REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTER DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU

Faculté sciences
Département mathématiques

Mémoire de fin d'etudes

En vu d'obtention du diplôme de master en :
Mathématiques appliquées à la gestion

Thème

Problème de placement unidimensionnel

Réalisé par *: Melle KEBAILI Hadjila* ***Dirigé par*** *: Mr.KASDI*

Année universitaire 2016/2017

Remerciements

Au terme de ce travail, je tien à exprimer mes vifs remerciements à mon promoteur Mr. KASDI pour l'aide très précieuse et sa contribution à l'accomplissement de ce travail dans les meilleures conditions, avec notre gratitude et respect.

J'exprime ma sincère gratitude à tous les enseignants qui ont contribué à notre formation.

Je tiens à remercier le président et les membres de jury pour leur honorable présence

Mes remerciements vont également à tous ceux qui ont participé de prés ou de loin à la réalisation de ce travail.

Dédicaces

Je rends grâce à dieu de m'avoir donné la santé, le courage et la volonté ainsi que la conscience d'avoir pu terminer mes études.

Je dédie Ce Travail à:

A ma chère et tendre mère
A celle qui a tout souffert, sans me faire souffrir, qu'elle trouve dans ce mémoire le témoignage de ma reconnaissance et de mon affection pour tous les sacrifices, l'extrême amour et la bonté qu'elle m'a offert pour me voir réussir.

A mon père
A l'homme que je dois ma réussite, mon bonheur, et tout le respect ;
Qu'il trouve ici l'expression de mon affection et une récompense des sacrifices consentis pour moi.

A mes frère et sœurs
En témoignage de mon profond amour, je vous souhaite le succès et le bonheur.

A ma grand-mère et toute la famille
En témoignage de mon profond amour et respect.

A mes amis
A tous mes ami(e)s, tous ceux que j'aime, tous ceux qui m'aiment et tous ceux qui me sont chers.

Introduction générale

Introduction générale

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.
Elle est définie comme l'ensemble des méthodes et techniques rationnelles d'analyse et de synthèse des phénomènes d'organisation utilisables pour élaborer **de meilleures décisions,** tout en proposant des modèles conceptuels permettant d'analyser et de maitriser des situations complexes pour permettre aux décideurs de comprendre et d'évaluer les enjeux afin de faire les choix les plus efficaces.
Cette discipline reçoit son nom pour la première fois en Angleterre et prouve son efficacité par un rapprochement de scientifiques et de militaires chargés de préparer les grandes décisions liées aux opérations militaires pendant la seconde guerre mondiale.

L'application de la Recherche Opérationnelle s'est élargie dans cette dernière décennie à divers domaines comme l'économie, la finance, le marketing et la planification d'entreprise. Plus récemment, elle a été utilisée pour la gestion des systèmes de santé et d'éducation, pour la résolution de problèmes environnementaux et dans d'autres domaines d'intérêt public.

Dans le secteur industriel, la matière première est une composante très importante lors du calcul des coûts de production. Il est donc prioritaire de minimiser les chutes (pertes) et d'améliorer sans cesse son utilisation dans l'intérêt de réduire les coûts, et pour aller plus loin réduire les impactes environnementaux engendrés par l'accumulation des déchet.
Le développement important des activités logistiques provoque des problèmes de conditionnement. Le fait de devoir placer d'énormes quantités de produits dans des véhicules (problèmes de transport) ou dans des centres de stockages a l'air d'une mission simple, mais en réalité, trouver le meilleur placement à moindre coût est une mission plus difficile que ce qu'on croit.

Découpe et conditionnement, deux variantes d'un problème d'intérêt majeur en Recherche Opérationnelle qui est le problème de **Placement**. Ce problème consiste principalement à ranger des articles dans des contenaires de la manière la plus économique possible. Les contraintes liées aux objets et aux contenaires varient selon la dimension du problème. Il comporte trois versions selon la dimension : la première en une dimension (1D), la deuxième en deux dimensions (2D) et la troisième en trois dimensions (3D).
Ces problèmes appartiennent à la catégorie des problèmes NP-difficile dans la classification de la complexité des problémes.

Dans ce mémoire, on étudiera le problème de placement en une dimension. C'est un cas déjà très utile en pratique, on trouve son application dans divers domaines on peut citer le

nombre minimal de CD-ROM pour stocker le contenu d'un disque dur, découper des planches de longueurs variables dans des grandes planches de longueur fixée, découpe dans des bandes de tissu, etc….).

Dans le premier chapitre nous introduisons des notions de l'optimisation combinatoires, ainsi que la théorie de complexité et nous citons les différentes classes des problèmes combinatoires.

Dans le deuxième chapitre nous définirons les problèmes de placement avec des exemples.

Le troisième chapitre sera consacré à la présentation des méthodes de résolution des problèmes de placement.

Le dernier chapitre sera consacré à une application que nous avons implémentée avec le langage pascal, et nous avons fait une simulation.

Nous terminons notre mémoire par une conclusion générale.

Chapitre I

Introduction à l'Optimisation Combinatoire

Introduction

L'Optimisation Combinatoire constitue une branche des mathématiques appliquées liée à l'algorithmique et à la recherche opérationnelle. C'est une discipline d'importance majeure, vu la difficulté des problèmes qu'elle contient d'une part, et le nombre d'application pratiques qui peuvent être formulé sous forme de problème d'optimisation combinatoire de l'autre part. Elle consiste à trouver une meilleur solution dans un ensemble fini et discret appelé « l'espace de recherche », qui est définit par une liste de contraintes, puis passer par la fonction objectif pour vérifier l'optimalité.

De prime abord, repérer les meilleures solutions dans un ensemble fini et discret à l'air simple, il suffit d'effectuer la comparaison de toutes les solutions possibles pour trouver celle qui est la plus adaptée et la plus efficace au problème étudié. Toutefois, en pratique, cette technique requiert énormément de temps. Or, c'est à cause de ce temps de recherche assez long que l'optimisation combinatoire est désignée comme étant complexe et difficiles. En effet, la plupart de ses problèmes appartiennent à la classe des problèmes NP-difficiles et ne peuvent être résolu avec exactitude.

Les stratégies d'optimisation combinatoire font aujourd'hui partie des moyens auxquels les dirigeants d'entreprises ont recours pour faciliter la prise de décision ou la conception de solutions devant un cas de figure inhabituel ou complexe. Elles ont maintes fois prouvé leur efficacité que ce soit dans le secteur de la production, de l'organisation ou encore en cas de soucis économiques ou informatiques.

I.1 formulation d'un problème d'optimisation combinatoire [1]

Un problème d'optimisation combinatoire se définit à partir d'un triplet (E, p, f) tel que:

- E est un ensemble discret appelé espace de solution ou encore espace de recherche.
- p est un prédicat sur E, i.e. une fonction de E dans {Vrai, Faux}.
- f : E →R associe à tout élément x □ E un coût f (x), f est appelé fonction objectif ou fonction de coût.

$$max_{x\epsilon E} f(x) = - min_{x\epsilon E} (-f(x))$$

L'ensemble des solutions admissibles du problème, noté E*a*, est défini tel que

Ea= {x □ E tel que p(x) est Vrai}.

Il s'agit de trouver l'élément x □ E*a* qui minimise f : f (x') = x□E f(x)
Le problème d'optimisation combinatoire consistant à chercher un élément maximum est de même nature.

I.2 Exemples de problème d'optimisation combinatoire

- Problème de remplissage optimal (sac-a-dos et placement).
- Problème de découpe.
- Problème de tournées de véhicules.
- Problème de transport.
- Problème de conception de l'emploi du temps.
- Problème d'ordonnancement.
- Problème d'affectation...etc.

I.3 Généralités sur la Théorie de Complexité [3]

La Théorie de la complexité constitue l'étude de la difficulté des problèmes (traités/résolus par des algorithmes). Lorsqu'un algorithme *A* résout un problème, on peut toujours se demander s'il y a un algorithme *A!* Qui résout aussi le problème mais qui est plus efficace. Il s'agit donc, d'abord d'évaluer, comparer, optimiser, les performances des algorithmes, en basant sur une estimation de temps, de calcul de l'espace mémoire occupé par l'exécution de l'algorithme. Généralement on considère le temps de calcul nécessaire pour la résolution.
La théorie de la complexité vise à savoir si la réponse à un problème peut être donnée très efficacement, ou au contraire être inatteignable en pratique.

Dans le but de mieux comprendre comment les problèmes se placent les uns par rapport aux autres, la théorie de la complexité établit des hiérarchies de difficulté entre les problèmes algorithmiques, dont les niveaux sont appelés des *« classes de complexité »*. Ces hiérarchies comportent des ramifications, suivant que l'on considère des calculs déterministes – l'état suivant du calcul est « déterminé » par l'état courant – ou non déterministes.

Exemples

- comparaison des algorithmes de tri : quel algorithme de tri vaux-il mieux utiliser pour trier les éléments d'un tableau ?
- si on lance le calcul de la factorielle de 100, combien de temps faudra t-il pour attendre le résultat?

Définition1 [4]

La Complexité d'un algorithme A est une fonction CA(N) donnant le nombre d'instructions exécutées par A dans le pire des cas pour une donnée de taille N (le paramètre de complexité). La complexité est basée généralement sur le pire cas, afin de borner le temps d'exécution de l'algorithme.

I.3.1 Types de complexité

Deux types de complexité peuvent être cités:

- **La complexité méthodologique** qui exprime une fonction du nombre d'opérations élémentaires de calcul effectuées par la méthode ou par l'algorithme de résolution en fonction du nombre des données du problème traité.
- **La complexité problématique** liée à la difficulté du problème à résoudre et au nombre des opérations élémentaires qu'un algorithme déterministe peut effectuer pour trouver l'optimum en fonction de la taille du problème.

I.3.2 Mesures de la complexité d'un algorithme

On peut citer quelques règles de calcule

1. Le temps d'exécution d'une affectation ou d'un test est considéré comme constant;
2. Le temps d'une séquence d'instruction est la somme des temps des instructions qui la composent;
3. le temps d'un branchement conditionnel est égal au temps des maximums des alternatives ;
4. Le temps d'une boucle est égal au produit du temps des instructions contenues dans la boucle par le nombre de passages dans celle-ci.

I.3.3 Les Problèmes d'existence (PE)

Vu que la Théorie de complexité est basée sur des outils de logique mathématique, elle ne traite que des problèmes d'existence (PE) : ce sont des problèmes dont les résultats ne peuvent prendre que les valeurs Oui-Non (0,1).
Ceci n'est pas trop gênant pour les problèmes d'optimisation combinatoire, puisqu'on peut associer à chaque problème un problème d'existence.

- **Définition2 [2]**

On appelle classe de complexité, un ensemble de problèmes dont la résolution nécessite la même quantité de ressources.

I.3.4 Les différentes classes de complexité [6]

On distingue deux principales classes des complexités : La classe des problèmes P et la classe des problèmes NP, nous développerons brièvement ces notions.

– **Classe des problèmes P**

Un Problème appartient à la classe P, si et seulement s'il existe un algorithme déterministe en temps polynomial (par rapport a la taille de l'instance n), permettant de répondre OUI au problème. On qualifie ce problème de polynomial, et on dit qu'il est facile.

- **Exemples**
  - Le test de primalité AKS est un algorithme qui montre que le problème de savoir si un entier est premier ou non peut être résolu par un algorithme polynomial.
  - Programmation linéaire : Les algorithmes de points intérieurs permettent de classer la programmation linéaire dans la classe **P**.

– **Classe des problèmes NP**

La classe NP réunit des problèmes pour lesquels la vérification d'une solution peut être décidée par un algorithme non-déterministe en un temps polynomial par rapport à la taille de l'instance n.

Intuitivement, les problèmes NP sont tous les problèmes qui peuvent être résolus en énumérant l'ensemble des solutions possibles et en les testant avec un algorithme polynomial. Il existe deux catégories de cette classe, les problèmes **NP-complet** et les problèmes **NP-difficiles**.

- **Exemple**
- Le problème du voyageur de commerce appartient à la classe des problèmes NP.
- **Les problèmes NP-complet :** cette classe contient la partie la plus difficile de NP .Un Problème d'existence est dit NP-complet s'il appartient à NP et si tout problème de NP peut être transformé de façon polynomiale en ce problème. Ce sont les problèmes les plus étudiés. Ceci parce que beaucoup de problèmes intéressant sont NP-complet et que l'on ne sait pas résoudre un problème NP-complet efficacement à cause du non déterminisme.

Un problème de décision est NP-complet s'il vérifie les propriétés suivantes :

- Il est possible de vérifier une solution efficacement (en temps polynomial) ;
- Tous les problèmes de la classe NP se ramènent à celui-ci via une réduction polynomiale ; cela signifie que le problème est au moins aussi difficile que tous les autres problèmes de la classe NP.

  Bien qu'on puisse *vérifier* rapidement toute solution proposée d'un problème NP-complet, on ne sait pas en *trouver* efficacement.

  - **Exemples**
  - Problème du voyageur de commerce,
  - Problème du sac à dos,
  - Problème de coloration de graphe …

▪ La notion de réduction polynomiale est très importante pour établir le NP complet d'un problème de décision donné.

▪ **Les problèmes NP-difficiles :** Un problème d'optimisation combinatoire est dit NP-difficile si et seulement si le Problème d'Existence associé est NP-complet.

✓ **Remarque :** Les problèmes NP-complet appartiennent aux problèmes NP-difficile par contre un problème NP-difficile n'est pas nécessairement dans NP**.**

I.4 Les Méthodes de résolution d'un problème d'optimisation combinatoire [3]

Les méthodes de résolution des problèmes d'optimisation combinatoire sont classées en deux catégories: les méthodes exactes et les méthodes approchées.

I.4.1 Les Méthodes de résolution exactes

Les méthodes de résolution exactes ou (complètes) examinent de manière implicite la totalité de l'espace de recherche. Ces méthodes de résolution exactes sont nombreuses, elles ont l'avantage de produire une ou plusieurs solutions dont l'optimalité est garantie lorsqu'aucune contrainte de temps n'est donnée.

Néanmoins, le temps de calcul nécessaire pour atteindre une solution optimale peut devenir vite prohibitif; ce malgré les diverses techniques et heuristiques qui ont été développées pour accélérer l'énumération des solutions.

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), le temps de calcul nécessaire pour trouver une solution risque d'augmenter exponentiellement avec la taille du problème.

On trouve parmi ces méthodes : les méthodes de la programmation linéaire, la méthode de branch and bound, la méthode hongroise, la méthode de la programmation dynamique,…etc.

Nous allons présenter ici deux méthodes exactes :

- **Programmation Linéaire**

Les problèmes de programmation linéaires (PL) sont des problèmes d'optimisation ou la fonction objectif, et les contraintes sont toutes linéaires. La programmation linéaire désigne également la manière de résoudre les problèmes linaires. La programmation linéaire est un domaine central de l'optimisation, car les problèmes de PL sont les problèmes d'optimisation les plus facile toutes les contrainte y étant linéaire beaucoup de problèmes réelle de recherche opérationnelle peuvent être exprimé comme problèmes de PL.

- **La technique de la programmation linéaire**

La mise en œuvre de la technique de programmation linéaire peut être subdivisée en cinq étapes.

Première étape

Identification du problème comme étant solvable par la programmation linéaire. Cette identification est le fruit d'une expérience dans la modélisation mathématique de problème.

Deuxième étape

Formulation du problème réel avec utilisation d'un modèle mathématique linéaire. Elle se fait en collaboration avec le décideur qui pose le problème.

Troisième étape

Résolution du problème théorique en utilisant des techniques algorithmiques.

Quatrième étape

Détermination d'une solution réelle à partir de la solution théorique.

Cinquième étape

Vérification de la validité de la solution et modification, si nécessaire de la formulation mathématique. Cette étape permet d'affiner le modèle afin d'apporter une solution acceptables par le décideur.

- **Méthode graphique**

Cette méthode consiste à représenter l'ensemble des contraintes sur un repère cartésien. Les contraintes ou apparaissent des inégalités correspondent géométriquement à des demi-plans. Intersection de ces demi-plans donnera ensemble des variables satisfaisant à toutes les contraintes. L'ensemble des contraintes est un polygone convexe.

- **La méthode du simplexe**

La méthode de simplexe est un algorithme de recherche d'une solution optimale d'un programme linéaire donné. La mise en œuvre de la méthode du simplexe peut être divisée en trois étapes :

Première étape : Mettre le modèle sous forme standard en y introduisant des variables d'écart qui ont pour rôle de transformer les inégalités en égalités.

Deuxième étape : Etablir le premier tableau de simplexe (tableau à l'origine).

Troisième étape : Procéder une série d'itérations sur les tableaux de simplexe aboutissant à la solution optimale.

Algorithme du simplexe

Pas 0 : *initialisation*

Mettre le problème sous forme standard (contraintes d'égalités et variables positives).

Trouver une première solution de base réalisable (utiliser la méthode section suivante si nécessaire).

Si aucune solution de base réalisable n'existe, alors STOP. Le problème est irrésoluble

***Pas 1** : Choix de la variable entrante*

✓ *Choisir comme variable entrante une variable hors base qui améliore la valeur courante de l'objectif.*

✓ *Si aucune variable hors base n'est candidate, alors STOP. La solution de base courante est une solution optimale.*

– **Méthode dual**

L'optimisation d'un programme linéaire à l'aide de la méthode de simplexe nous oblige parfois à introduire des variables artificielles pour obtenir une solution de départ lorsque les contraintes sont de type $AX \leq b$ ($b \geq 0$).

La méthode des phases et la M-méthode ont était utilisées pour optimiser un programme linéaire ayant des variables artificielle, afin de réduire la taille de PL et le nombre d'itération en générale, on utilise l'algorithme dual du simplexe (qui n'utilise pas de variables artificielles).

– **Algorithme dual de simplexe**

• Mettre le problème sous forme standard (contraintes d'´egalit´es et variables positives)

• Trouver une solution de base dual-réalisable (tous les couts relatifs cj sont positifs (négatifs) dans le cas d'un problème de minimisation (maximisation))

Pas 1 : Choix de la variable sortante

• Choisir comme variable sortante une variable en base dont la valeur est strictement négative. Soit $\boldsymbol{x}_{ir}$ cette variable, $\boldsymbol{b}_r$ la valeur de cette variable et r le numéro de la contrainte qui donne cette valeur.

• Si aucune variable de base n'est strictement négative, alors STOP. La solution de base dual réalisable courante est primal réalisable et optimale.

❖ **La Programmation Dynamique**

La Programmation Dynamique est une méthode exacte de résolution de problèmes d'optimisation, due essentiellement à R. Bellman (1957). Bien que très puissante, son cadre d'application est relativement restreint, dans la mesure où les problèmes qu'elle traite doivent vérifier un principe dit *principe d'optimalité* qui stipule qu'une solution optimale d'un problème de taille *n* peut s'exprimer en fonction de la solution optimale de problèmes de taille inférieure à *n*. La programmation dynamique consiste à placer le problème dans une famille de problèmes de même nature mais de difficulté différente, puis à trouver une relation de récurrence liant les solutions optimales de ces problèmes. Elle est basée sur le principe de Richard BELLMAN « Toute politique optimale ne peut être formée que de sous politiques optimales »

Le plus gros du travail réside dans l'expression d'une solution d'un problème en fonction de celles de problèmes "plus petits" (formule de récurrence). Si on se rend compte qu'on est amené à recalculer plusieurs fois la solution de mêmes problèmes, on est dans le cadre de la programmation dynamique. Le nombre de sous problèmes peut être grand et l'algorithme obtenu n'est pas forcément polynomial**.**

I.4.2 Les méthodes approchées [9]

Typiquement ce type de méthodes, dites **heuristiques** est particulièrement utile pour les problèmes nécessitant une solution en temps réel (ou très court) ou pour résoudre des problèmes difficiles sur des instances numériques de grande taille. Elles peuvent aussi être utilisées afin d'initialiser une méthode exacte.

Parmi ces méthodes, il faut distinguer les heuristiques ciblées sur un problème particulier et les **méta-heuristiques** plus puissantes et adaptables pour résoudre un grand nombre de problèmes.

Les méthodes approchées permettent d'obtenir une solution de bonne qualité (c'est-à-dire assez proche de l'optimum) dans un contexte de ressources (temps de calcul et/ou mémoire) limitées. Dans ce cas l'optimalité de la solution ne sera pas garantie.

Les heuristiques et les méta-heuristiques sont le compromis entre le temps de résolution et les qualités des résultats.

Quelques heuristiques : les heuristiques gloutonnes (l'algorithme de Dijkstra pour le plus court chemin), la méthode de la descente,…etc.
Quelques méta-heuristiques : méthode de recherche Tabou, méthode de recuit simulé, les algorithmes de colonie de fourmis, les algorithmes génétiques…etc.
Nous allons présenter quelques heuristiques :

- **La Recherche Tabou**

Bien que son origine remonte à 1977, la Recherche Tabou n'est proposée qu'au milieu des années 80 par Fred Glover. Cette méthode, développée pour résoudre des problèmes combinatoires, la plupart NP-difficiles, propose de surmonter le problème des optima locaux par l'utilisation d'une mémoire.

La méthode tabou est une procédure itérative qui, partant d'une solution initiale, tente de converger vers la solution optimale en exécutant, à chaque pas, un mouvement dans l'espace de recherche. Chaque pas consiste d'abord a engendré un ensemble de solutions voisines de la solution courante pour ensuite en choisir la meilleure, même si ce choix entraîne une augmentation de la fonction objectif à minimiser.
En acceptant de détériorer la valeur de la solution courante, le minimum local peut être évité mais, en contre partie, des parcours répétitifs sont déplorés.
Aussi, pour palier à l'inconvénient majeur des méthodes de recherche locale, la recherche Tabou a pour but d'améliorer à chaque étape, la valeur de la fonction objectif, en utilisant une mémoire afin de conserver les informations sur les solutions déjà visitées.
Cette mémoire constitue la liste Tabou qui va servir à interdire l'accès aux dernières solutions visitées. Lorsqu'un optimum local est atteint, il y a interdiction de revenir sur le même chemin.
-Un critère d'aspiration, est également utilisé pour lever l'interdiction d'utilisation d'un mouvement si ce dernier conduit à une meilleure solution.
Plusieurs stratégies ont été proposées récemment afin d'améliorer l'efficacité de la méthode tabou. L'intensification et la diversification de la recherche constituent deux d'entre elles.
-L'intensification consiste à explorer en détails une région de l'espace de recherche jugée prometteuse. Sa mise en œuvre consiste, le plus souvent, en un élargissement temporaire du voisinage de la solution courante dans le but de visiter un ensemble de solutions partageant certaines propriétés.
- **La diversification** a pour objectif de diriger la procédure de recherche vers des régions inexplorées de l'espace de recherche. La stratégie de diversification la plus simple consiste à

redémarrer périodiquement le processus de recherche à partir d'une solution, générée aléatoirement ou choisie judicieusement, dans une région non encore visitée de l'ensemble des solutions admissibles.

Les domaines d'application de la recherche tabou sont vastes et variés, ils passent de l'ordonnancement à la robotique, au problème du voyageur de commerce, à l'électronique voire même aux applications médicales.

- **Les algorithmes génétiques [1]**

L'algorithme génétique est une classe d'adaptation stochastique des algorithmes d'optimalisation de la recherche et l'optimisation. Ils ont d'abord été utilisés par les Pays-Bas (1975). L'idée de base est d'essayer d'imiter une simple image de la sélection naturelle, en vue de trouver un bon algorithme.

La première étape consiste à muter, ou varier de façon aléatoire, une collecte de l'échantillon programs. La deuxième étape est une étape de sélection, qui est souvent fait par le biais de la mesure contre une fonction de remise en forme. Le processus est répété jusqu'à ce qu'une solution appropriée soit trouvée.

Il existe un grand nombre de différents types d'algorithmes génétiques. L'étape de mutation dépend de la façon dont l'échantillon programs sont représentés, ainsi que de savoir si le programmeur comprend diverses techniques de liaison.

Le test d'aptitude est également en hausse pour le programmeur. Comme un gradient de flux optimisation, il est possible pour que le processus reste bloqué dans un maximum local de la fonction de remise en forme.

L'un des avantage d'un algorithme génétique est qu'il ne nécessite pas de fonction de l'aptitude à être très bon, car une recherche aléatoire est effectué au lieu de la suivre la voie de la moindre résistance .mais pour réussir, il faut qu'il y'ait de belles relations entre les paramètres modifiables à la remise en forme.

- **Méthode gloutonne [4]**

Une méthode gloutonne consiste à fixer à chaque étape la valeur d'une variable sans remettre en cause les choix effectués précédemment. Par exemple, une heuristique gloutonne bien connue pour la coloration introduite par Brélaz est la suivante : pour choisir le nœud suivant à colorier, prendre celui dont les nœuds adjacents sont déjà coloriés avec le plus grand nombre de couleurs différentes et lui assigner la couleur autorisée de plus petit rang possible. Les méthodes gloutonnes sont généralement rapides, mais fournissent le plus souvent des solutions de qualité médiocre. Elles ne garantissent l'optimum que dans des cas particuliers.

Conclusion

Dans ce chapitre nous avons abordé l'optimisation combinatoire d'une manière globale, nous avons commencé donner une définition et quelques exemples des problèmes qu'elle traite.

Puis nous avons définit la théorie de complexité qui est une notion de base, elle permet la classification des problèmes d'optimisation combinatoire en deux catégorie : les problèmes faciles et les problèmes difficiles.

On a déduit que les problèmes combinatoires sont généralement NP-difficiles ou NP-complets.

Etant donné l'importance de ces problèmes, de nombreuses méthodes de résolution on été développées.

Méthodes exacte : programmation linéaire, programmation dynamique, Branch and bound.

Méthodes approché : Heuristique, Méta-heuristique, Recherche tabou, Le recuit simulé, Algorithmes génétiques, Méthode gloutonne.

Chapitre II

Généralités sur le problème de Placement

Introduction

Le problème de placement est un problème d'optimisation combinatoire d'intérêt majeur qui intervient dans des problématiques diverses. Le problème concret de placement se pose lorsque l'on cherche à remplir une ou plusieurs boîtes avec un ensemble fini d'objets ou lorsque l'on cherche à obtenir un ensemble fini d'objets en découpant un ou plusieurs objets de taille supérieurs et cela de la manière la plus économique possible. Dans le premier cas, il s'agit d'un problème de remplissage, et dans le deuxième cas il s'agit d'un problème de découpe.

II.1 Formulation mathématique d'un problème de placement

Le problème de placement est défini de la façon suivante : étant donné un ensemble de n objets, $Y = Y_1, ..., Y_n$ et un nombre illimité de boîtes (rectangles identiques), de dimensions plus larges que celles des objets. Le problème consiste à déterminer le nombre minimum de centenaires à utiliser pour ranger l'ensemble de tous les objets sans chevauchement.

II.2 Modèle mathématique

Le problème de placement peut être modélisé de la manière suivante :

- un ensemble de n, d'objets qu'on appel boîtes.

  $B=B_1, B_2,\ldots,B_j,\ldots, B_n$, .

- un ensemble de n d'objets $Y=Y_1, Y_2,\ldots Y_i,\ldots, Y_n$.

- W_j : La longueur de la boite B_j.

- w_i : La longueur de l'objet Y_i.

- $x_{ij} = \begin{cases} 1 \text{si l'objet } Y_i \text{ est range dans la boîte } B_j \\ 0 \text{ sil' objet } Y_i \text{ ne peut être rangé dans la boîte } B_j \end{cases}$

  $b_j = \begin{cases} 1 \text{ si la boite } B_j \text{ est utilisée} \\ 0 \text{si la boite } B_j \text{ n'est pas utilisée} \end{cases}$

On cherche à minimiser le nombre de boites à utiliser, c'est-à-dire :

$$Min \quad \sum_{j=1}^{n'} b_j$$

$b_{j= \{0,1\}, j=1,.., n'}$

On suppose que les Boîtes ont la même taille, c'est-à-dire :

$\forall j, k \in N, W_j = W_k$

- **Contraintes du problème :**

1. Un Objet est placé uniquement dans une seule Boîte.

$$\begin{cases} \sum_{j=1}^{n'} x_{ij} = 1 \\ xij \in \{0,1\} \\ i=1,..,n \end{cases}$$

x_{ij}=1 impose à tous les objets d'être rangés dans une boite et une seule.

2. La longueur de l'ensemble des objets rangés dans une boite B_j ne doit pas dépasser la longueur de ceci.

$$\sum_{i=1}^{n} w_i * x_{ij} \leq W_j$$

L'inégalité signifie qu'on ne peut dépasser la taille d'une boite pour un rangement.

A noter que la partie droite de l'inégalité oblige b_j à prendre la valeur 1 dès qu'un article est rangé dans la boite $_j$.

Toute solution pour laquelle les deux équations sont vérifiées est dite réalisable.

Le modèle est donc le suivant :

$$\begin{cases} Min \quad \sum_{j=1}^{n'} b_j \\ \sum_{j=1}^{n'} x_{ij} = 1 \\ \sum_{i=1}^{n} w_i * x_{ij} \leq W_j \\ x_{ij} \in \{0,1\} \\ i=1,..,n \; ; \; j=1,..,n \end{cases}$$

La modélisation décrite plus haut a été proposée par Leonid Kantorovitch en 1960. Il existe d'autres formulations linéaires pour ce problème, sous forme d'un problème de flot maximum dans un graphe, ou utilisant une décomposition de Dantzig Wolfe.

Exemple

Supposons que nous avons des récipients de taille 10. Quel nombre d'entre eux sont nécessaires pour ranger les poids de la taille 3, 6, 2, 1, 5, 7, 2, 4, 1, 9?

Une solution est d'utiliser 4 boites et arranger les objets comme suit

-premier casier : objets 2 ,8

-deuxième casier : objets 4,10

-troisième casier : objets 1,6

-quatrième casier : objets 3, 5, 7,9

-troisième boite : objets 5, 4, 8.

II.3 Classification et contraintes pratiques

Il existe un grand nombre de variantes pour le problème de placement. Cependant chaque problème réel présente ses propres spécificités telles que :

- **Les caractéristiques propres aux objets**

- objets de formes homogènes ou non homogènes.
- objets de tailles uniformes ou différentes.
- objets déformables ou non déformables.
- etc.

- **Les spécificités propres au problème**

- le nombre de dimensions du problème.
- disposer d'une seule boite c (problème de décision ou problème de maximisation).
- minimiser le nombre de boites à utiliser.
- minimiser la surface ou le volume global des objets à placer.
- etc.

- **Les contraintes propres au problème**

- l'équilibre entre les objets.
- l'ordre dans lequel les objets doivent être retirés des boites.
- l'orientation d'un objet.
- le poids (par exemple, le poids d'une boite complète ne peut pas excéder une limite donnée).

- le placement : certains objets très lourds doivent être placés en bas, d'autres fragiles doivent être placés en haut.
-etc.

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

• le nombre de dimensions du problème.
• le type de tâche : tous les objets et une sélection de boites, ou bien une sélection d'objets et toutes les boites.

• les caractéristiques des boites : une seule boite, des boites de tailles identiques, ou bien des boites de tailles différentes.

II.4 Classification des problèmes de placement [11]

Il y a trois grandes familles de problèmes selon le nombre de dimensions des objets :

II.4.1 problème de placement en une dimension

Le placement en une dimension est la version standard des problèmes de placement, il consiste le rangement un ensemble d'objets caractérisés par une seule variable (hauteur, largeur, poids, ou autre) dans un ensemble de boites de taille fixé W sans chevauchement.
Une instance de placement 1D notée D, et une paire (Y,W) où $Y = \{Y_1,..., Y_n\}$ est une la liste des articles à ranger et *W* une valeur positive.

Chaque article Y_i possède une longueur w_i inferieur à W.

La valeur optimale du nombre de boites nécessaires pour ranger tous les objets d'une instance **D** est notée OPT(**D**).
Le cas le plus fréquent consiste à utiliser des barres de métal existant dans une entreprise pour satisfaire des demandes en barres différentes. On cherche à minimise ma longueur des chutes non réutilisables (inférieur à une longueur minimale).

II.4.2 problème de placement en deux dimensions

Plus formellement, le problème de placement en deux dimensions est défini de la façon suivante : étant donné un ensemble de n objets rectangulaires $Y = \{Y_1,..., Y_n\}$ et un nombre illimité de rectangles identiques (boîtes) de dimensions plus larges que celles des objets, les (w_i, h_i) les dimensions d'un objet ai appartenant à l'ensemble des rectangles à ranger A, et (W, H) les dimensions d'un boites B : B = (W, H). Nous considérons, sans perte de généralité, que les dimensions des objets et des boites sont des entiers. Une instance de placement 2D notée **I** est alors définie par la paire (A, B). La valeur optimale du nombre de boites nécessaires pour ranger tous les objets d'une instance **I** est notée OPT(I).
Ces problèmes se posent en particulier chez les fabricants de verre, de tôles, et chez les fabricants de vêtements. Il s'agit de placer des commandes rectangulaires sur des formes rectangulaires de formats standards en minimisant le nombre de formes à découper ou la surface des chutes non réutilisables obtenues.

II.4.3 Problème de placement à trois dimensions :

Dans ce cas les données sont les suivantes :

Il existe « N » objets de volume $v_{i \text{ et M boites}}$ d'un volume V_j (éventuellement des boites toutes identiques de volume V) Le volume total des objets est plus petit que le volume total des boites

$$(\sum v_i \leq \sum V_j)$$

II.4.4 Exemples de problèmes de placement

1. Sauvegarde de données sur un fichier informatique

On souhaite sauvegarder le contenu d' disque dur sur des CD-Rom de 700 Mo. Chaque dossier a une taille donnée, et on ne souhaite pas découper les dossiers. Comme il ne nous reste plus beaucoup de CD, on cherche le minimum de disques qu'on peut utiliser. Notez bien que dans cette version du problème, on ne cherche pas à regrouper spécialement des fichiers par genre. On a un problème de placement en une dimension à résoudre !

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 placement à une dimension.

3. Organisation d'une fête

On a une fête à organiser, et on doit inviter un certain nombre de familles, chaque famille est composée d'un ou plusieurs personnes, Comme il ne nous reste pas beaucoup d'argents, on cherche le minimum de tables à utilisé pour ce dîner. Sachant qu'on ne peut pas séparer une personne de sa famille.

Il semble avoir un problème de placement à une dimension.

4. Réalisation des taches

On doit réaliser un ensemble de taches à faire, chaque tache doit être exécutée dans un temps quelconque. Des machines ont utilisées pour réaliser ces taches dans un délai fixé. On a besoin de minimiser les machines utilisées sans passer le délai !

-Tableau : caractéristiques des problèmes de placement-

Type de problèmes	Critères	Dimension	Domaine d'application
Découpe	Minimiser -les chutes -la matière première	-1dim -2dim	Découpes de pièces dans -l'aéronaval -la confection -la papeterie -l'ébénisterie -la maroquinerie -la sidérurgie -la verrerie …
Placement de taches dans le temps	Maximiser -la rentabilité Optimiser -l'utilisation des ressources	1dim	Emploi du temps -d une école -d'un artisan -d'un atelier …
Utilisation des surfaces	Minimiser -les surfaces utilisées	-1dim -2dim	-étalage en grandes surfaces -machines outils dans un atelier -puces utiles sur une tranche de silicium -palettes ou étagères dans un entrepôt -palettes sur une machine automatiques …

Remplissage de volumes	Minimiser -l'espace utilisé -Déplacement pour atteindre le colis	-1dim -2dim -3dim	Objets dans un entrepôt -caisses dans un wagon, un camion, un cargot -colis dans une caisse -pièces dans un four en sidérurgie
Maximisation du poids transporté	Maximiser -le poids embarqué	-1dim	-Remplissage de cargots ou de camions par les objets lourds(le critère volume est secondaire)
Remplissage de rayons	Maximiser l'utilisation des rayons	-1dim	-Grandes surfaces -bibliothèque -entrepôts
Gestion de mémoire par zones	Minimiser les zones inaccessibles	-1dim	-informatique
Choix des investissements	Maximiser les bénéfices	-1dim	Investissements - banques - sociétés

II.5 Problème du Sac à Dos

Le problème du sac à dos (en anglais, Knapsack problem) est un problème classique de l'optimisation combinatoire appartenant à la classe des problèmes NP-complets.

Il modélise le remplissage par objet un sac à dos qui ne peut supporter plus qu'un certain poids, chaque objet ayant un poids et une valeur définis.

Les objets mis dans le sac à dos doivent alors maximiser la valeur totale, sans pour autant dépasser le poids maximum.

✓ Modèle mathématique

L'énoncé du problème est simple, étant donné un ensemble de n objets ou chaque objet est caractérisé par un poids w_i et un profit p_i.

On cherche les sous ensemble d'objets à charger dans un sac de capacité C afin de maximiser la somme des profits.

Ainsi le problème du sa à dos se présente sous la forme mathématique suivante

$$(KP)\begin{cases} Max \quad \sum_{i=1}^{n} p_i x_i \\ \sum_{i=1}^{n} w_i * x_i \leq C \\ x_i \in \{0,1\},\ i \in \{1,..,n\} \end{cases}$$

Les poids wi et les profits pi ainsi que la capacité C sont des entiers positifs $_{i \in \{1,n\}}$

La variable x_i est la variable de décision ; elle prend la valeur 1 si l'objet est chargé dans le sac, sinon elle prend la valeur 0.

Résoudre ainsi le problème du sac à dos revient à trouver le vecteur solution :

$X^*=(x_1 *,..,x_n*)^T$ qui maximise la fonction objectif.

Le problème du sac à dos et ses différentes variantes on été longuement étudiés depuis le travail pionnier de Dantzing en 1957. L'intérêt porté au problème du sac à dos est du au fait qu'il permet de modéliser de très nombreux problèmes comme les problèmes de chargement de capital, de chargement de cargaison, de rotation de l'équipage, de tournées et de livraison, par ailler ce problème apparait comme un sous problème de nombreux problèmes d'optimisation combinatoire.

Le problème de Placement et le problème du sac a dos sont des problèmes entièrement différent même s'il y'a un lien très fort entre les deux. En effet le problème de placement consiste à la recherche de minimum de boites que peuvent contenir les objets, et celui du sac à dos consiste à la recherche du maximum d'objets que peuvent contenir une seule boite en respectant l'ordre d'importance.

Chapitre III

Méthodes de résolution pour les problèmes de Placement

Introduction [10]

Comme la plupart des problèmes d'optimisation combinatoire, le problème de placement étant NP-difficile, l'énumération de toutes les solutions réalisables pour trouver la meilleure solution s'avère impossible même pour un problème de taille moyen.
Toutefois, on peut aborder la résolution par des méthodes approchées, en particulier des heuristiques et des méta-heuristiques.

III.1 Méthodes de résolution pour le problème de placement

III.1.1 Méthodes exactes [2]

Le problème de placement à été largement étudié dans la communauté de recherche opérationnelle. Il existe plusieurs méthodes exactes pour le résoudre comme la programmation linéaire, la programmation dynamique, la procédure par séparation et évaluation PSE et la méthode de génération des colonnes …
A titre d'illustration nous allons présenter ici la méthode de branch and bound pour le problème de placement.

❖ Procédure par séparation et évaluation

La procédure par séparation et évaluation (PSE), ou en anglais branch and bound, est un algorithme qui permet d'énumérer intelligemment toutes les solutions possibles. En pratique, seul les solutions potentiellement de bonne qualité seront énumérées, les solutions qui ne peuvent pas conduire à améliorer la solution courante ne sont pas explorées.
Pour présenter une PSE, nous utilisons un « Arbre de recherche » constitué

- Des nœuds ou sommets, ou un nœud représente une étape de construction de la solution
- d'arcs pour indiquer certains choix faits pour construire la solution.

Dans notre exemple, les nœuds représentent une étape pour laquelle des objets auront été mis dans les boîtes et d'autres pour lesquelles aucune décision n'aura encore été prise.
Chaque arc indique l'action de mettre un objet dans la boîte courante ou, au contraire, de ne pas le mettre dans la boîte. Les nœuds sont étiquetés par une liste d'ensemble chacun correspond à une boîte, de plus le dernier ensemble de la liste correspond à boîte que l'on est entrain de remplir. La figure suivante représente l'arbre de recherche du problème donné en exemple

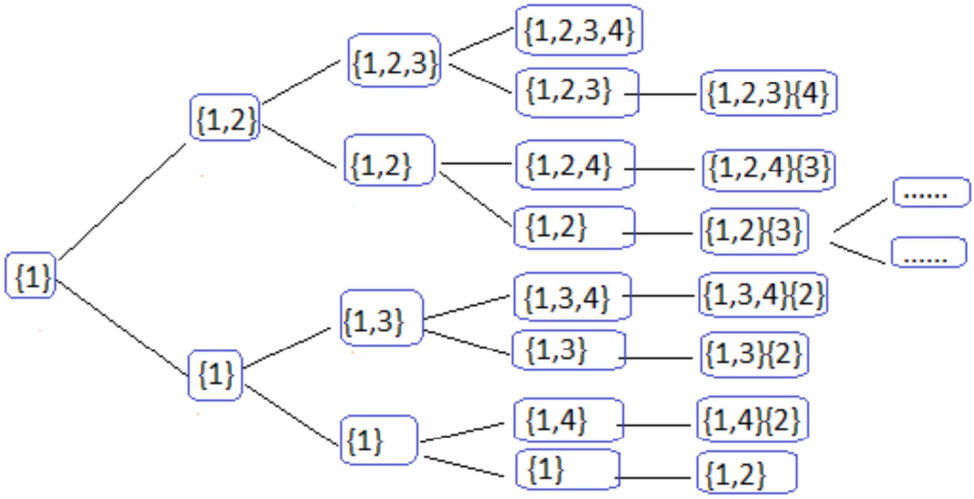


Figure1 : L'exploration des solutions dans un arbre de recherche

Au départ, on cherche à remplir la première boîte. Tout d'abord on lui affect l'objet n°1 : comme il faudra bien que cet objet soit contenu dans une boîte, on décide en quelque sorte de nommer 1 la boîte le contenant. La racine de l'arbre, dont la profondeur est par définition 0, contiendra donc l'ensemble {1} qui représente le contenu de la 1ère boîte. Puis pour un nœud de profondeur i on construit deux fils : celui du haut ou l'on ajoute l'objet i+1 dans la boîte et celui du bas ou la boîte reste tel quel. Lorsque l'on a considéré tous les objets pour un boîte donné, on poursuit la construction de l'arbre en passant au boîte suivant et en lui affectant initialement l'objet non sélectionné dont l'indice est le plus petit on poursuit ainsi la construction de l'arbre jusqu'à avoir rangé tous les objets. Dans l'arbre de recherche achevé, chaque feuille représente une solution potentielle mais forcement réalisable. Dans le schéma, les feuilles au bord épais représentent les propositions irréalisables car supérieures au poids maximal a ne pas dépasser.
Pour déterminer la solution, il suffit de calculer le nombre de boîtes pour chaque nœud feuille acceptable et de prendre la solution ayant la plus petite valeur.

- Cependant la taille de l'arbre de recherche est exponentielle en le nombre d'objets, et il n'est pas question d'implanter un tel arbre en mémoire. Aussi il existe de nombreuses techniques algorithmiques de parcours de ce type d'arbre. Ces techniques ont pour but d'augmenter la rapidité du calcul en diminuant la taille de l'arbre de recherche. Par exemple on peut remarquer que le poids du nœud interne {1, 2,3} dépasse déjà le poids maximal, il n'était donc pas nécessaire de développer l'étape suivante. Les PSE permettent d'élaguer (éliminer les ensembles de solutions pour lesquels il n'existe pas d'optimum) encore plus cet arbre en utilisant des bornes inferieures et supérieures de la fonction objectif
- Une borne inferieure est une valeur minimum de la fonction objective. Autrement dit c'est une valeur qui est nécessairement inferieur a la valeur de la meilleure.

III.1.2 Heuristiques de résolution pour le problème de Placement en une dimension (1D)

Comme dans tous les problèmes combinatoires, des méthodes heuristiques ont été proposées pour le problème de placement dans le but de trouver des solutions de bonne qualité dans un temps raisonnable sans garantie d'optimalité.
Heureusement pour le problème en une dimension on trouve des algorithmes avec garantie de performance qui sont connus sous le nom de First-fit (FF) ou Best-fit (BF), Next-fit et Worst-fit(WF), on parle de first-fit decreasing (FFD) ou best-fit decreasing (BFD), next-fit decreasing (NFD) et worst-fit decreasing(WFD) quand il s'agit de trier les objets à ranger dans l'ordre décroissant.

III.1.2.1 Stratégie Next-Fit(NF)

Dans cette stratégie on ne considère qu'une boite ouverte à la fois.
Les objets sont traités selon un ordre donné. Les objets sont rangés successivement dans la boite ouverte tant qu'il y a de la place pour l'objet en cours, sinon la boite en cours est fermée et une nouvelle est ouverte.
-Une heuristique qui adopte une stratégie NF à l'avantage d'avoir une complexité temporelle linéaire en fonction du nombre d'objets a placé.

Par contre, le fait de ne considérer qu'une seule boite à la fois cause beaucoup de perte d'espaces exploitables.
-On parle de Next-Fit-Decreasing (NFD) quand il s'agit de trier les objets à ranger dans l'ordre décroissant de hauteurs avant de les ranger suivant une stratégie NF.

III.1.2.2 Stratégie First Fit (FF)

Initialement une seule boîte est considérée, et les objets sont traités selon un ordre donné. Quand il n'y a plus de la place dans la première boîte pour ranger l'objet en cours, une deuxième est alors ouvert mais sans fermer l première.
-Dans une étape intermédiaire où on dispose de k boites ouvertes numérotées de 1 à k selon l'ordre de leur première utilisation, un objet a_i en cours est rangé dans la boite du plus faible numéro qui peut le contenir.
-Dans le cas où aucune boite ne peut contenir une nouvelle boite (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.

III.1.2.3 Stratégie Best Fit (BF)

Comme dans la stratégie FF, les algorithmes BF laissent les boites toujours ouvertes. Cependant, le choix de la boite dans laquelle l'objet a_i en cours va être placé dépend des valeurs des gaps (espace libre restant dans la boite) (hauteurs non utilisées) présentes dans les boites.

Ainsi, sera placé dans la boite qui présente le moindre gap parmi les boites qui peuvent le contenir.

-Les heuristiques BF et FF peuvent être implantées en O (nlog (n)) en utilisant une structure de données appropriée Johnson (1973).

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

III.1.2.4 Stratégie de Worst-Fit

Cette approche est similaire à Best Fit mais on place un élément dans une des boites ouvertes pour que le reste de la capacité disponible de la boite soit maximum.

-on parle de worst fit decreasing (WFD) quand il s'agit de trier les objets à ranger dans l'ordre décroissant.

III.1.3 Exemple d'application

Exemple

Supposons que nous avons des bacs de taille 10.

Quel nombre d'entre eux sont nécessaires pour stocker les poids de la taille 3, 6, 2, 1, 5, 7, 2, 4, 1, 9?

Voici le résultat d'application du Next-Fit à l'exemple

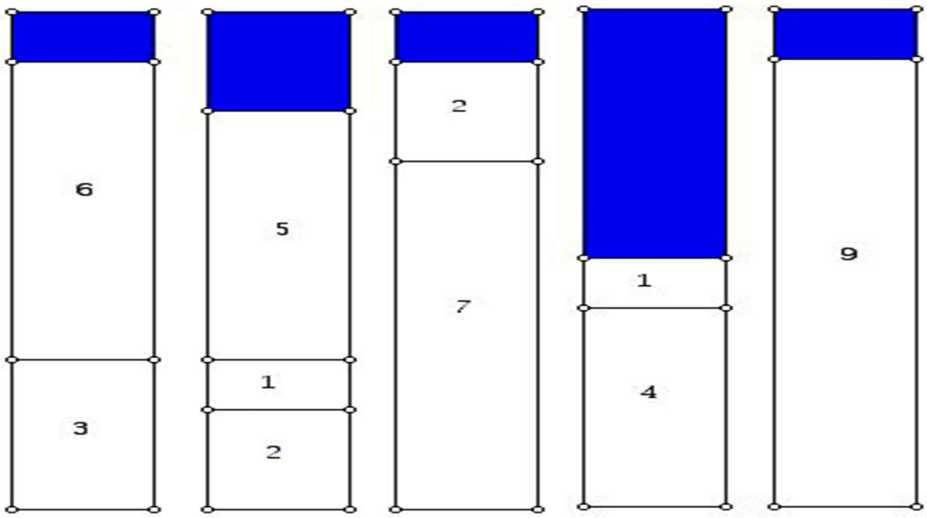


Figure2 : placement suivant la procédure Next-fit

L'aspect attrayant de Next-Fit est qu'il est très simple et, dans certains paramètres appliqués du problème de placement optimal, il permet de débarrasser les boîtes rapidement, car même s'il y a de la place supplémentaire dans une boites , nous n'attendons pas autour dans l'espoir qu'un article viendra plus tard dans la liste qui remplira cet espace vide. On peut imaginer avoir une flotte de camions avec une restriction de poids (la capacité C) et un poids de paquets dans les camions. Si le poids suivant ne peut pas être emballé dans le camion au quai de chargement, ce camion part et un nouveau camion s'insère dans le quai. Comme Next Fit est effectué, tout ce qui est nécessaire, c'est de garder une trace de la quantité restante dans la boite ouverte à ce moment-là. En termes de temps nécessaire pour trouver le nombre de bacs pour n poids, on peut répondre à la question en utilisant une procédure qui prend un temps linéaire dans le nombre de poids (n). De toute évidence, Next-fit ne produit pas toujours un emballage optimal pour un ensemble donné de poids. Vous pouvez vérifier cela en trouvant un moyen d'emballer les poids dans l'exemple 1 dans 4 boîtes.
Une pensée naturelle serait que si nous voulons garder les boîtes ouvertes dans l'espoir que nous pourrons remplir l'espace vide avec des éléments plus tard dans la liste L, nous utiliserons généralement moins de boîtes. La façon la plus simple de réaliser cette idée est le placement avec First Fit.

Le résultat de la réalisation de First Fit pour la liste de l'exemple

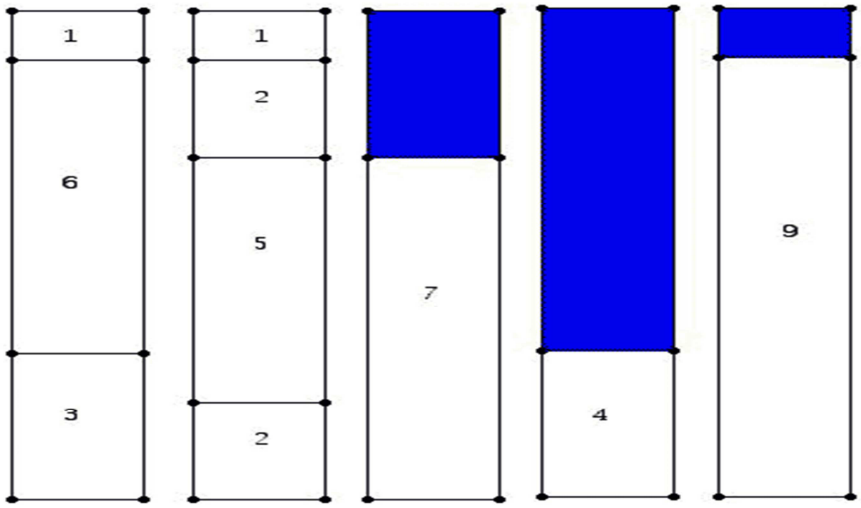


Figure 3 : placement suivant la procédure First-fit

Jusqu'à présent, les deux méthodes que nous avons essayées ont donné 5 boîtes. Nous savons que ce n'est pas le meilleur que nous pouvons espérer. ***Un aperçu simple est obtenu en calculant la somme totale des poids et en divisant ce nombre par la capacité des boîtes.*** Comme nous traitons des nombres entiers, le nombre de boîtes dont nous avons besoin doit être au moins équation où l'équation $\Omega = \sum_{i=1}^{40} poids$ (Notez que la fonction Plafond indique le plus petit nombre entier supérieur ou égal à x). De toute évidence, le nombre de boîtes doit toujours être un nombre entier. Dans l'exemple que nous traitons, puisque Omega est 40 et C est 10, on peut conclure qu'il y a de l'espoir d'utiliser seulement 4 boiites. Cependant, ni Next Fit ni First Fit n'atteint cette valeur avec la liste donnée dans l'exemple 1. Peut-être nous avons-nous besoin d'une meilleure procédure.

Si on continue le placement avec les deux procédures Best-Fi et Worst-Fit on trouvera que Best Fit donnera le même emballage que l'emballage First Fit et le Worst Fit emballant le même que l'emballage Next Fit, bien qu'en général ce ne sera pas toujours vrai. Notez que dans notre exemple, le fait qu'un élément de poids lourd (poids 9) s'est produit en tant que dernier élément de la liste, était impitoyable en essayant d'utiliser la liste donnée pour emballer les éléments. En termes de compromis avec Next Fit, le temps nécessaire pour trouver le nombre minimum de boîtes en utilisant FF, WF ou BF est plus élevé que pour NF.

❖ Remarques

Ces algorithmes ne sont pas optimaux, mais ils permettent d'obtenir de très bons résultats en pratique.

- Les algorithmes Best Fit Decreasing et First Fit Decreasing n'utilisent jamais plus de 11/9 **OPT** + 1 boîtes (où **OPT** est le nombre optimal de boîtes dans une solution optimale) La procédure de tri est la partie la plus coûteuse de l'algorithme, mais sans elle, la qualité de la méthode est beaucoup moins bonne. On obtient dans ce cas des solutions utilisant au pire 17/10 **OPT** + 2 boîtes.
- Une version plus efficace de FFD utilise au plus 71/60 **OPT** + 1 boîtes.

III.2 Heuristiques de résolution pour le problème de Placement en deux dimensions

Les méthodes proposées pour le problème en deux dimensions sont en général des heuristiques.

Ces derniers peuvent être divisés en deux familles différentes, les algorithmes en une phase et les algorithmes en deux phases.

-Les algorithmes en une phase consistent à ranger directement les objets dans les boîtes, tandis que les algorithmes à deux phases commencent d'abord par ranger les objets dans une boîte de hauteur infinie. Par exemple en cherchant une solution pour le problème de (2D) suivant :

Étant donné l'ensemble **A** d'objets à placer et une boite unique de largeur **W** et de hauteur infinie, l'objectif est de ranger tous les objets de **A** dans la boite en minimisant la hauteur totale utilisée.

Dans la deuxième phase, un algorithme de résolution pour le problème en une dimension consiste à utiliser la solution du 2D obtenue pour construire le rangement dans les boîtes à utiliser effectivement

III.2.1 Méthodes en une phase

Les méthodes en une phase consistent à ranger les articles itérativement dans les boîtes. Deux règles à définir : l'ordre dans lequel les articles sont examinés, la boîte et la position dans laquelle on cherche à le placer.

-Parmi les algorithmes qui procèdent en une phase, nous trouvons quelques uns qui placent les objets par niveaux comme l'algorithme First Fit (FF). Cet algorithme 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 de la première boîte qui peut le contenir.

-Si aucun niveau ne peut contenir l'objet en cours un nouveau niveau est créé dans la première boîte appropriée ou bien en initialisant une nouvelle boîte.

Parmi les algorithmes qui ne rangent pas les objets par niveau, nous trouvons essentiellement ceux qui adoptent une stratégie connue sous le nom de 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.2.2 Méthodes en deux phases

Les Méthodes en deux phases fonctionnent de la manière suivante

La première étape (Emballage en bande) 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.

Le premier objet, le plus long, est placé dans le premier niveau qui correspond à l'arête inférieure de la boîte.

Les autres objets sont rangés suivant une stratégie NF, FF ou BF par rapport à la largeur de la boîte-9, on parle respectivement d'une stratégie NFDH, FFDH et BFDH.

Lodi et Al ont proposé une approche pour le rangement par couches. Cette approche s'appelle Floor Ceiling (FC), où la première phase consiste à placer les objets par couches suivant la stratégie BFDH avec la modification suivante :

Dans la deuxième phase, les niveaux sont placés dans des boîtes finis selon une stratégie BFD, ou bien en utilisant une méthode exacte pour le placement en une dimension.

-Une autre approche dite (Emballage en sac à Dos) a été proposée par les auteurs.

-Dans cette approche, on ne considère qu'un niveau à la fois.

Un niveau initialisé est rempli en entier en résolvant un problème de sac à dos en une dimension

-Dans un problème de sac à dos en une dimension, comme pour le placement en une dimension, on dispose d'une boîte de hauteur H et d'un ensemble d'objets de hauteurs données. A chaque objet ai est associé un profit.

-L'objectif est de ranger dans la boite un sous-ensemble d'objets qui maximise la somme des profits associés.

- Le profit associé à chaque objet est la surface dans la méthode du sac à dos.

La deuxième phase est identique à celle employée dans la méthode FC. Burke et al ont proposé une nouvelle heuristique pour le problème d'emballage en bande en deux dimensions avec la possibilité de tourner les objets de 90 degrés. Ils adoptent une stratégie BF :

- A chaque étape du rangement d'un objet, l'espace disponible le plus en bas est considéré. L'objet dont une des dimensions remplie au mieux cet espace y est placé.

-Un algorithme en deux phases appelé Hybrid First Fit (HFF) a été proposé par Chung et al. (1982). Dans la première phase, le problème d'emballage en bande est construit suivant une stratégie FFDH.

Soit H1 H2 HNL les hauteurs des NL niveaux résultants. Observons que

H1 H2 HNL puisque les objets ont été triés par ordre décroissant des hauteurs.

-Dans une deuxième étape, une solution du problème de placement bidimensionnel initial est obtenue en résolvant un problème de placement unidimensionnel avec NL objets ai (i = 1, ..., NL) de taille (W, Hi) chacun correspondant à un niveau (la boîte étant de taille (W, H)).

-Une variante de l'algorithme HFF a été proposée par Berkey et Wang (1987), elle est appelée Hyprid Best Fit (HBF). Elle consiste à employer la stratégie BFDH pour résoudre le problème de d'emballage ne bande (première phase).
-Pour la résolution du problème 1BP de la deuxième phase (ranger les niveaux obtenus dans des bins finis) la stratégie BFD est employée.

III.3 Le problème d'emballage en bande

Le problème d'emballage en bande est une variante très étudiée des problèmes de placement bidimensionnels. On dispose d'une liste A d'articles et d'un conteneur unique de largeur W et d'une hauteur infinie. L'objectif est de ranger tous les articles de A dans les boîtes en minimisant la hauteur totale à utiliser.

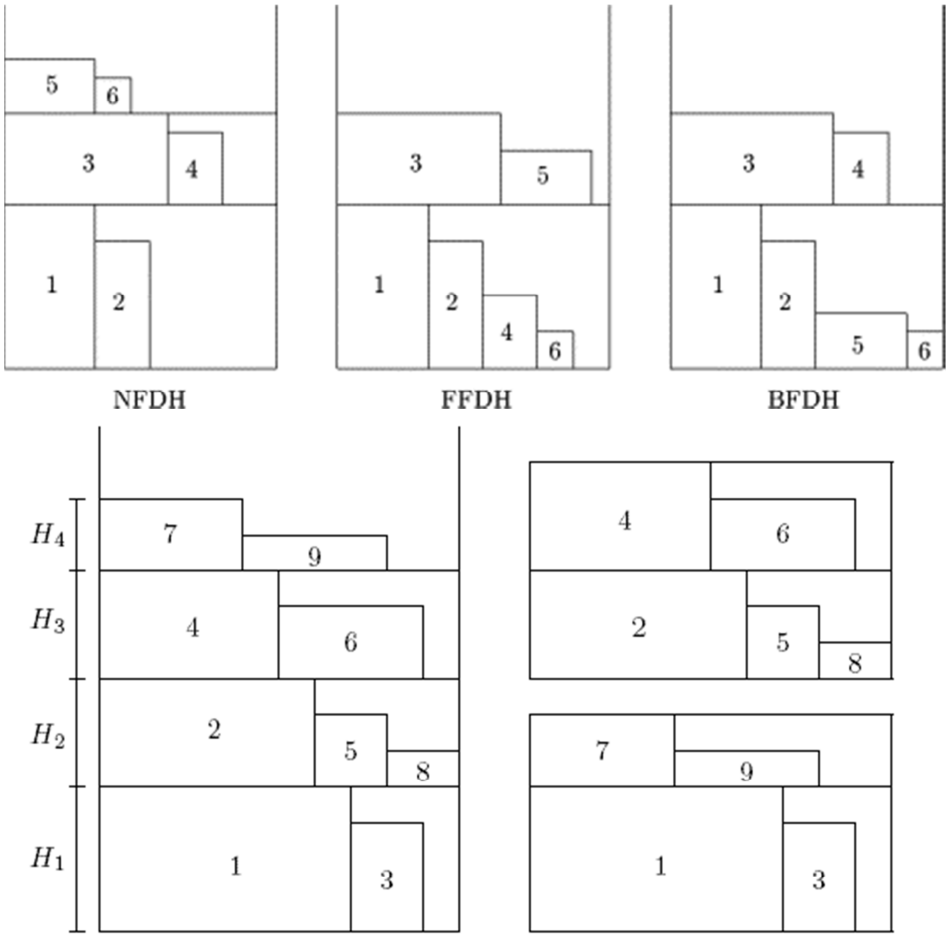


Figure 4 : illustration d'un problème d'emballage en bande et sa résolution

Conclusion

Dans ce chapitre nous avons présenté les méthodes de résolution d'un problème de placement.

En premier lieu on a parlé des méthodes de résolution exactes, avec comme exemple la méthode de branch and bound pour le problème de placement, puis nous avons passer au heuristiques de résolution d'un problèmes unidimensionnel et leurs explications (First-fit ;Next-fit ;Best-fit ;Worst fit), avec un exemple d'application.

Dans un second lieu nous avons abordé les heuristiques de résolution pour un problème bidimensionnel, et on a cité quelques méthodes relatives à ces problèmes.

Chapitre IV

Implémentation de Algorithme Best Fit Decreasing

Introduction

La résolution du problème placement à une dimension, utilise des méthodes exactes et des méthodes approchées que nous avons développées dans le chapitre précédent.
Dans ce chapitre, on s'intéressera à une application pratique qu'on résoudra avec une heuristique qui est l'algorithme de Best Fit Decreasing qu'on programmera avec le langage pascal, puis on fera une simulation pour voir comment se comporte BFD, en variant le nombre d'objets et la capacité des boites.

IV.1 Méthodologie d'implémentation de l'algorithme BFD

L'implémentation de l'heuristique BFD, se fera sous l'environnement PASCAL.
On utilisera un fichier objet qui contiendra les objets à placer qui sont caractérisés par leur numéro et leur taille.
La création de ce fichier se fera à l'aide d'une procédure qui remplira ce dernier soit :
1-D'une manière aléatoire dans le cas ou on désire évaluer BFD pour un ensemble d'objets très grand.
2- En faisant la saisie, lorsqu'on est en face d'un problème pratique où la taille des objets à saisir est connu.
Les paramètres de cette procédure création sont :
Le nombre d'objets à placer, la capacité d'une boite où placer les objets, ainsi que la méthode de saisie.

La deuxième procédure étant l'application de l'heuristique BFD, qui utilisera le fichier des objets crée qu'elle exploitera pour créer une liste linéaire triée selon la taille des objets par ordre décroissant, un objet dans cette liste est caractérisé par son numéro, sa taille (valeurs prise directement du fichier) et le numéro de la boite dans laquelle il sera placer, cette valeur sera remplis lors du placement de l'objet dans la boite, cette liste sera automatiquement triée par ordre décroissant grâce aux trois procédure suivante :

1- Insère début : insérant un élément au début de la liste.

2- Insert milieu : insérant un élément au milieu de la liste dans un endroit précis.

3- Insert fin : insérant un élément à la fin de la liste
Puis elle calcule le nombre de boites optimales(OPT) nécessaires pour placer les objets en sommant la taille de tous les objets divisés par la capacité de la boite.

Elle procédera ensuite au placement des objets dans les boites, elle utilisera pour cela un tableau qui contiendra la capacité de chaque boite, cette capacité qui est au départ égale à la taille de la boite, mais sera mise à jour à chaque affectation d'un objet.

Le placement se fera en affectant l'article dans la boîte la plus complète qui l'accueillera, c'est-à-dire celle qui laissera le moins d'espace restant. Si aucune boite n'est trouvée, démarrez une nouvelle boite jusqu'à épuisement des objets. A la fin elle fournira le nombre de boîtes nécessaires pour placer tous les objets.

IV.2 Algorithme

1-Création du fichier objet

Si simulation alors

Saisie aléatoire

Sinon (cas concret)

Saisie par l'utilisateur

2-Création de la liste linéaire

Tant que non fin du fichier

-Prendre un objet

Si objet à l'élément

Insert début (objet)

Sinon si objet < au dernier élément

Insert fin (objet)

Sinon insert milieu

FIN (fin de tant que)

3-(Placement des objets dans les boites de capacité = CAP)

T=0

Tant que non fin de liste

J=1, T=T+taille, optbfd=j

While taille > capacité

J=j+1,

Fin While

Si j>optbfd alors

Optffd =j

finsi

Capacité := capacité – taille

Nbin=j ;

On prend un autre objet.

Fin Tant que

OPT = T / CAP (théorique)

Afficher les deux optimums (opt et optbfd)

Afficher la répartition des objets par bin .

FIN.

IV-3 exécution du programme

1- Application

Nous allons appliquer ce programme sur une application dont la taille de la boite est de 1400 et le nombre d'objets est de 16 dont les tailles sont respectivement : 26, 35,52,77,88,94,137,164,253,364,372,388,406,432,461,851

On exécutera notre programme sur micro ordinateur PC doté d'un système d'exploitation windows seven 32 bits ou l'environnement PASCAL fonctionne sans problème. .

La figure 1 suivante donne l'écran d'exécution de ce programme; elle montre la création du fichier objet.

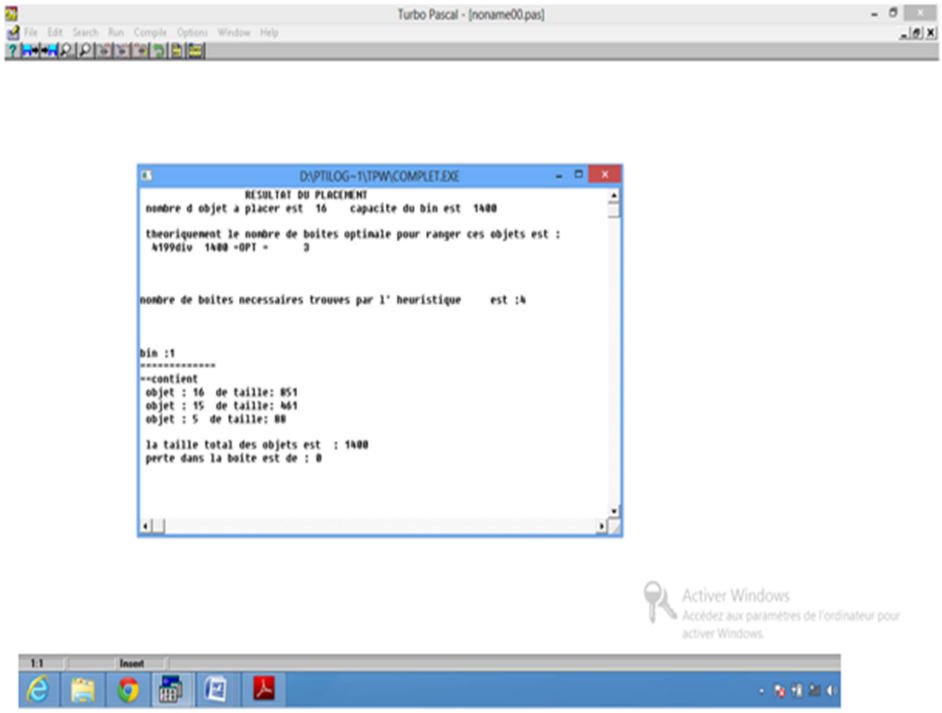

```
D:\PTILOG-1\TPW\COMPLET.EXE
 CREATION DU FICHIER DES OBJETS

1:  aléatoire  ===> PAS DE SAISIE LA TAILLE SERA PRIS ALEATOIRE
2:  saisie =======> SAISIE DE LA TAILLE DES OBJETS
3:  executer FFD SANS SAISIE
-----------------> VOTRE CHOIX S V P  --->2
 saisissez le nombre d'objets --->16
saisissez la taille de la  boite --->1400
creation d un fichier contenant 16 objet de taille < à  1400
saisissez la taille de l'objet 1  26
saisissez la taille de l'objet 2  35
saisissez la taille de l'objet 3  52
saisissez la taille de l'objet 4  77
saisissez la taille de l'objet 5  88
saisissez la taille de l'objet 6  94
saisissez la taille de l'objet 7  137
saisissez la taille de l'objet 8  164
saisissez la taille de l'objet 9  253
saisissez la taille de l'objet 10  364
saisissez la taille de l'objet 11  372
saisissez la taille de l'objet 12  388
saisissez la taille de l'objet 13  406
saisissez la taille de l'objet 14  432
saisissez la taille de l'objet 15  461
saisissez la taille de l'objet 16  851
```

La figure 2 donne Les résultants obtenus avec BFD :

-Le nombre de boite optimal (OPT).

-Le nombre de bins obtenus par la méthode BFD.

-La répartition des objets par boites.



Interprétation des résultats

L'heuristique BFD donne une solution égale à 4 boites pour stocker les 16 objets, cette solution n'est pas optimale mais proche de l'optimum qui est égale à 3.

Comme prévue une heuristique donne souvent des solutions approchées mais son avantage est qu'elle s'exécute pour un grand nombre de variable donnée, chose difficilement réalisable avec une méthode exacte. Pour le montrer on simulera une série d'exécution sur un certains nombre d'objets tirés aléatoirement qu'on placera sur divers boites de capacités différentes.

2- Simulation

Dans cette partie on essayera de voir le comportement de l'heuristique BFD,

En variant le nombre d'objets et la capacité des boîtes.

Les résultats de cette simulation sont résumés dans le tableau suivant.

Cette heuristique fournit le nombre de boites optimaux théorique OPT, ainsi que le nombre de boites BFD.

Ces résultats sont obtenue par l'exécution notre programme.

NBRE OBJETS	50		100		250		1000		1500		5000		10000	
TAILLE DE LA BOITE	OPT	BFD	OPT	BFD	OPT	BFD	OPT	BFD	OPT	BFD	OPT	BFD	OPT	BFD
10	22	28	49	55	128	147	504	517	752	760	2506	2515	5006	5029
20	25	32	52	61	127	146	501	513	755	763	2498	2509	4965	4979
50	24	30	53	63	125	143	500	512	749	756	2501	2512	5031	5090
100	24	30	54	64	128	147	494	504	751	758	2505	2516	5022	5049
200	26	33	50	62	130	149	504	517	753	761	2510	2532	5067	5104
500	27	34	64	72	125	143	489	499	755	763	2530	2548	5039	5069
1000	27	34	57	69	128	147	502	514	748	755	2495	2505	5021	5047

Les résultats obtenus en appliquant BFD sont assez proche de l'optimum théorique et ce quel que soit le nombre de variables ainsi que la variation de la capacité d'une boite.

Conclusion générale

Conclusion générale :

Nous avons traité dans ce mémoire le problème de placement unidimensionnel, qui est un modèle mathématique très attrayant, et pourtant, travailler sur ce problème est étonnamment récent.

Le placement optimal en tant que sujet organisé, n'a que 35 ans environ. Les principaux pionniers et contributeurs de ce problème sont « Edward Coffman, Jr », « Michael Garey », « Ronald Graham »et « David Johnson ».

L'importance de ce problème réside par le domaine vaste de ses applications concrètes en transport - logistique et en différentes industries (papier, bois,…) ainsi que dans d'autres domaines d'intérêt publique.

Le problème de placement est un problème NP-Complet, c'est-à-dire il n'existe pas une méthode exacte qui nous permet de trouver la solution optimal dans un temps raisonnable sauf si on utilisé une méthode de parcoure de tous l'ensemble des solutions réalisable.

Ce modeste travail nous a permet de cerner et de comprendre les problèmes de l'optimisation combinatoire, les méthodes de résolution ainsi que les classes de complexités.

On a ensuite traité un problème d'optimisation combinatoire à savoir le problème de placement en présentant les diverses heuristiques de résolution puis d'implémenter l'heuristique Best Fit Decreasing, l'application informatique aussi simple que soit elle nous a permet aussi de voir la rapidité de l'obtention des résultats et de confirmer que les solutions données sont assez proches de la solution optimale et ça grâce aune simulation des données généré de manière aléatoire.

Annexe

Programme SOURCE de l'application

```
 programbestfit;
useswincrt;
TYPE
OBJET = record
NUM:integer;
TAILLE:INTEGER;
end;
LISTE=^num ;
num= record
n:integer;
tail:integer;
bin:integer;
suiv:liste
end;
fichier =file of objet;
VAR
bf:fichier;
e:OBJET ;

mat1,i,n,repo:longint;
rep,y,nbo,ta,ca:longint;r:char;
a:array[1..1000]of integer;
opt:real;
max,l,tt,j,nn:LONGint;
el,dern:LONGint; p2, p1,tet,p:liste;
Procedure creation (var f: fichier; varnbo,Ca:longint;y:integer);
Begin

writeln ('creationd un fichier contenant ', nbo, ' objet de taille < à ', Ca);
randomize;
assign (f,'emp');
rewrite(f);
if y=1 then
begin
i:=1;
repeat
e.NUM:=i; n:=random(ca);
if n=0 then
i:=i-1

else

repeat
```

```
write('saisissez la taille de l''objet ',i,' ');
readln(ta);
until ta<= ca ;
e.taille:=ta;
write(f,e); i:=i+1;
until i>nbo ;
end;
close(f);
writeln( 'voulez vous afficher le fichier tapez 1 ? ');
readln(rep);
if rep=1 then begin
writeln('affichage du fichier');
reset (f);
i:=1;
while not eof(f) do
begin
read(f,e);
with e do
begin

writeln ('objet ',num, ' : ',TAILLE,' --- ');
i:=i+1;
if i=24 then
begin
writeln(' .../');readln;clrscr;i:=1;end;end;end;
close(f); end;
writeln(' ')
procedure ffd (var f:fichier);
procedure insdeb(vartet:liste; p:liste);
begin
p^.suiv:=tet;
tet:=p
end;
procedure insm(vartet:liste; r,p:liste);
var p1:liste;
begin
p^.suiv:=r^.suiv;
r^.suiv:=p

 end;
(* programme principale de la procedure FFD **)
BEGIN
reset(f);
```

```
writeln(' ');
tt:=0; i:=1; j:=1;
while i<=nbo do
begin
read(f,e);
tt:=tt+E.taille;
i:=i+1;
end;
if (tt mod ca =0) then
opt:=tt/ca
else
opt:=(tt div ca)+1;
writeln;
close(f);
(*==============================================*
)
(* CREATION DE LA LISTE TRIEE CONTENANT LES OBJETS AVEC LEUR
TAILLE RESPECTIF *)
RESET(f);

read(f,e);
new(tet); tet^.n :=e.num ; tet^.tail :=e.taille; tet^.suiv:=nil ; dern:=tet^.tail;
rep:=2 ;
while rep<=nbo do
begin if e.taille>= tet^.tail then
insdeb(tet,p)
else
if e.taille<dern then
begin
dern:=e.taille;
insfin(tet,p);
end
else
begin
p2:=tet;
while p2^.tail >e.taille do
begin
p1:=p2; p2:=p2^.suiv
read(f,e); new(p) ;p^.n:=e.num; p^.tail :=e.taille end;
insm(tet,p1,p);
end;
rep:=rep+1;
end;
```

```
(* affectation des bin aux objets *)
for i:=1 to nbo do
a[i]:=ca;
p:=tet;
p^.bin:=1;
a[1]:=ca-p^.tail;
p:=p^.suiv ;
max:= begin
i:=1;
while p^.tail > a[i] do
i:=i+1;
if i>max then
max:=i;
p^.bin:=i;
a[i]:=a[i]-p^.tail;
end ;
writeln; clrscr;
WRITELN (' RESULTAT DU PLACEMENT ');
writeln(' nombre d objeta placer est ',nbo, ' capacite du bin est ',ca);writeln; 1;
while p<> nil do writeln (' theoriquement le nombre de boites optimale pour
ranger ces objets est :');
writeln(' ' ,tt ,'div ',ca,' =OPT = ', opt:5:0);
writeln; writeln; writeln;
writeln('nombre de boites necessaires trouves par l'' heuristique FFD est :',
max); writeln;
readln;
writeln;
(* affichage par bin *)
for i:=1 to max do
begin
l:=0; writeln('bin :',i); writeln('=============='); writeln('==contient');
p:=tet;
while p<> nil do
begin
ifp^.bin =i then
begin
writeln (' objet : ',p^.n,' de taille: ', p^.tail); l:=l+p^.tail; end;
p:=p^.suiv;
end;
writeln;
writeln (' la taille total des objets est : ',l,' ');
writeln (' perte dans la boite est de : ',ca-l) ;
readln;
```

```
end; writeln('" au revoir et merci !!!"');
END;
BEGIN (**** PG PRINCIPAL ***)
writeln(' CREATION DU FICHIER DES OBJETS '); repo:=1;
while repo=1 do
begin
writeln(' ');
writeln('1: aléatoire ===> PAS DE SAISIE LA TAILLE SERA PRIS
ALEATOIRE ');
writeln('2: saisie ========> SAISIE DE LA TAILLE DES OBJETS ');
WRITELN ('3: executer FFD SANS SAISIE ');
WRITE ('==================> VOTRE CHOIX S V P ===>') ;

readln(y);
while(y<>1)and (y<>2) and (y<>3) do
begin
clrscr;
writeln('1: aléatoire ===> PAS DE SAISIE LA TAILLE SERA PRIS
ALEATOIRE ');



writeln('2: saisie ========> SAISIE DE LA TAILLE DES OBJETS ');
WRITELN ('3: EXECUTER FFD ');
WRITE ('==================> VOTRE CHOIX S V P ===>') ;
readln(y);
end;
if y=3 then
BEGIN
clrscr;
writeln(' ffd sans saisie ');assign(ff, 'emp');
reset (ff) ;write(' saisissez le nombre d''objets ===>');
readln(nbo);

write('saisissez la taille de la boite ===>');
readln(ca);
END
else
begin
write(' saisissez le nombre d''objets ===>');
readln(nbo);
write('saisissez la taille de la boite ===>');
readln(ca);
creation (ff,nbo,ca,y);end;
```

```
FFD (ff);
WRITELN ('Pour recommencer tapez 1 ?'); readln(repo);
clrscr;
end;end.
```

Bibliographie

[1] « étude des capacités de la meta-heuristique d'optimisation par la colonie de fourmis » Thèse de Doctorat

[2] M.R Garey, D.S Johnson et al. "Computers and Intractability: A guide to the theory of NP-completeness", Freeman, USA, 1979.

[3]Ben said Omar Cours des Méthodes de Résolution Exactes Heuristiques et Méta heuristiques Université Mohammed V, Faculté des Sciences de Rabat

[4] Nadia Brauner « Recherche Opérationnelle I »

[5] B. Mariou Complexité Algorithmique — Automne 2012

[6] Peter Van « complexité » , fichier4 Licence Informatique - Semestre 2 - Algorithmique et Programmation, Roy 26 /03/2015

[7] HAMDAOUI FAWZIA « CONTRIBUTION A LA RESOLUTION DUPROBLEME DU SAC A DOSMULTIDIMENSIONNEL A CHOIX MULTIPLE »MÉMOIRE En vue de l'obtention du Diplôme de Magistère

[8] K.Saddouni, « Optimisation combinatoire Avancée », Cours Magistral, USTOMB, 2016

[9] D.Du and P.M Pardalos "Handbbok of combinatorial optimization". Springer, 2007

[10] BEN MOHAMED AHMED « Résolution approchée du probléme de bin-packing » Thèse de Doctorat ,Université du Havre /2009.

[11] Amara Aida kenza « résolution numérique dans les problèmes de type bin packing »

Mémoire magister

Table des matières

Résumé

Pour accomplir ce travail nous avons commencé par un introduction aux problèmes d'optimisation combinatoire et les méthodes de résolution ainsi que les classes de complexités.
On a ensuite traité un problème d'optimisation combinatoire à savoir le problème de placement en présentant les diverses heuristiques de résolution puis d'implémenter l'heuristique Best Fit Decreasing, l'application informatique aussi simple que soit elle nous a permet aussi de voir la rapidité de l'obtention des résultats et de confirmer que les solutions données sont assez proches de la solution optimale et ça grâce aune simulation des données généré de manière aléatoire.