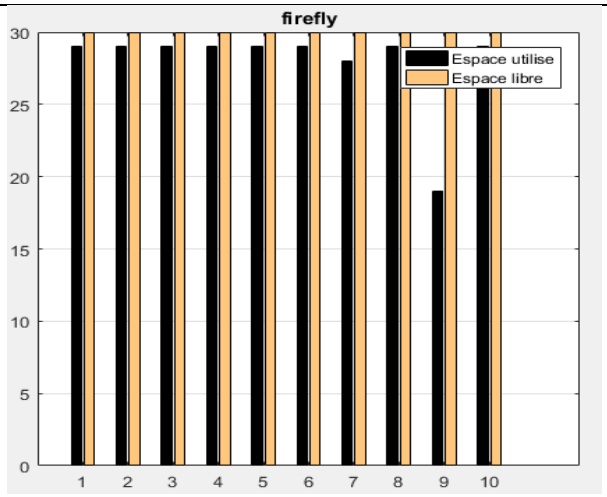
Dans ce travail de recherche, trois classes d'ensemble de données de référence 1, l'ensemble de données 2 et l'ensemble de données 3 pour BPP-1 sont classées en instances de classes Facile, Moyen et Difficile. La référence dans l'instance de classe facile, n (nombre d'articles) varie de 10 à 25 et c (capacité du bac) de 15 à 30. Dans l'instance de classe moyenne, n est compris entre 25 et 30 et c est égal à 50 et dans les instances de classe dure, n est égal à 50 et c est égal à 70.

B. Résultats de l'implémentation des instances de classe Facile (Easy)

Dans ce travail de recherche, 10 instances de classe facile ont été résolues en utilisant les algorithmes First-Fit, Best-Fit et FireFly. Les résultats de trois algorithmes ont été comparés et analysés. L'algorithme FireFly a donné a montré de bonnes performances par rapport à l'algorithme le mieux adapté et pour rester approximativement la même. Il a été constaté à partir des résultats que les algorithmes FireFly et Best-Fit sont efficaces en termes de bac utilisé et d'espace libre par rapport à l'algorithme First-Fit. Les résultats des calculs sont donnés dans le tableau 2 et des figures montrent la représentation graphique du tableau 2.



Figures IV.19. First et Best Fit.



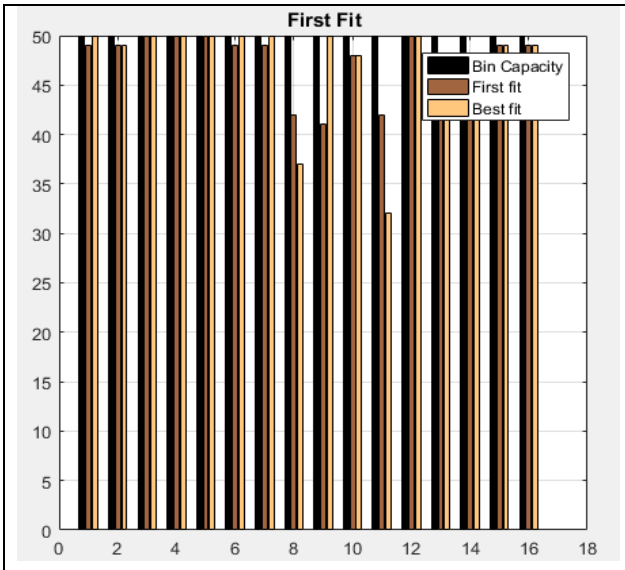
Figures IV.20. FireFly.

Nom instances	N	C	Opt	First-Fit		Best-Fit		FireFly	
				Bin utilize	Escape liber	Bin utilize	Escape liber	Bin utilize	Escape liber
inst01	10	15	7	8	4	8	4	7	7
inst02	12	15	8	8	5	8	4	8	8
inst03	20	20	10	11	6	11	5	10	10
inst04	20	22	10	11	6	11	4	10	10
inst05	21	25	11	13	10	13	9	11	11
inst06	16	25	9	12	9	12	7	9	9
inst07	12	27	7	9	6	9	4	7	7
inst08	25	30	10	13	7	13	4	10	10
inst09	25	30	16	17	10	17	10	16	16
inst010	19	30	12	13	8	13	6	12	12

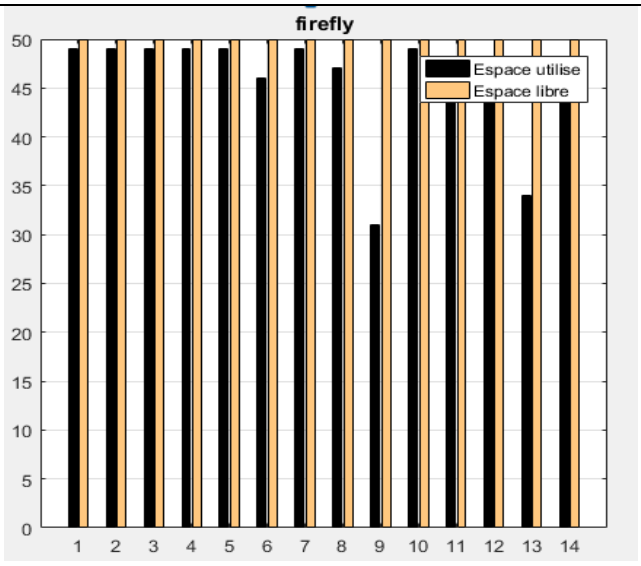
Tableau IV.2. Résultat comparatif du PBP de classe Facile.

C. Résultats de de l'implémentation des instances de classe moyenne (Medium)

En classe moyenne, 5 instances ont été résolues. L'algorithme FireFly a donné des a montrés de bonnes performances par rapport aux deux autres techniques. Les algorithmes FireFly et Best-Fit sont efficaces en termes de bac utilisé et d'espace libre par rapport à l'algorithme First-Fit. Les résultats des calculs sont donnés dans le tableau 3. Des figure et montre la représentation graphique du tableau 3.



Figures IV.21. First et Best Fit.



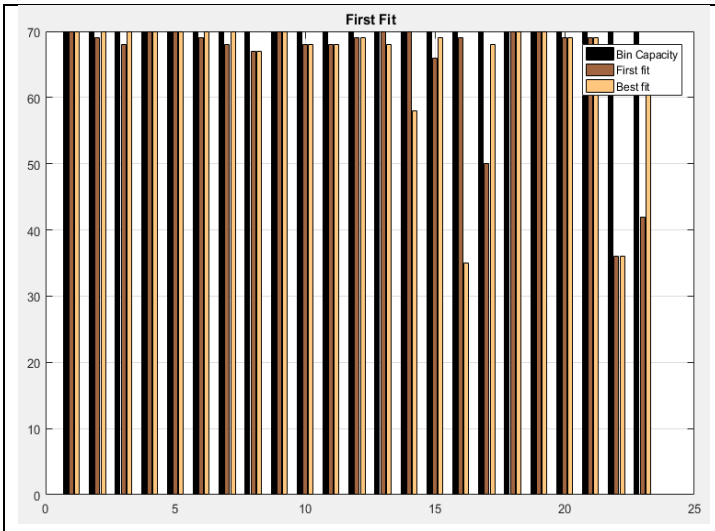
Figures IV.22. FireFly.

				First-Fit		Best-Fit		FireFly	
Nom instances	N	C	Opt	Bin utilize	Escape liber	Bin utilize	Escape liber	Bin utilize	Escape liber
inst01	25	50	12	12	8	12	8	12	12
inst02	27	50	14	15	12	15	8	14	14
inst03	29	50	14	15	12	15	8	14	14
inst04	30	50	14	16	12	16	8	14	14
inst05	30	50	17	17	9	17	7	17	17

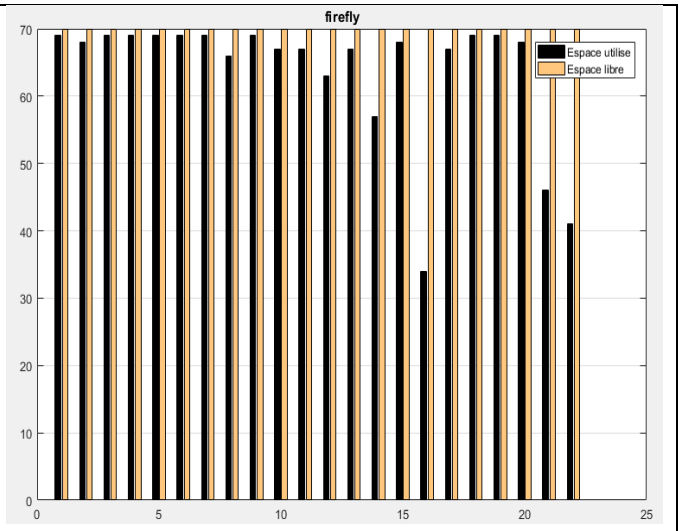
Tableau IV.3. Résultat comparatif du PBP de classe Moyenne.

D. Résultats de la mise en œuvre des instances de classe difficile (Hard)

Les instances de classe difficile du problème Bin Packing sont plus difficiles et plus difficiles à résoudre. . Dans la classe difficile, 3 cas ont été résolus. Il a été constaté à partir des résultats que les algorithmes de Best-Fit et FireFly ont obtenus des solutions presque optimales. L'algorithme FireFly et l'algorithme de Best-Fit montrent de bonnes performances pour toutes les instances par rapport aux techniques de First-Fit. Les résultats de calcul sont présentés au tableau 4. Des figures représentation du tableau 4.



Figures IV.23. First et Best Fit.



Figures IV.24. FireFly.

				First-Fit		Best-Fit		FireFly	
Nom instances	N	C	Opt	Bin utilize	Escape liber	Bin utilize	Escape liber	Bin utilize	Escape liber
inst01	50	70	21	21	13	21	12	21	21
inst02	50	70	22	23	15	23	13	22	22
inst03	50	70	20	21	13	21	13	20	20

Tableau IV.4. Résultat comparatif du PBP de classe difficile.

Conclusions

Dans ce travail de recherche, l'algorithme FireFly (AFF), l'algorithme Best fit et l'algorithme First-Fit a été implémenté pour le PBP uni-dimensionnel avec des bacs de taille fixe. Ces algorithmes ont été testés sur des instances de problèmes de référence standard. Les résultats obtenus par l'algorithme FireFly montrent de bonnes performances pour la plupart des cas problématiques par rapport à l'algorithme First-Fit et à peu près les mêmes par rapport aux résultats de l'algorithme Best fit. Ce travail peut être étendu en modifiant les paramètres de l'algorithme FireFly pour améliorer les résultats. L'algorithme FireFly peut être modifié efficacement pour d'autres types de problèmes d'optimisation combinatoire. L'algorithme FireFly peut également être appliqué pour résoudre le problème de bin packing en trois dimensions.

Dans ce travail, nous avons effectué une étude expérimentale de méthodes de résolution heuristiques. En particulier, nous avons présenté trois algorithmes fondés sur le First-Fit, Best-Fit et l'algorithme FireFly pour la résolution du problème de Bin Packing.

Des expérimentations comparatives des trois algorithmes ont été réalisées sur trois classes d'ensemble de données. Les résultats ont montré les points suivants :

- Un bon choix des paramètres d'une méthode est très important, le réglage des paramètres est crucial et conditionne la qualité des résultats obtenus.

Les perspectives issues de ce travail sont :

First-Fit (FF) ne pourrait pas atteindre un meilleur temps d'exécution car il garde toutes les poubelles non vides actives et tente d'emballer chaque article dans ces poubelles avant d'en ouvrir une nouvelle.

Best-Fit (BF) Cet algorithme essaie de choisir le bac le plus complet possible avec suffisamment d'espace chaque fois qu'un élément est assigné. Toutes les poubelles inoccupées sont gardées ouvertes jusqu'à la fin.

Les expérimentations que nous avons effectuées ont bien montré l'efficacité de l'algorithme FireFly. Nous avons constaté que l'algorithme FireFly est influencé par les facteurs tels que la taille de la population, le nombre d'itération et le nombre de canaux.

A titre d'exemple, si on augmente le nombre de canaux on va augmenter forcément le temps d'exécution. Ce qui va aider de trouver une solution optimale.

Il y a d'autre algorithme pour résolu le problème de bin packing par exemple : PSO, Algorithme génétique … .

Bibliographiques

[1] Yann Collette , Patrick Siarry , *Optimisation Multiobjectif* , ÉDITIONSEYROLLE S61, Bld Saint-Germain 75240 Paris Cedex 05 ,2002.

[2] Xin-She Yang , *Engineering Optimization An Introduction with Metaheuristic Applications* , University of Cambridge, United Kingdom ,2010.

[3] Laurent Deroussi , *Métaheuristiques pour la logistique* , ISTE Editions Ltd 27-37 St George's Road London SW19 4EU UK ,216.

[4] Osyczka, A. In Multicriteria optimization for engineering design (1985), Design Optimization, pp. 193–227**.**

[5] Steuer, R.E., 1986.Multiple Criteria Optimization – Theory, Computation and Application .Wiley, New York**.**

[**6] G. Nemhauser and L**. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.

[7] S. Kirkpatrick, C. Gellat, and M., 1983. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680.

[8] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[9] H. R. Louren¸co, O. Martin, and T. St¨utzle. Iterated local search. In Glover and Kochenberger [17], pages 321–353.

[10] P. Hansen and N. Mladenovi´c. An introduction to variable neighborhood search. In S. Voß, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-heuristics: advances and trends in local search paradigms for optimization*, pages 433–438. Kluwer Academic Publishers, 1999.

[11] T. B¨ack, D. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation*. Oxford University Press, New York NY, 1997.

[12] F. Glover, M. Laguna, and R. Mart´ı. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39(3):653–684, 2000.

[14] P. Larra˜naga and J. Lozano. Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation. Kluwer Academic Publishers, 2001.

[14] J. Puchinger et G. R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification, in proceedings of the first international work-coriference on the interplay between natural and artificial computation. Springer (Ed), Las Palmas, Spain, LNCS :41–53, 2005.

[15] A. H. Land et A. G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28 :497–520, 1960.

[16] Yang X.S. :"Nature-Inspired Metaheuristic Algorithms". Luniver Press, UK. (2008).

[17] Yang,X.S, :Firefly algorithms for multimodal optimization. In: Stochastic Algorithms: Foun-dations and Applications, SAGA 2009. Lectur Notes in Computer Sciences, vol. 5792, pp. 169-178 (2009).

[18] Fister, I., Fister Jr., I., Yang, X.S., Brest, J.: Acomprehensive review of firefly algorithms. Swarm Evol. Comput. 6 (in press) (2013).http://dx.doi.org/10.1016/j.swevo.2013.06.001.

[19] Yang, X.-S. *Nature-Inspired Metaheuristic Algorithms*; Luniver Press: Frome, UK, 2010.

[20] Yang, X.-S. *Engineering Optimization: An Introduction with Metaheuristic Applications*; Wiley: Hoboken, NJ, USA, 2010.

[21] Jervase, J.A.; Bourdoucen, H.; Al-Lawati, A. Solar cell parameter extraction using genetic algorithms. *Meas. Sci. Technol.* **2001**, *12*, 1922.

[22] Bernhard Korte ,Jens Vygen , *Optimisation combinatoire Théorie et algorithmes* , Avant-propos a la quatrième édition originale , 2010.

[23] B. Korte, J. Vygen ; Optimisation combinatoire théorie et algorithmes ; Springer-Verlag France (2010).

[24] Andrea Lodi, Silvano Martello, Daniele Vigo .*Recent advances on two-dimensional bin packing problems* , Dipartimento di Elettronica, Informatica e Sistemistica, University of Bologna, Viale Risorgimento 2 ,2001.

[25] Silvano Martello ,David Pisinger ,Daniele Vigo.*The Three-Dimensional Bin Packing Problem* ,Università degli Studi di Bologna Dipartimento di Elettronica, Informatica e Sistemistica ,1997.

[26] Mitsuo Gen, Runwei Cheng. *Genetic algorithms and engineering optimization* . New York [u.a.] : John Wiley & Sons, 2000.

[27] P.Gilmore et R. Gomory. A linear programming approach to the cutting stock problem-part ii. *Ops.Res*, 11: 863-888, 1963.

[28] P.Gilmore et R. Gomory. A linear programming approach to the cutting stock problem-part ii. *Ops.Res*., 13: 94-120,,1965.

[29] J.Schepers , van der veen, et S. Fekete . A combinatorial characterization of higher-dimensional orthogonal packing. Mathematics of Operations Research, 29 :353-368, 2004.

[30] Erick Dube , Leon R. Kanavathy , *OPTIMIZING THREE-DIMENSIONAL BIN PACKING THROUGH SIMULATION* **,** All content following this page was uploaded by Leon Reeves Kanavathy on 31 January 2017.

[31] J. E. Beasley, An exact tow-dimensional non-guillotine cutting tree search procedure.Operations Research, 33,49-64,1985**.**

[32]V. D. Cung, M. Hifi, B. Le Cun. Constrained two-dimensional cutting problem : a best-first branch-and-bound exact algorithm. International Transactions in Operational Research,7(3),185-210,2000.

[33]E. Hadjiconstantinou, N. Christophides, An exact algorithm for general, orthogonal, two-dimensional Knapsack problem. European Journal of Operational Research,83, 39-56,1995.

[34] M. Hadjiconstantinou, N. Christophides, An exact algorithm for constrained two-dimensional cutting stock. Computers and Operations Research, 24, 727-736,1997.

[35] D. S. Johnson, Fast algorithms for bin packing. Journal of Computers and System Sciences, 8, 272-314, 1974.

[36] http://mathworld.wolfram.com/BinPackingProblem.html

[37]http://www.astarmathsandphysics.com/a_level_maths_notes/D1/a_level_maths_notes_d1_bin_packing_algorithms_first_fit_algorithm.html

[38] www.ams.jhu.edu/~castello/362/Handouts/BinPacking.doc.

[39] www.or.deis.unibo.it/kp/Chapter8.pdf.

[40]http://www2.informatik.hu-berlin.de/alkox/lehre/lvws1011/coalg/bin_packing.pdf

[41] J.O. Berkey, P.Y. Wang, Two dimensional finite bin packing algorithms, J. Oper. Res. Soc. 38 (1987) 423–429.

[42] J.B. Frenk, G.G. Galambos, Hybrid next-fit algorithm for the two -dimensional rectangle bin-packing problem, Computing 39 (1987) 201–217.

[43] A. Lodi, S. Martello and D. Vigo, Two-dimensional packing problems : A survey. Eur. J. Oper. Res. 141 (2002) 241–252.

[44]Emanuel Falkenauer, A hybrid grouping genetic algorithm for bin packing, *Journal of Heuristics,* 1996.

[45] Leon Kos, Joze Duhovnik, Rod Cutting Optimization with Store Utilization, *International design conference –DESIGN, Dubrovnik,* 2000.

[46] Ullas Mohan, "*Bio Inspired Computing*", Division Of computer science, SOE, CUSAT, July 2008.

[47] Swathi Venigella, "*Cloud Storage and Online Bin Packing*", May 2008.

Résumé

Algorithme FireFly inspiré du comportement clignotant des lucioles Le comportement clignotant des lucioles est d'attirer d'autres lucioles du groupe pour l'accouplement. La luciole la moins brillante sera attirée par la plus brillante. Comme toutes les lucioles sont supposées être unisexuées, chaque luciole est attirée par l'autre. Dans cette thèse, l'algorithme FireFly (AFF) est utilisé pour résoudre le problème bin packing en une dimension. Et comparez-le davantage avec les algorithmes First-Fit et Best-Fit. 1D-PBP consiste à emballer orthogonalement un ensemble d'articles de poids différents dans un nombre minimum de bacs, minimisant ainsi le gaspillage d'espace et le temps d'exécution.

Les résultats ont montré que FireFly et best-fit sont approximativement les mêmes et bien plutôt que le premier ajustement pour la plupart des cas problématiques.

Mots clés : Algorithme FireFly, problème Bin packing, algorithme de first fit, algorithme de Best fit.

Abstract

Firefly algorithm inspired by the flashing behaviour of fireflies the flashing behaviour of the fireflies is to attract other fireflies in the group for mating. The less bright firefly will be attracted by the brighter one. As all the fireflies are assumed to be unisexual, each firefly is attracted to the other. In this thesis, Firefly Algorithm (FFA) is used to solve One Dimensional Bin packing problem. And further compare it with First-Fit and Best-Fit algorithms. 1D-BPP is to pack a set of items with different weights orthogonally into a minimum number of bins, minimizing bin space wastage and execution time.

The results have shown that the firefly and best fit is approximately same and well rather than first fit for most of the problem cases**.**

Keywords: Firefly Algorithm, Bin packing problem, first fit algorithm, Best fit algorithm.

الملخص

خوارزمية FireFly مستوحاة من السلوك الوامض لليراعات، يتمثل السلوك الوامض لليراعات في جذب اليراعات الاخرى في المجموعة للتزاوج. تنجذب اليراعة الاقل سطوعا الى اليراعة الاكثر اشراقا. نظرا لأنه من المفترض ان تكون جميع اليراعات ثنائية الجنس، فان كل يراعة تنجذب الى الاخرى. في هذه الاطروحة، يتم استخدام خوارزمية FireFly (AFF) لحل مشكلة تعبئة الحاوية احادية البعد. ومقارنتها ايضا بخوارزميات First-Fit و Best-fit.1D-PBP هو حزم مجموعة من العناصر بأوزان مختلفة بشكل متعامد في اقل عدد من الحاويات، مما يقلل من اهدار مساحة الحاوية ووقت التنفيذ.

اظهرت النتائج ان FireFly و Best-Fit هي نفسها تقريبا وهي جيدة بدلا من كونها مناسبة اولا لمعظم حالات المشكلة .

الكلمات المفتاحية : خوارزمية FireFly ،مشكلة تعبئة الحاوية ،خوارزمية First-Fit، خوارزمية Best-Fit.