REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université Mohamed Boudiaf -M'sila
Faculté des Mathématiques et de l'Informatique
Département des Mathématiques

Mémoire de Master

Domaine : Mathématiques et Informatique
Filière : Mathématiques
Option : Analyse Mathématique et Numérique

Thème

Etude comparative entre deux algorithmes polynômiaux Pour un problème d'optimisation

Présentée par :
REDAOUI Ikram

Soutenu publiquement le :2020

Devant le jury composé de :

DILMI Mustapha	MCB	Université de M'sila	**Président**
SELT Omar	MCA	Université de M'sila	**Encadreur**
GAGUI Bachir	MCA	Université de M'sila	**Examinateur**

Année universitaire 2020

Remerciements

Je tiens à remercie, en premier lieu, **Mon Dieu** qui m'a donné la force de rédiger ce modeste travail.

Je tiens à remercie les membres du jury, qui ont accepté d'évaluer mon travail de mémoire.

Je tiens à remercie **SELT Omar** encadreur de mon mémoire, pour sa disponibilité et ses conseils judicieux tout au long de ce travail.

Je tiens à exprimer tout mes respects à ma mère, ma soeur qui m'ont toujours encouragée.

Je remercie tous les professeurs du département de Mathématiques, sans oublier aussi mes collègues et amies, ainsi tous ceux qui ont participé de loin ou de près à l'élaboration de ce mémoire.

Dédicaces

Je dédie ce modeste travail :

-A mes parents ma mère et mon pére.

- A mes soeurs

-A toute ma famille.

-A toutes mes amies.

NOTATIONS

P	Polynômail time
NP	non deterministic polynômial time
P'OC	problème d'optimisation combinatoire
PVC	le problème du voyageur de commerce
B&B	la méthode de Branch and Bound
RT	la méthode de recherche tabou
RS	la méthode de recuit simule
PL	programe linéaire
VB	les variables de base
VHB	les variables hors base

Table des matières

Introduction

Le domaine de l'optimisation combinatoire est un domaine très important, il est situé au carrefour de la recherche opérationnelle, mathématique et informatique. Son importance s'explique par la grande difficulté posée par les problèmes d'optimisation d'une part, et par le nombre important des applications pratiques pouvant être formulées sous forme de problèmes d'optimisation combinatoires d'autre part. L'optimisation combinatoire consiste à parcourir l'espace de recherche (l'ensemble de combinaisons pouvant être prises par les variables du problème) afin d'en extraire une solution optimale parmi un ensemble fini de solution, d'une taille souvent très grande telle que son énumération exhaustive est une tâche fastidieuse. En fait, un problème d'optimisation combinatoire se ramène à résoudre une de ses instances par un procédé algorithmique permettant la maximisation (en cas de problème de maximisation) ou la minimisation (n cas de problème de maximisation) d'une (ou de plusieurs) fonction(s) objectif(s) en respectant certaines contraintes rendant infaisable une partie de l'espace de recherche. En effet, la plupart de ces problèmes appartiennent à la classe des problèmes dites NP-difficiles et ne possèdent donc pas à ce jour de solution algorithmique efficace valable pour toutes les données.

En fait le plan de travail comme dans le premier chapitre, constitue l'état de l'art, ici on donne les définitions des différentes composantes de la théorie de l'analyse combinatoire, on donne la définition d'un problème d'optimisation combinatoire et quelques exemples, puis on parle de la complexité algorithmique où on classifié les problèmes en trois sous classes: **P** les problèmes polynomiaux là où les problèmes peuvent être résolus par un algorithme polynomial. La sous classe **NP**-complète là où des algorithmes polynomiaux n'ont pas été trouvés pour résoudre ce genre des problèmes. Puis la sous classe des problèmes ouverts qu'on ne sait à présent s'ils sont dans **P** on **NP**.

Pour le second chapitre : on considère les méthodes approchées et les méthodes exactes. Les méthodes approchées sont classées en deux catégories : les heuristiques et les métaheuristiques. Une

heuristique est une méthode approchées spécifique à un problème donné et les métaheuristiques sont des méthodes approchées polyvalentes, les métaheuristiques peuvent être classées en deux catégories : les métaheuristiques à base de solution unique et les métaheuristiques à base de population de solutions, dans cette méthode , il existe de nombreux types, nous les mentionnons **:** la recherche tabou qui s'inspire de la mémoire des êtres humains et le recuit simulé qui est inspiré d'processus métallurgique. Tandis que. Les méthodes exactes sont connues par le fait qu'elles garantissent l'optimalité de la solution mais elles demandent des coûts de recherche (temps de calcul et espace mémoire) souvent prohibitifs qui augmentent avec la taille de l'instance du problème traité, dans cette méthode , il existe de nombreux types, nous les mentionnons **:**méthode de simplexe et méthode de Branch and Bound.

Dans le chapitre 3: nous avons utilisé deux algorithmes sont: algorithme de minimisation, qui est l'order du plus petit au plus grand et algorithme de maximisation, qui est l'order du plus grand au plus petit. Le but des algorithmes est de résoudre le problème des tâches, de chaque tâche qui a du temps(T) et du poids (P). Ce qu'il fout, c'est calculer le poids dévissée par le temps (P/T) et comparer les résultats par l'ordonnancement

Dans algorithme de minimisation, nous mettons deux exemples: le premier exemples nous avons utilisé cinq tâche, et dans le deuxième exemple nous avons utilisé dix tâche, le même travaille pour le deuxième algorithme. Enfin, nous avons comparé entre les résultats de deux algorithmes.

Chapitre 1

Problème d'optimisation combinatoire

1.1 Introduction

Dans cette chapiter nous présentons quelques définitions sur problème l'optimisation et l'optimisation combinatoire, le voisinage, la théorie de la complexité, quelque problème d'optimisation.

1.2 Définition de l'optimisation

Définition 1.2.1 *:est un branche des mathématique cherchant à modéliser à analyser et à résoudre analytiquement ou numériquement les problèmes qui consistent à minimiser ou maximiser une fonction sur un ensemble. Beaucoup des systèmes susceptibles d'être décrits par un modèle mathématique sont optimisée.* ***(voir [14])***

1.3 Problèmes d'optimisation discrète

Les problèmes d'optimisation discrète, par opposition à l'optimisation continue, forment une classe de problèmes d'optimisation particulièrement étudiée. Tout ou partie des variables de ce type de problèmes appartiennent à l'ensemble des entiers, autrement dit

$X \subseteq Z_m \times R_{n-m}$, avec $0 \leq m \leq n$. Lorsque $m = n, P(X, f)$ est dit problème en nombres entiers, sinon il fait partie des problèmes d'optimisation mixtes en nombres entiers. Bien que l'ensemble réalisable soit plus restreint dans le cas de l'optimisation discrète, ces problèmes sont souvent plus

difficiles que leurs versions continues.Les ensembles réalisables des problèmes d'optimisation et des problèmes mixtes peuvent être infinis, et ceux des problèmes en nombres entiers sont au plus dénombrables. Les problèmes auxquels nous nous intéressons, à savoir les problèmes d'optimisation combinatoire. Ces problèmes sont connus pour leur difficulté bien qu'ils soient définis sur des ensembles admissibles finis. **(voir [7])**

1.4 L'optimisation combinatoire

L'optimisation combinatoire(**OC**) occupe une place très importante en recherche opérationnelle, en mathématiques discrètes et en informatique. 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.

Bien que les problèmes d'optimisation combinatoire soient souvent faciles à définir, ils sont généralement difficiles à résoudre. En effet, la plupart de ces problèmes appartiennent à la classe des problèmes **NP**-difficiles et ne possèdent donc pas à ce jour de solution algorithmique efficace valable pour toutes les données. L'optimisation combinatoire est minimiser (ou maximiser) une fonction souvent appelée fonction coût, d'une ou plusieurs variables soumises à des contraintes. Le sujet de l'optimisation combinatoire dans un domaine discret. Il faut trouver permit touts les possibilités, souvent en nombre fini, la possibilité optimale. Ceci parait facile mais devient infaisable dès que la taille du problème est suffisamment grande. La taille pour laquelle la recherche d'un optimum devient infaisable est petite, très souvent plus petite que la taille des problèmes pratiques. En général, la difficulté d'un problème grandit très vite avec le nombre des variables. Il n'est pas alors faisable d'examiner toutes les possibilités. Les méthodes d'optimisation peuvent être reparties en deux catégories :

1. Méthodes exactes.
2. Méthodes approchées. **(voir[3])**

1.5 Les problèmes d'optimisation combinatoire

Un problème d'optimisation combinatoire **(d'OC)** consiste à déterminer un plus grand (petit) élément dans un ensemble ni valué. En d'autres termes, étant donnés une famille $\mathcal{F}$ de sous-ensembles d'un ensemble ni $E = \{e_1, ..., e_n\}$ et un système de poids $w = (w \ \ (e_1), ..., w(e_n))$

associé aux éléments de E, un problème d'optimisation consiste à trouver un ensemble $F \in \mathcal{F}$ de poids $w(F) = \sum_{e \in F} w(e)$ maximum (ou minimum),

i.e. $\min ou \max \{w(F) \mid F \in \mathcal{F}\}$.

La famille $\mathcal{F}$ représente donc les solutions du problème. Elle peut correspondre à un ensemble de très grande taille que l'on ne connaît que par des descriptions ou des propri étés théoriques qui ne permettent pas facilement son énumération. **(voir [9])**

1.6 Problèmes d'optimisation combinatoire multi-objectif:

Nous définissons dans cette section le problème de recherche de solutions de bon compromis pour un problème d'optimisation combinatoire multi-objectif. Dans un premier temps nous rappelons les notions préliminaires de l'optimisation multi-objectif, ensuite nous précisons la notion de compromis et nous formalisons ces problèmes.

Un problème d'optimisation combinatoire multi-objectif peut s'écrire sous la forme du problème (1.1), où p représente le nombre d'objectifs et X un ensemble combinatoire

$$minx \in Xz(x) = (_{z_1}(x), ..., z_p(x)). \tag{1.1}$$

Afin de ne pas confondre les solutions réalisables et leurs images dans R_p, il est d'usage en optimisation multi-objectif de distinguer l'espace des décisions X, qui contient l'ensemble admissible X, et l'espace des objectifs $Z \subseteq R_p$ qui lui contient l'image de l'ensemble admissible X dans R_p. toute solution réalisable x est associé un vecteur critère

$z(x) = (z_1(x), ..., z_p(x))$ qui correspond à son image dans Z, telle z que z_k est une fonction objectif pour tout k dans $\{1, ..., p\}$. L'ensemble des points correspondant aux images des solutions admissibles dans Z est noté $Z = \{z(x) : x \in X\}$. En raison de la nature conflictuelle des objectifs, il n'y a généralement pas une solution réalisable qui minimise simultanément tous les objectifs. L'optimalité dans un contexte multi-objectif est basée sur la notion de dominance et d'efficacité au sens de Pareto dont nous rappelons le formalisme dans les définitions 1 et 2.**(voir [7])**

Définition 1.6.1 *: Soient $z, z\prime \in R_p$ deux vecteurs. z domine largement $z\prime$, noté $z \leqq z\prime$, si $z_k \leq z\prime_k$ pour tout k dans $\{1, ..., p\}$. z domine $z\prime$, noté $z \leq z\prime$, si $z_k \leq z\prime_k$ pour tout k dans $\{1, ..., p\}$ et z $z\prime$. z domine strictement $z\prime$, noté $z < z$', si $zk < z\prime k$ pour tout k dans $\{1, ..., p\}$.* ***(voir [7])***

Définition 1.6.2 *: Un point z dans Z est (faiblement) non-dominé s'il n'existe pas un autre point z′ dans Z tel que z′domine (strictement) z. Une solution x dont l'image z(x)dans Zest (faiblement) non-dominé est dite (faiblement) efficace. On notera ZN et ZWN les ensembles de points non-dominés et faiblement non-dominés. Il est facile de voir que ZN ⊂ ZWN. Les ensembles des solutions efficaces et faiblement efficaces seront notés respectivement XEet XWE.* ***(voir [7])***

1.7 Les problèmes minimum et maximum

♦ **Le minimum:**

Soit f fonction définie sur un intervalle I , a est un réel de l'intervalle I.

Dire que f admet un minimum sur I en a signifie que pour tout réel x de l'intervalle I, $f(x)$ est supérieur ou égale à $f(a)$.Ce minimum est $f(a.)$.

♦ **Le maximum:**

Soit f fonction définie sur un intervalle I ,b est un réel de l'intervalle I.

Dire que f admet un maximun sur I en b signifie que pour tout réel x de l'intevalle I, $f(x)$ est inférieur ou égal à $f(b)$; $(f(x) \preceq f(b))$. Ce maximum est $f(b)$.

1.7.1 Minimum global et minimum local

♦ **Minimum global**

On a la fonction $f : \Omega \subseteq \mathbb{R}^n \longrightarrow \mathbb{R}$, tel que $\Omega \neq \varnothing$. $x^* \in \Omega$. Pour $x^* \in \Omega$ optimum global si et seulement si :

$$\forall x \in \Omega : f(x^*) \leq f(x)$$

Tel que

- x : l'optimum global ;
- f: : la fonction objective ;
- Ω: La région faisable ($\Omega \in s$);
- s : l'espace de recherche global.

Le minimum global est illustré par le point M3 dans la Figure (1.1). **(voir [10])**

♦ **Minimum local**

Un point x est un minimum local de la fonction f si et seulement si :

$$f(x^*) \leq f(x), \forall x \in v(x^*) \text{ et } x^* \neq x$$

D'où $v(x)$ définit un voisinage de x .

Deux minimums locaux sont illustrés dans la figure (1.1) sont les points M1 et M2. **(voir [10])**

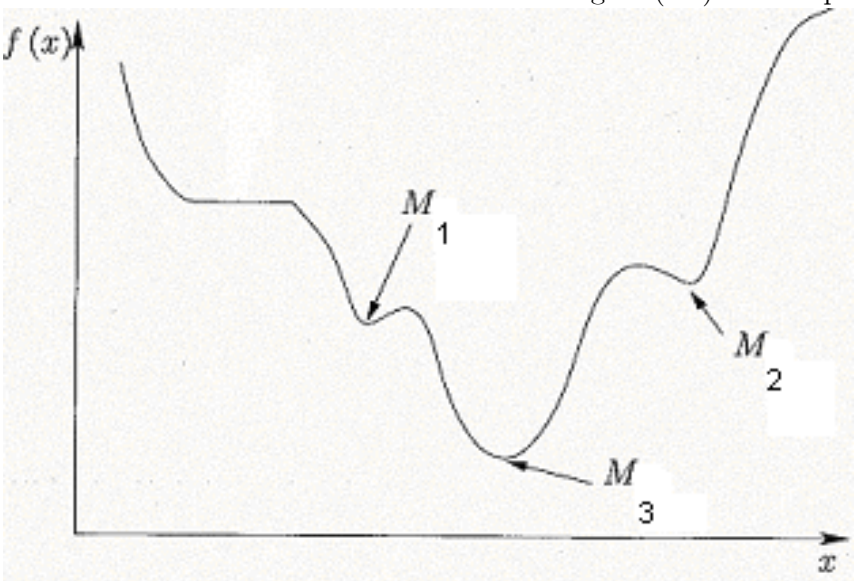


Figure 1.1:Illustration des diffiérents d'un fonction objective.

1.8 La fonction objectif

La fonction objectif f (ou la fonction de coût) formule l'objectif à atteindre. Elle associe à chaque point de l'espace de recherche une valeur qui décrit la qualité de la solution,

$f : s \longrightarrow \mathbb{R}$. Le but du problème d'optimisation est alors de minimiser ou de maximiser cette fonction. **(voir [12])**

1.9 Généralités sur de complexité

1.9.1 Complexité d'un problème

La complexité d'un algorithme c'est le temps et l'espace mémoire nécessaires pour son exécution. Elle est calculée en fonction du nombre N des données appelées aussi taille du problème. Dans

les cas le plus défavorable, la théorie de complexité permet de majorer le nombre d'optimisations nécessaires par une fonction de la taille N du problème posé. La complexité d'un algorithme et F noté (FN), il existe une constante C est un entier A tels que le nombre d'instructions élémentaires $I(N)$ vérifie $I(N) \leq CF(N)$ pour tout $N \geq A$.

Si $F(N)$ est un polynôme alors l'algorithme est dit polynomial. Bien que la théorie de la complexité se concentre sur des problèmes de décision, elle peut être étendue aux problèmes d'optimisation. Elle classe les problèmes selon leurs complexités en deux classes principales:
la classe **P**(polynomial time) et la classe **NP**(non déterministe polynomial time). En outre, elle partage les problèmes de la classe **NP** en deux sous classes : **NP**-Complet et **NP**-Difficile. **(voir [8])**

Classification des problèmes d'optimisations

- **Le problème de classe P:** L'appartenance d'un problème quelconque à cette classe nécessite qu'il soit un problème de décision et ayant au moins un algorithme polynomial comme méthode de résolution, il est nommé facile ou de classe **P**. La vérification de l'existence d'un tel algorithme est obligatoire pour montrer l'appartenance du problème à cette classe.

- **Le problème de classe NP:** Le problème de cette classe ne peut pas être résolu dans un temps polynomiale, il est dit **NP** difficile. Les méthodes de résolution utilisées sont heuristiques qui permettent de trouver une solution optimale mais non démontrables. Les problèmes de classe **P** peuvent être résolus par les algorithmes ordinaires ou exacts qui sont incluse dans la famille des heuristiques, par la suite la classe **P** est inclus dans la classe **NP**.

- **Le problème de classe NP-complet:** Un problème de décision **P** est dit **NP**-complet s'il appartient à la classe **NP** et si pour tout problème $\mathbf{P}'$de **NP**, on a les propriétés suivantes:

1. Il existe une application polynomiale qui transforme toute instance $\mathbf{I}'$ de $\mathbf{P}'$en une instance **I** de **A**.

2. $\mathbf{P}'$ admet une réponse $\ll$ *oui* $\gg$ pour l'instance $\mathbf{I}'$, si et seulement si **P** admet une réponse oui pour l'instance **I**.

Une propriété générale des problèmes **NP** complet : Si un seul problème **NP**-complet est polynomial, alors tous les problèmes **NP**-complet sont polynomiaux. Si le problème est **NP**-difficile, il n'existe pas d'algorithme polynomial le résolvant.

- **Le problème de classe NP-Difficiles** englobe les problèmes de décision et les problèmes d'optimisation. Les problème **NP**-Difficiles sont aussi difficiles que les problèmes **NP** Complet. Si un problème de décision associé à un problème d'optimisation **P** est **NP**-Complet alors **P** est un **NP**-Difficile. Par conséquent, afin de prouver qu'un problème d'optimisation est **NP**-Difficile, Il suffit de montrer que le problème de décision associé à **P** est **NP**-Complet. Il est noté que jusqu'à maintenant, aucun algorithme polynomial n'est connu pour résoudre ce type de problèmes (i.e.**NP**-Difficiles). **(voir [8])**

La relation entre les problèmes P et NP

La question « **P = NP ?** » est une des questions les plus importantes qui n'ont pas encore été résolues en informatique théorique. En fait, la réponse à cette question a construit un champ de recherche ouvert. La réponse à la question « **P = NP ?** » par $\ll$ *oui* $\gg$, revient à montrer que tous les problèmes de la classe **NP** sont dans la classe **P**. Autrement dit, la réponse à cette question revient à répondre à la question : Peut-on trouver en temps polynomial ce qu'on peut prouver en temps polynomial ? Cependant, aucune démonstration n'a été faite pour prouver que **NP**$\subseteq$ **P** ,ni que **NP** $\nsubseteq$ **P**. La relation évidente c'est que **P** $\subseteq$ **NP**. En effet, un problème qui peut être résolu en un temps polynomial par un algorithme déterministe, pourra aussi être résolu par un algorithme non déterministe. Ce qui est admis et connu jusqu'à maintenant, c'est que
P $\neq$ **NP**. Néanmoins, aucune preuve n'a encore été prouvée jusqu'à ce jour. Si un jour on arrivera à trouver un algorithme polynomial permettant la résolution dans problème **NP**-Complet, on pourra résoudre tous les problèmes **NP**-Complets en temps polynomial. Cook a prouvé dans que tous les problèmes de la classe **NP** sont réductibles au problème de la satisfiabilité d'une formule logique, cela veut dire que si quelqu'un trouvera un algorithme polynomial pour le problème de SAT, alors la question « **P = NP ?** » sera résolu! Selon la figure **(2.1)**, les problèmes **NP** faciles à résoudre peuvent construire des problèmes de la classe **P**. Les autres problèmes, i.e. les plus difficiles à résoudre, peuvent être partagés en deux autres groupes: le groupe des problèmes **NP**-Complets et le groupe des problèmes dont le statut est indéterminé car ces problèmes n'ont pas été prouvés comme appartenant à la classe **P** ni à la classe **NP**-Complet, comme le cas du problème d'isomorphisme de deux graphes. En effet, il semblait que ce problème appartient à la classe **P**. Néanmoins, jusqu'à ce jour aucune preuve n'a été montrée. **(voir [1])**

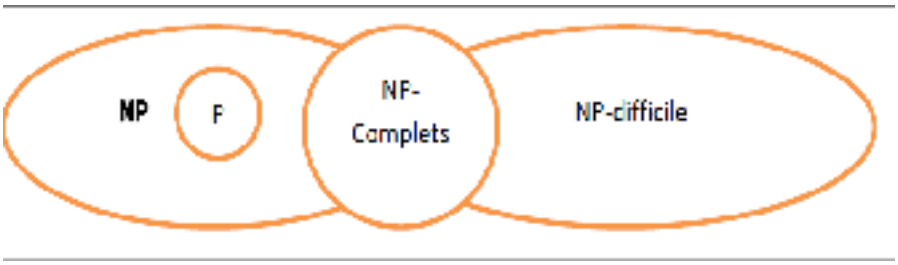


Figure 2.1:la relation entre les différentes classes du problème

1.10 Exemple sur le problème d'optimisation combinatoire

1.10.1 Le problème du voyageur de commerce (PVC):

Le "problème du voyageur de commerce", ou **pvc** (en anglais Traveling Salesman Problem) est un problème il s'agit de trouver le chemin le plus court reliant **n** villes données, chaque ville ne devant être visitée qu'une et une seule fois, et revenir à la ville de départ. La difficulté du problème vient de l'explosion combinatoire du nombre de chemins à explorer lorsque l'on accroît le nombre de villes à visiter. Plus formellement, ce problème peut être modélisé comme suit : Etant donné un graphe non orienté complet $G = (S, A)$, et une fonction de distance $d : A \rightarrow IR^+$, on cherche à déterminer un cycle Hamiltonien 2 de distance totale minimale. Pour ce problème, trouver s'il existe un chemin Hamiltonien est un problème NP-complet, la recherche du plus court chemin Hamiltonien est un problème **NP**-difficile. Le problème **PVC** est alors modélisé par minimiser

$$\sum_{i=1}^{n-1}(D\left(V_i, V_{i+1}\right)) + D(V_n, V_1)$$

ou $D\left(V_i, V_{i+1}\right)$ est la distance entre les deux villes V_i et V_{i+1}. **(voir [8])**

1.11 Notions de voisinage

Les méthodes de voisinage sont fondées sur la notion de voisinage. Nous allons donc introduire d'abord cette notion fondamentale, ainsi que d'autres notions qui lui sont associées. La définition de la structure de voisinage est une étape clef, dans la mise au point d'un algorithme de recherche locale, puisqu'elle permet de définir l'ensemble des solutions qu'il est possible d'atteindre, à partir d'une solution donnée, par une série de transformations.

En particulier, la définition du voisinage permet d'identifier toutes les paires de solutions voisines. **(voir [12])**

1.11.1 Définition du voisinage

Le voisinage de s est un sous-ensemble de configurations de S, directement atteignables à partir d'une transformation donnée de s. Il est noté $V(s)$ et une solution $s^{'} \in V(s)$ est dite voisine de s.

Cette notion de voisinage structure 1'espace de recherche dans le sens ou elle permet de définir des sous-ensembles de solutions. En particulier, à partir d'une solution donnée, on peut établir plusieurs structures de voisinage selon la transformation que 1'on s'autorise, celle-ci étant définie comme une application $V(x) : S \rightharpoonup P(S)$ Chaque structure de voisinage fournit un voisinage, ou un ensemble de solutions, précis via la transformation définie.
Pour caractériser une structure de voisinage dans le contexte de l'optimisation, elle peut être définie en fonction de différentes propriétés. Il en existe trois principales.

1. La complexité spatiale qui détermine la relation entre la taille du voisinage d'une solution et la taille de 1'espace de recherche du problème.

2. La complexité temporelle qui correspond au temps nécessaire pour 1'évaluation des solutions voisines en fonction de la taille du voisinage.

3. La portée qui permet de mesurer le degré de transformation entre la solution ourante et les solutions voisines. Trois degrés de portée sont proposés dont les qualifications générales sont assez imprécises mais qu'il convient d'apprécier pour chaque problème. Le premier degré de portée est la portée locale qui est caractérisée par l'introduction de peu de modifications dans la solution courante. Le second degré est nommé régional car une plus grande quantité de modifications est introduite dans les solutions voisines par rapport à la solution courante. Enfin, les structures de voisinage de portée globale transforment beaucoup les caractéristiques de la solution initiale.

La portée permet de différencier deux structures de voisinage en fonction de 1'é1oignement de 1'ensemble des solutions accessibles par rapport à la solution initial. **(voir [12])**

1.11.2 Portée d'une structure de voisinage

La portée d'une structure de voisinage $V(s)$ est la distance d maximale des solutions $s^{'} \in V(s)$ par rapport à la solution s. Deux structures de voisinage V_1 et V_2 sont de portées identiques si et seulement si :

$$d(s, V_1^{'}(s)) = d(s, V_1(s)), \forall s$$

Avec,

$$d(s, V_k(s)) = \max_{s' \in V_k(s)} (d(s, s^{'}))$$

et $d(s; s^{'})$ la distance entre les solutions s et $s^{'}$. Elles sont de portées différentes dans le cas contraire. **(voir [12])**

1.11.3 Optimum local et optimum global

♦ **Optimum local:**

Une solution s E S est un optimum local si et seulement s'il n'existe pas de solution $s^{'} \in V(s)$ dont l'évaluation est de meilleure qualité que s, soit :

$$\forall s^{'} \in V(s) = \begin{cases} f(s) \leq f(s^{'}) \text{ dans le cas d'un problème de minimisation} \\ f(s) \geq f(s^{'}) \text{ dans cas d'un problème de maximisation} \end{cases}$$

avec $V(s)$ 1'ensemble des solutions voisines de s. La définition d'un optimum local est liée à la structure de voisinage. En d'autres termes, un optimum local pour une structure

de voisinage donnée ne l'est pas forcement pour une autre structure de voisinage. **(voir [12])**

♦ **Optimum global:**

Une solution s qui vérifie la propriété précédente pour toutes les structures de voisinage du problème est appelée optimum global. Beaucoup d'algorithmes de recherche spécifiques à un problème (heuristiques) ou généraux (métaheuristiques) utilisent une seule structure de voisinage. L'intérêt est la simplicité de mise en oeuvre et la meilleure compréhension de ce que fait l'algorithme.

Cependant, l'utilisation de plusieurs structures de voisinage présente un intérêt certain : une solution s reconnue comme un optimum local pour une structure de voisinage donnée peut ne pas être un optimum local pour une autre structure de voisinage. Il peut donc être intéressant d'utiliser des algorithmes combinant plusieurs structures de voisinage pour la recherche dans les problèmes disposant de nombreux minimums locaux. **(voir [12])**

Chapitre 2

Les méthodes de résolution de problèmes d'optimisation

2.1 Introduction

Deux catégories de méthodes se présentent à nous pour la résolution des problèmes d'optimisation combinatoire : les méthodes dites exactes et les méthodes approchées. Malgré leur efficacité, les premières coûtent cher en temps et espace mémoire des ordinateurs. Alors, que si les secondes sont plus économiques, elles sont moins efficaces que les premières. Les métaheuristiques sont un moyen médian pour équilibrer entre le temps nécessaire à l'exécution du programme concerné et la qualité de la solution obtenue. Nous rappelons que la plupart des métaheuristiques sont d'inspiration biologique au point qu'elles sont appelées « algorithme inspirés de la nature ». Ce chapitre se focalise sur la présentation de quelques méthodes utilisées pour la résolution des problèmes d'optimisation combinatoire.

La résolution de différentes sortes de problèmes rencontrés dans notre vie quotidienne a poussé les chercheurs à proposer des méthodes de résolution et à réaliser de grands efforts pour améliorer leurs performances en termes de temps de calcul nécessaire et/ou de la qualité de la solution proposée.

Au fil des années, de nombreuses méthodes de résolution de problèmes de diffiérentes complexités ont été proposées. Ainsi, une grande variété et des différentes remarquables au niveau du principe, de la stratégie et des performances ont été discernées. Cette variété et ces différentes ont permis de regrouper les différentes méthodes de résolution de différents problèmes en deux classes principales: la classe de méthodes exactes, la classe des méthodes approchées.

2.2 Les méthodes exactes

Dans Les méthodes exactes toutes les solutions de l'espace de recherche sont énumérées implicitement en utilisant des mécanismes qui détectent des échecs (calcul de bornes). Grâce à Ces méthodes on peut trouver des solutions optimales. Mais ces méthodes s'avèrent, malgré les progrès réalisés, plutôt inefficaces à mesure que la taille du problème devient importante. Dans cette classe des méthodes exactes.

Les méthodes exactes ont permis de trouver des solutions optimales pour des problèmes de taille raisonnable et rencontrent généralement des difficultés face aux applications de taille importante. Dans cette classe des méthodes exactes, on peut trouver les algorithmes classiques suivants : La programmation dynamique, La programmation linéaire, Les méthodes de recherche arborescente (Branch & bound). **(voir [5])**

2.2.1 La méthode de Branch et Bound (B&B)

Pour plusieurs problèmes, en particulier les problèmes d'optimisation, l'ensemble de leurs solutions est fini (en tous les cas, il est dénombrable). Il est donc possible, en principe, d'énumérer toutes ces solutions, et ensuite de prendre celle qui nous arrange. L'inconvénient majeur de cette approche est le nombre prohibitif du nombre de solutions : il n'est guère évident d'effectuer cette énumération. La méthode de branch and bound (procédure par évaluation et séparation progressive) consiste à énumérer ces solutions d'une manière intelligente en ce sens que, en utilisant certaines propriétés du problème en question, cette technique arrive à éliminer des solutions partielles qui ne mènent pas à la solution que l'on recherche. De ce fait, on arrive souvent à obtenir la solution recherchée en des temps raisonnables. Bien entendu, dans le pire cas, on retombe toujours sur l'élimination explicite de toutes les solutions du problème. Pour ce faire, cette méthode se dote d'une fonction qui permet de mettre une borne sur certaines solutions pour soit les exclure soit les maintenir comme des solutions potentielles. Bien entendu, la performance d'une méthode de branch and bound dépend, entre autres, de la qualité de cette fonction (de sa capacité d'exclure des solutions partielles tôt).

La méthode de Branch and Bound sont basées sur l'idée de l'énumération des solutions réalisables. L'avantage certain de ces méthodes est qu'elles sont exploitables pour tout problème d'optimisation combinatoire puisqu'elles tirent leur validité du seul fait que l'ensemble admissible d'un problème est fini. Bien évidemment, le principe de ces méthodes n'est pas d'énumérer toutes les solutions réalisables mais de restreindre progressivement l'ensemble des solutions réalisables,

et ce en alternant une étape dite de séparation et une étape d'évaluation jusqu'à obtention de la preuve de l'optimalité d'une solution. C'est la raison pour laquelle elles sont aussi appelées méthodes d'énumération implicite.

La séparation consiste en la division de l'ensemble des solutions réalisables en plusieurs sous-ensembles et l'évaluation repose sur la résolution des sous-problèmes engendrés par la séparation et à la suppression des sous-ensembles qui ne contiennent pas la solution optimale. **(voir [11])**

L'algorithme général

Par convenance, on représente l'exécution de la méthode de branch-and-bound à travers une arborescence. La racine de cette arborescence représente l'ensemble de toutes les solutions du problème considéré. Dans ce qui suit, nous résumons la méthode de branch-and-bound sur des problèmes de minimisation.

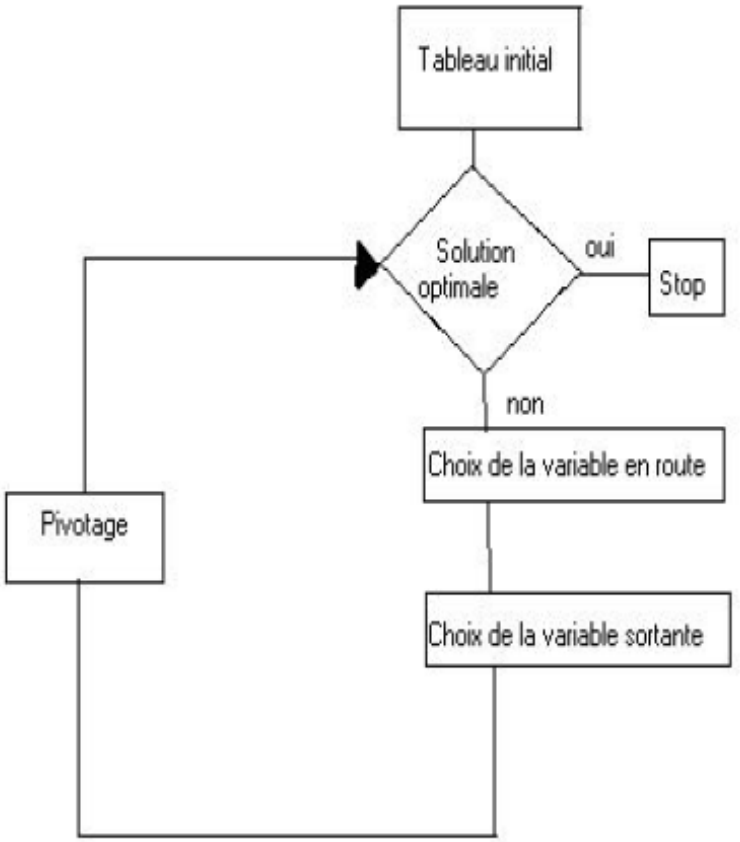
Pour appliquer la méthode de branch-and-bound, nous devons être en possession :

1. d'un moyen de calcul d'une borne inférieure d'une une solution partielle
2. d'une stratégie de subdiviser l'espace de recherche pour créer des espaces de recherche de plus en plus petits.
3. d'un moyen de calcul d'une borne supérieure pour au moins une solution.

La méthode commence par considérer le problème de départ avec son ensemble de solutions, appelé la racine. Des procédures de bornes inférieures et supérieures sont appliquées à la racine. Si ces deux bornes sont égales, alors une solution optimale est trouvée, et on arrête là. Sinon, l'ensemble des solutions est divisé en deux ou plusieurs sous-problèmes, devenant ainsi des enfants de la racine. La méthode est ensuite appliquée récursivement à ces sous-problèmes, engendrant ainsi une arborescence. Si une solution optimale est trouvée pour une sous-problème, elle est réalisable, mais pas nécessairement optimale, pour le problème départ. Comme elle est réalisable, elle peut être utilisée pour éliminer toute sa descendance : si la borne inférieure d'un nœud dépasse la valeur d'une solution déjà connue, alors on peut affirmer que la solution optimale globale ne peut être contenue dans le sous-ensemble de solution représenté par ce nœud. La recherche continue jusqu'à ce que tous les nœud sont soit explorés ou éliminés. **(voir [11])**

2.2.2 Méthode de simplexe

ik



3.png

Figure 2-1:plan explique méthode de simplexe

Introduction

Un programme linéaire **(PL)** mis sous la forme particulière où toutes les contraintes sont des équations et toutes les variables sont non négatives est dit sous forme standard. Il est noté **(PL=)**.

Un programme linéaire **(PL)** mis sous la forme particulière où toutes les contraintes sont des équations et toutes les variables sont non négatives est dit sous forme standard. Il est noté **(PL=). (voir [6])**

Variables d'écart et d'excédent

Avant que l'algorithme du simplexe puisse être utilisé pour résoudre un programme linéaire, ce programme linéaire doit être converti en un programme équivalent où toutes les contraintes technologiques sont des équations et toutes les variables sont non négatives.

a) Contraintes de type ($\leq$) : Pour chaque contrainte i de ce type, on rajoute une variable d'**écart** e_i, tel que e_i est une variable positive ou nulle.

Exemple 2.2.1 *:*$3x_1 + 2x_2 \leq 2$ *se transforme en* $3x_1 + 2x_2 + e_1 = 2$, $e_1 \geq 0$

b) Contraintes de type ($\geq$) : Pour chaque contrainte i de ce type, on retranche une variable d'**excédent** e_i, tel que e_i est une variable positive ou nulle.

Exemple 2.2.2 *:* $3x_1 + 2x_2 \geq 2$ *se transforme en* $3x_1 + 2x_2 - e_2 = 2$, $e_2 \geq 0$

Un programme linéaire qui contient des contraintes (technologiques) de type $\leq$ est noté **(PL)**. Un programme linéaire qui contient des contraintes (technologiques) de type $(\leq, \geq, =)$ est noté **(PG)**. Un programme linéaire **(PL)** resp **(PG)** converti tel que toutes les contraintes technologiques sont des équations et toutes les variables sont non négatives est noté **(PL=)** resp **(PG=)**. **(voir [6])**

Variables de base et variables hors base

Considérons un système d'équations à n variables et m équations où $n \geq m$. Une solution de base pour ce système est obtenue de la manière suivante :

a) On pose $n - m$ variables égales à 0. Ces variables sont appelées variables hors base **(V.H.B.)**.

b) On résout le système pour les m variables restantes. Ces variables sont appelées les variables de base **(V. B)**.

c) Le vecteur de variables obtenu est appelé solution de base (il contient les variables de base et les variables hors base).

Une solution de base est **admissible** si toutes les variables de la solution de base sont ≥ 0.

Il est vraiment important d'avoir le même nombre de variables que d'équations. **(voir [6])**

Solutions admissibles

Toute solution de base de **(PL=)** pour laquelle toutes les variables sont non négatives, est appelée solution de base admissible. Cette solution de base admissible correspond à un point extrême. **(voir [6])**

La méthode du simplexe est une méthode exacte et itérative . En effet, d'un point de vue géométrique, étant donné qu'une solution d'un programme linéaire se trouve toujours sur un point extrême du polyèdre des contraintes, le simplexe consiste à se déplacer d'un point extrême à l'autre, le long des arêtes du polyèdre, jusqu'à trouver le point associé à la solution optimale.

Algébriquement, le simplexe s'interprète comme la détermination d'une suite de bases adjacentes réalisables telles que les valeurs associées de la fonction objectif soient croissantes. Considérons le programme linéaire **(P1)** et soit **B** une base réalisable (non dégénérée) issue de la matrice des contraintes. On décompose la fonction objectif :

$$z = c.x = c_N.x_N$$

En effectuant le changement de variable :

$$x_B = B^{-1}.b - B^{-1}.N.x_N$$

on peut exprimer z en fonction de x_N :

$$z = (c_N - c_B.B^{-1}.N)x_N + c_B.B^{-1}.b$$

Pour la solution de base associée à B, on a $z = c_B.B^{-1}.b$ $(car\ x_N = 0)$.
On pose $\pi = (\pi_1, ..., \pi_m)$, défini par :

$$\pi = c_B.B^{-1}$$

le vecteur des multiplicateurs du simplexe et :

$$\overline{c_N} = c_N - \pi.N$$

le vecteur des profits marginaux (ou réduits) des variables hors base. Les profits marginaux donnent le gain obtenu en augmentant de 1 la variable associée. S'il existe une variable hors base x_r dont le profit marginal $\overline{c_N^r}$. N est strictement positif, alors il suffit d'augmenter sa valeur dans la solution de base pour augmenter la valeur de z. La base B courante n'est donc pas optimale ; on augmente alors la variable hors base x_r, actuellement à 0, jusqu'à annuler une variable de base. La suite de cette opération de pivotage, la variable hors base augmentée, dite entrante (dans la base), remplace la variable de base annulée, dite sortante : le problème est exprimé dans une nouvelle base $B^{'}$, adjacente à B. C'est le principe d'itération de la méthode du simplexe. Géométriquement, cette opération revient à passer sur un sommet adjacent du polyèdre.

Si on a, au contraire, $\overline{c_N} \leq 0$, alors toute augmentation de variable diminuerait la valeur de z; la valeur optimale (le maximum) de z est donc obtenue pour la solution de base $(B^{-1}.b|0)$.

Notons que le vecteur est alors la solution optimale du problème dual (D_1) de (P_1)
$(u^* : \pi = (\pi_1, ..., \pi_m), (\pi_1, ..., \pi_m)$ sont les valeurs duales de $(P1))$.

Sous l'hypothèse de non-dégénérescence des bases, la convergence est garantie en un nombre fini d'itérations, du fait du nombre fini de sommets d'un polyèdre. Comme nous l'avons déjà dit, la complexité du simplexe est théoriquement exponentielle. C'est ce qui en fait une méthode très efficace et très utilisée pour la résolution de problèmes réels.

Le simplexe permet également d'indiquer l'existence d'une infinité de solutions optimales ainsi que le cas où il n'y a pas de solution optimale à valeur finie. Pour plus de détails sur la méthode du simplexe, notamment sur la prise en compte de ces cas spéciaux et sur l'étude de sa complexité. **(voir [4])**

2.3 Les méthodes approchées

2.3.1 La recherche Tabou

La recherche tabou **(RT)** est une métaheuristique à base d'une solution unique. Elle a été proposée en 1986 par Glover [Glover, 1986]. **RT** est une méthode de recherche locale avancée, elle fait appel à un ensemble de règles et de mécanismes généraux pour guider la recherche de manière intelligente [Glover et Laguna, 1997]. L'optimisation de la solution avec la recherche tabou se base sur deux astuces: l'utilisation de la notion du voisinage et l'utilisation d'une mémoire permettant le guidage intelligent du processus de la recherche. En parcourant le voisinage de la solution courante s, la recherche tabou ne s'arrête pas au premier optimum local rencontré. Elle examine un échantillonnage de solution du voisinage de s et retient toujours la meilleure solution voisine $s\prime$, même si celle-ci est de piètre qualité que la solution courante s, afin d'échapper de la vallée de l'optimum local et donner au processus de la recherche d'autres possibilités d'exploration de l'espace de recherche afin de rencontrer l'optimum global. En fait, les solutions de mauvaise qualité peuvent avoir de bons voisinages et donc guider la recherche vers de meilleures solutions. Cependant, cette stratégie peut créer un phénomène de cyclage (i.e. on peut revisiter des solutions déjà parcourues plusieurs fois). Afin de pallier à ce problème, la recherche tabou propose l'utilisation d'une mémoire permettant le stockage des dernières solutions rencontrées pour ne pas les visiter dans les prochaines itérations et tomber dans le problème du cyclage répétitif. Cette mémoire est appelée « la liste

tabou », d'où le nom de la métaheuristique tabou. La taille de la liste tabou est limitée, ce qui empêche l'enregistrement de toutes les solutions rencontrées. C'est la raison pour laquelle la liste tabou procède comme une pile **FIFO**, où la plus ancienne solution sera écartée pour laisser place à la dernière solution rencontrée. Le but de faciliter la gestion de la liste tabou en termes de temps de calcul ou d'espace mémoire nécessaires à pousser les chercheurs à proposer de ne conserver dans la liste que des attributs (i.e. caractéristiques) des solutions au lieu des solutions complètes. Particulièrement, dans le cas où le nombre de variables est très élevé. Toutefois, l'utilisation de la liste tabou peut mener à un blocage dans certains cas. En fait, les nouvelles solutions qui possèdent des attributs des solutions tabou, seront considérées tabou aussi même si elles ne sont pas encore été visitées. Pour remédier à ce problème, on a défini une technique appelée « critère d'aspiration» permettant de sortir du blocage causé par la ressemblance des attributs des solutions tabou. Le critère d'aspiration permet d'omettre le statut tabou sur une solution lorsque certaines circonstances sont respectées. Un critère d'aspiration très classique consiste à autoriser une solution tabou si celle-ci est l'une des meilleures solutions rencontrées depuis le démarrage de la recherche. **(voir [1])**

Algorithme 2.3.1 *: La recherche tabou*

Début
Construire une solution initiale s ;
Calculer la fitness $f(s)$ de s ;
Initialiser une liste tabou vide ;
$s_{best} = s$;
Tant que le critère d'arrêt n'est pas vérifié **faire**
Trouver la meilleure solution $s\prime$ dans le voisinage de s qui
ne soit pas tabou ou qui vérifie le critère d'aspiration ;
Calculer $f(s\prime)$;
Si fitness de $(s\prime)$ est meilleure que fitness de (s_{best}) **alors**
$s_{best} = s\prime$;
Fin Si
Mettre à jour la liste tabou ;
$s = s\prime$;
Fin Tant que
Retourner s_{best} ;

Fin

La recherche tabou a été largement utilisée pour la résolution de problèmes d'optimisation difficiles comme: le problème du voyageur de commerce, les problèmes du sac à dos, les problèmes de routage, d'ordonnancement, de coloration de graphes, d'exploitation géologique, etc. C'est une méthode facile à mettre en œuvre, rapide, donne souvent de bons résultats et permet de se sauver du premier optimum local rencontré contrairement à la méthode de la recherche locale simple. En revanche, la liste tabou demande de ressources importantes si elle est de grande taille. En fait, elle demande une gestion soigneuse de sa taille et de ce qu'elle doit contenir comme informations sur les solutions trouvées, de manière à ne pas être très exigeante en temps de parcours et d'espace mémoire requis et aussi pour éviter les confusions causées par la ressemblance des attributs des solutions qui bloquent la recherche.

Il est à noter qu'il existe plusieurs variantes de la recherche tabou. Ces variantes dépendent essentiellement du choix du voisinage et de la manière de gérer la liste tabou. L'algorithme 2.3.1 représente un schéma général de l'algorithme de la recherche tabou. **(voir [1])**

2.3.2 La méthode recuit simule

L'algorithme de recuit simule (Simulated Annealing SA) est inspire du principe de la thermodynamique et a été proposé par Kirkpatrick , Gelatt et Vecchiet en 1983. L'idée de cet algorithme consiste à reproduire le phénomène de recuit obtenu lors de refroidissement d'un matériau solide préalablement chauffe : la structure du matériau obtenu varie en fonction de la vitesse de refroidissement. En partant d'une haute température à laquelle le matériau solide est devenu liquide. Avec plus de liberté aux atomes qui le composent, la phase de refroidissement conduit la matière liquide à retrouver sa forme solide par une diminution progressive de la température. Si le refroidissement est trop brusque, les atomes peuvent, s'éloigner de leur état d'équilibre et causer une imperfection au morceau de métal. Le refroidissement du métal est donc lent et régulier pour permettre aux atomes de se stabiliser peu à peu dans une position d'énergie minimale. L'algorithme de recuit simule reproduit les variations d'énergie observées lors de processus d'un recuit. L'énergie désigne la valeur de la fonction coût de la solution courante et l'état correspond à une solution du problème traité. Il consiste donc en une recherche aléatoire de l'espace d'état à favoriser les descentes, mais sans interdire tout à fait les remontrées. Cependant, une augmentation du niveau d'énergie permet de sortir des minimum locaux. L'algorithme montre qu'on tend très rapidement vers un minimum local proche de la meilleure solution.

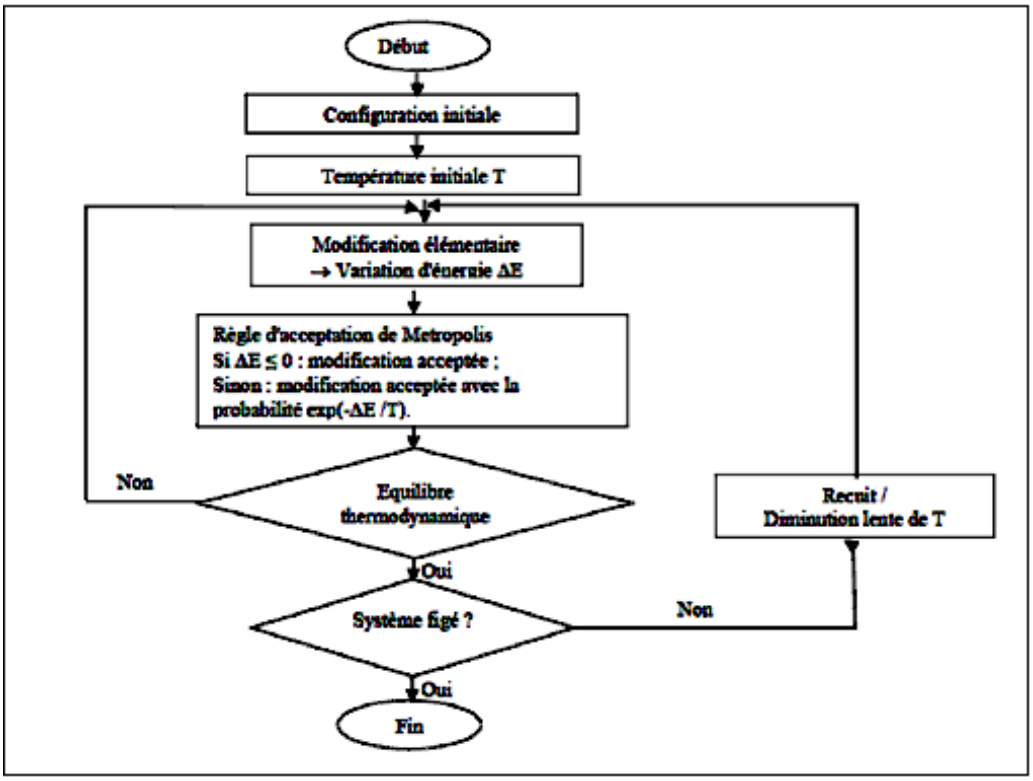


Figure 2.2 : Organigramme général du recuit simulé

Initialement on donne le système une très haute température puis on le refroidit petit à petit Le refroidissement du système doit se faire très lentement pour avoir l'assurance d'atteindre un état d'équilibre à chaque température T.

♦ Le voisinage $N(s)$ d'une solution s. s'appartient à l'ensemble des états atteignables depuis l'état courant en faisant subir des déplacements élémentaires aux atomes du

système Physique.

♦ A chaque itération, une seule solution voisine s'est générée et elle est acceptée si elle est meilleure que la solution courante s dans le cas contraire on a les cas :

· Si T grande, $exp(-delta\ \mathrm{E}/T)$ est de l'ordre de 1, on garde toujours le mouvement, même il est mauvais.

· Si T très petit, $exp(-delta\ \mathrm{E}/T)$ est de l'ordre de 0, donc les mouvements qui augmentent l'énergie (la différence) sont disqualifiés.

Méthode : on tire au hasard dans l'intervalle $[0, 1[$, si le nombre est $< exp(-delta\ \mathrm{E}/T)$, on garde sinon on jette.

♦ La meilleure solution trouvée est mémorisée dans la variable s^*.

Les avantages:

- Traite les fonctions de coût avec les degrés tout à fait arbitraires de non linéarité, la discontinuité, et l'imprévisibilité.
- Processus assez arbitraire sur les conditions limites et les contraintes imposées sur les fonctions de coût.
- Simple à implémenter.
- Garantie Statistique de trouver une solution optimale.

Les inconvénients:

- le non déterminisme.
- Difficulté de choisir les paramètres efficaces par exemple le schéma de refroidissement.
- Le compromis entre la vitesse et l'optimisation. **(voir [5])**

Un schéma général de l'algorithme du recuit simulé (RS) est présenté dans l'algorithme 2.3.2. Le recuit simulé permet d'accepter une solution de piètre qualité que la solution courante afin de diversifier la recherche et échapper au piège de l'optimum local. Le fait d'accepter des solutions de mauvaises qualités peut mener la recherche vers la meilleure solution (l'optimum global) car cette dernière peut faire partie du voisinage d'une mauvaise solution et non pas d'une bonne solution (i.e. la solution courante qui est de meilleure qualité que sa voisine) qui peut représenter un optimum local. Cependant, l'acceptation de solutions de mauvaises qualités peut causer une perte de la meilleure solution rencontrée au cours de la recherche et entraine une convergence vers une solution de mauvaise qualité qu'une autre déjà trouvée. Ce problème peut être facilement résolu en ajoutant une variable permettant la mémorisation de la meilleure solution trouvée.

Algorithme 2.3.2 *Le recuit simulé*

Début

Construire une solution initiale s ;

Calculer la fitness $f(s)$ de s ;

Initialiser une valeur de la température T ;

$s_{best} = s$;

Tant que la condition d'arrêt n'est pas satisfaite **faire**

Générer une solution $s^{'}$ voisine de s;

Calculer $f(s^{'})$;

Calculer $\Delta(f) = f(s^{'}) - f(s)$;

Si $\Delta(f) \geq 0$ **alors** // cas de maximisation

$s_{best} = s^{'}$;

$s = s^{'}$;

Sinon Si $r < \exp \frac{\Delta(f)}{T}$ **alors**

$s = s^{'}$;

Fin Si

Décroitre la température T ;

Fin Tant que

Retourner s_{best};

Fin

L'acceptation d'une solution de mauvaise qualité est établi en fonction de deux facteurs: l'écart de qualité entre la solution courante et sa voisine d'un côté, et la température de l'autre côté. Plus la température est élevée, plus la probabilité d'accepter des solutions de mauvaises qualités est forte. La valeur initiale de la température T décroit au cours de la recherche pour tendre vers le 0. Ce paramètre (i.e. la température) a un effet non négligeable sur la performance de l'algorithme. Il doit être soigneusement ajusté tout au long de la recherche.

En fait, un refroidissement rapide de la valeur de T peut entrainer une convergence prématurée de l'algorithme vers un optimum local de mauvaise qualité. Tandis qu'un refroidissement lent peut mener la recherche vers une solution de bonne qualité. Cependant, le refroidissement trop lent nécessite un temps de calcul élevé. En outre, le choix de la valeur initiale de T joue un rôle primordial dans le processus de recherche. Il dépend de la qualité de la solution initiale. Si cette dernière est choisie aléatoirement, il sera préférable d'affecter une valeur élevée à T pour donner plus de chance d'aboutissement à de bonnes solutions. Si le choix de la valeur initiale de la solution est expérimental ou empirique, la température initiale peut être basse.

Le recuit simulé est un algorithme basé sur la recherche à voisinage (recherche locale). C'est un algorithme simple, facile à implémenter et à adapter à un grand nombre de problèmes : traitement

d'image, sac à dos, voyageur de commerce, ordonnancement, etc. Comparé avec la recherche locale simple, le **RS** permet de se sauver du piège de l'optimum local et d'offrir des solutions de bonne qualité comme il présente une souplesse d'intégration des contraintes liées au problème traité. En revanche, cet algorithme dispose d'un nombre important de paramètres (température initiale, paramètres liés à la fonction d'ajustement de la température... etc.) à ajuster. En outre, le RS est un algorithme lent surtout avec les problèmes de grande taille. **(voir [1])**

Chapitre 3

Comparaison entre deux algorithmes

3.1 Définition d'ordonnancement:

« Ordonnancer, c'est programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution».

Soit un problème d'ordonnancement. Les notations suivantes sont introduites pour définir l'énoncé du problème. Soit T l'ensemble de n tâches, $T = \{T1, T2, ..., Tn\}$, P l'ensemble de m

processeurs (machines) $P = \{P1, P2, ..., Pm\}$ et R l'ensemble des ressources additionnelles $R = \{R1, R2, ..., Rs\}$. On peut définir le problème d'ordonnancement comme suit

« L'ordonnancement, de manière générale, est l'affectation des processeurs de P et (éventuellement) des ressources de R aux tâches de T dans le but de réaliser toutes les tâches en respectant les contraintes». **(voir [13])**

3.2 Les types (classes) d'ordonnancement:

Plusieurs classes d'ordonnancement sont distinguées

- **long terme :** choix concernant la reserve de processus a exécuter.
- **moyen terme :** choix concernant le nombre de processus en mémoire.
- **court terme :** choix concernant le processus ex ecut e par le processeur.
- **L'ordonnancement d'entree sortie :** quelle requête sera exécutée. **(voir [2])**

3.3 Les éléments d'un problème d'ordonnancement:

Un ordonnancement est décrit, par les éléments qui le constituent. Quatre éléments sont retenus pour décrire d'une manière explicite un ordonnancement, ces éléments sont : les tâches, les ressources, les contraintes et les critères à prendre en considération lors du processus d'optimisation. Dans ce qui suit, on donnera une définition détaillée de chacun de ces éléments.

1. Les tâches

Une tâche est un travail élémentaire dont la réalisation nécessite un certain nombre d'unités de temps (sa durée) et d'unités de chaque ressource.

Une tâche (i) est localisée dans le temps par une date de début (t_i) et/ou de fin (C_i), et par une durée d'exécution (P_i): telle que $P_i = t_i - C_i$

Deux autres éléments définissant la tâche sont le r_i; la date de disponibilité de la tâche et le F_i; qui représente la durée de séjour de l'opération i sur la machine avant qu'elle redevienne disponible (flow time en anglais) ;

Si les tâches sont, liées entre elles par des conditions d'origines diverses on dit alors que les tâches sont interdépendantes, dans le cas contraire, elles sont dites indépendantes.

De même, les tâches sont dites interruptibles si elles peuvent être exécutées par morceaux par une ou plusieurs ressources; dans le cas contraire, si on ne peut pas interrompre une tâche une fois celle-ci commencée, on parle alors de tâches non préemptives.

2. Les ressources :

Les ressources sont les moyens ou les outils nécessaires à la réalisation des tâches. On distingue deux grandes familles de ressources ;

- Les ressources renouvelables sont les ressources réutilisables après la fin d'exécution des opérations. Cette famille est constituée des machines, d'hommes et des outils de productions.

souvent des ressources additionnelles sont considérées comme par exemple l'utilisation des robots, ou des moyens de transport, ou des ressources d'un autre genre comme les buffers d'entrée/sortie. Parmi ces ressources on distingue aussi, des ressources disjonctives qui ne peuvent pas exécuter qu'une seule opération à la fois et des ressources cumulatives qui peuvent exécuter plusieurs opérations à la fois c'est le cas par exemple des machines parallèles ; notons qu'une machine fait partie de l'ensemble des ressources renouvelables une machine de capacité 1 est une ressource disponible en un unique exemplaire.

- La deuxième famille est dite consommable et regroupe toutes les ressources qui deviennent indisponibles après une utilisation, c'est le cas des matières premières de l'argent, de l'énergie etc.

3. Les contraintes

Selon les contraintes expriment les restrictions que peuvent prendre conjointement les variables de décision. Donc les contraintes nous renseignent sur les limites imposées par l'environnement. On distingue plusieurs types de contraintes, qui sont:

♦ **les contraintes de localisation temporelles :** Pour la réussite d'un projet ou d'un plan de production, on doit se soumettre aux impératifs de production (réalisation) traduits par le respect d'un échéancier fixé auparavant. Au niveau global on parlera d'une date de lancement d'un projet et d'une date de livraison. A un autre niveau de détail plus fin, pour une tâche ou une opération. Ce dernier niveau se traduit par la définition des dates suivante:

- date de disponibilité r_i ;
- date d'échéance : d_i;

♦ **les contraintes de précédence :** une contrainte qui lie le début d'une activité à la fin d'un autre est appelée contrainte de succession ou de précédence. Les gammes opératoires sont un exemple de ces contraintes, d'autres contraintes sont imposées dans certain cas telles que les contraintes de synchronisation, de simultanéité, ou de recouvrements etc.

♦ **Les contraintes de ressources :** Les contraintes de ressources représentent le fait que les activités requièrent tout au long de leur exécution une certaine quantité de ressources. Les contraintes de ressources concernent les deux points suivants :

1. l'utilisation de ces ressources, (leur nature, la quantité nécessaire, et les caractéristiques d'utilisation) ;
2. la disponibilité et la quantité de ces ressources.

Ces contraintes nous donnent une information précise sur la nature de l'atelier, et influencent le choix des méthodes d'optimisation ; Il est important de noter qu'en fonction des objectifs de l'étude, ces contraintes peuvent être strictes ou non. Selon, lorsqu'elles sont strictes, elles constituent des obligations à respecter. Lorsqu'elles ne sont pas strictes, on peut ne pas les satisfaites et on les appelle alors des contraintes de préférence.**(voir [13])**

3.4 L'ordonnancement des opérations sur une machine multitâche

Les machines multitâches effectuent plusieurs tâches simultanément. Ces tâches peuvent avoir la même durée ou des durées différentes. Dans le cas où toutes les tâches à exécuter simultanément sont de même durée, le problème est semblable à celui de la machine monotâche. Le cas qui nous intéresse ici est celui où plusieurs des tâches à exécuter ont des durées différentes et l'exécution de chacune de ces tâches nécessite d'effectuer un réglage au préalable que nous appelons le réglage de produit. En plus, le passage d'une famille de produits à une autre nécessite un autre réglage commun à tous les produits (tâches) qui composent la famille. Ce réglage sera appelé réglage de famille. Dans un contexte industriel, nous recevons des commandes où chaque commande a une date limitée de livraison et comprend un ensemble de produits différents à livrer avec des quantités précises. Chaque produit se définit par un ensemble de caractéristiques telles que le modèle, les dimensions et les couleurs. La fabrication d'un produit nécessite l'exécution d'une opération sur la machine multitâche et chaque unité du produit doit subir cette opération. Pour planifier les opérations découlant des commandes reçues, nous devons constituer un ensemble de tâches à exécuter où chaque tâche consiste à effectuer une opération précise pour un lot d'un même produit. Les éléments d'un tel lot peuvent venir d'une même commande ou de plusieurs commandes différentes. Ainsi, les éléments d'un produit appartenant à une commande donnée peuvent se trouver dans une même tâche ou dans plusieurs tâches différentes. Définir ou constituer une tâche consiste donc à préciser le produit concerné, les commandes de provenance et la quantité venant de chacune de ces commandes. Notre problème consiste donc à définir toutes les tâches à effectuer et à déterminer la date de début et de fin d'exécution de chacune de ces tâches de façon à minimiser la durée totale des opérations et, dans la limite du possible, en respectant les dates limitées de livraison des commandes.**(voir [13])**

3.5 Les algorithmes des minimisations et des maximisations

Soit un problème d'optimisation combinatoire de n tache et un machine single.

- $I = 1$, machine unique.
- $j = \{1, 2, 3, ..., n\}$,ensemble des tâches.
- P_j : denée de tâche j.

- W_j : poids de tâche j .
- h : l'indice de l'ordre .

3.5.1 Algorithme de maximisation et exemples

1.Ordonnées les tâches j selon l'ordre.

$\max\left\{\frac{P_j}{W_j}\right\}$et determier h.

2.Affecter h à la machine I

3.Prendre $j = j - \{h\}$.

Exemple 3.5.1 *maximisation au minimisaton*

Tâche	Task N°1	Task N°2	Task N°3	Task N°4	Task N°5
Poids	9.00	5.00	2.00	8.00	7.00
Temps	52.0	16.00	80.00	17.00	34.00
P/T	0.17	0.31	0.02	0.47	0.20
Ordonnancement	Task N°4	Task N°2	Task N°5	Task N°1	Task N°3

Tâche	Task N°1	Task N°2	Task N°3	Task N°4	Task N°5	Task N°6
Poids	8.00	5.00	1.00	4.00	9.00	8.00
Temps	25.00	12.00	98.00	70.00	19.00	35.00
P/T	0.32	0.41	0.01	0.05	0.47	0.22
Ordonnancement	Task N°9	Task N°5	Task N°2	Task N°1	Task N°6	Task N°10

Task N°7	Task N°8	Task N°9	Task N°10
6.00	3.00	8.00	9.00
64.00	78.00	10.00	80.00
0.09	0.03	0.80	0.11
Task N°7	Task N°4	Task N°8	Task N°3

3.5.2 Algorithme de minimisation et exemples

1.Ordonnées les tâches j selon l'ordre.

$\max\left\{\frac{P_j}{W_j}\right\}$et determier h.

2. Affecter h à la machine I

3. Prendre $j = j - \{h\}$.

Exemple 3.5.2 *minimisation au maximisation*

Tâche	TaskN°1	TaskN°2	TaskN°3	TaskN°4	TaskN°5
Poids	9.00	5.00	2.00	8.00	7.00
Temps	52.00	16.00	80.00	17.00	34.00
P/T	0.17	0.31	0.02	0.47	0.20
Ordonnancement	TaskN°3	TaskN°1	TaskN°5	TaskN°2	TaskN°4

Tâche	TaskN°1	TaskN°2	TaskN°3	TaskN°4	TaskN°5	TaskN°6
Poids	8.00	5.00	1.00	4.00	9.00	8.00
Temps	25.00	12.00	98.00	70.00	19.00	35.00
P/T	0.32	0.41	0.01	0.05	0.47	0.22
Ordonnancement	TaskN°3	TaskN°8	TaskN°4	TaskN°7	TaskN°10	TaskN°6

TaskN°7	TaskN°8	TaskN°9	TaskN°10
6.00	3.00	8.00	9.00
64.00	78.00	10.00	80.00
0.09	0.03	0.80	0.11
TaskN°1	TaskN°2	TaskN°5	TaskN°9

Conclusion générale

L'optimisation combinatoire est un domaine concerné par la recherche des valeurs des variables d'un problème compté. Habituellement, tout problème numéroté pour trouver ses valeurs doit être recherché de toutes les manières pour résoudre et choisir le meilleur, mais cette méthode est une méthode coûteuse pour les ordinateurs, donc cette science est venue la résoudre de différentes manières théoriques et non physiques. Exemple: Le problème du voyageur de commerce voix pour le résoudre en triant toutes les solutions possibles. Si le nombre de villes était 100, il faudrait des millions d'années pour le résoudre, il existe donc des moyens dans cette science pour le résoudre plus rapidement en utilisant des techniques algorithmiques et autres.

Dans notre mémoire nous avons présenté la résolution de différentes sortes de problèmes rencontrés dans notre vie quotidienne a poussé les chercheurs à proposée des méthodes de résolution et à réalisé de grands efforts pour améliorer leurs performances en termes de temps de calcul nécessaire et/ou de la qualité de la solution proposée. Au fil des années, de nombreuses méthodes de résolution de problèmes de différentes complexités ont été proposées. Ainsi, une grande variété et des différences remarquables au niveau du principe, de la stratégie et des performances ont été discernées. Cette variété et ces différences ont permis de regrouper les différentes méthodes de résolution des différents problèmes en deux classes principales: la classe de méthodes exactes et la classe de méthodes approchées. L'hybridation des méthodes de ces deux classes a donné naissance à une pseudo classe qui englobe des méthodes dites hybrides. Les méthodes exactes sont connues par le fait qu'elles garantissent l'optimalité de la solution mais elles sont très gourmandes en termes de temps de calcul et de l'espace mémoire nécessaire. C'est la raison pour laquelle, elles sont beaucoup plus utilisées pour la résolution des problèmes faciles. La nécessité de disposer une solution de bonne qualité (semi optimale) avec un coût de recherche (temps de calcul et espace mémoire) raisonnable a excité les chercheurs à proposer un autre type de méthodes de résolution de problèmes, communément connu par les méthodes approchées. Ces dernières constituent une alternative aux méthodes exactes. En fait, elles permettent de fournir des solutions de très bonne qualité en un temps de calcul raisonnable. De nombreuses méthodes approchées ont été proposées. Elles sont plus pratiques pour la résolution des problèmes difficiles où des problèmes dont on cherche des solutions en un bref délai. Ces méthodes sont souvent classées en deux catégories: des méthodes heuristiques et des méthodes métaheuristiques. De plus, nous avons étudié deux algorithmes, puis nous avons comparé leurs résultats et enfin nous avons appliqué des exemples dans chacun des algorithmes. Nous modifions certaines données afin de trouver la différence entre

les résultats des algorithmes pour résoudre les problèmes présentés. Ces résultats obtenus sont contrôlés par des conditions de base telles que le temps, le poids et le nombre des tâches, à partir de là, nous concluons à atteindre la solution optimale en fonction de chaque algorithme et à quelle vitesse il peut trouver le résultat.

Bibliographie

[1] **A. Gherboudj**. Méthodes de résolution de problèmes di ciles académiques. Pour l'obtention du diplôme de Doctorat. Université de Constantine2. 2013

[2] **A.Sedoglavic**, Licence miage. Universite Lille 1.Cours 08.Semestre 6. 2013

[3] **A. Si Tayeb**. Méta heuristiques pour l'optimisation des puissances actives dans un réseau d'énergie électrique.Magistère en Electrotechnique.Université de Sciences et Technologie d'Oran,2011

[4] **C. Mancel**. Modelisation et resolution de problemes d'optimisation combinatoire issus d'applications spatiales. Automatique / Robotique. INSA de Toulouse, 2004. Français.

[5] **H. Bara**.Développement et implémentation d'une méthode hybride pour la résolution du problème d'assignation quadratique.Présenté pour l'obtention du diplôme de Master. Universite Mohamed Boudiaf. 2019

[6] **HEC MONTR¯EAL** https://www.hec.ca/cams/rubriques/algorithme_simplexe.pdf.

[7] **L. Belhoul**. Résolution de problèmes d'optimisation combinatoire mono et multi-objectifs par énumération ordonnée. Autre [cs.OH]. Université Paris Dauphine - Paris. Français.2014

[8] **M.Djaalab**.Diplôme de master Académique.Le problème du voyageur de commerce : méthodes exactes, méthodes approchées. Université Mohamed Boudiaf -M'sila. 2017

[9] **P. Fouilhoux**.Mise en oeuvre d'un algorithme de Branch-and-Cut avec le framework SCIP. Thèse Master .Université Pierre et Marie Curie Master Androide.26 mars 2015.

[10] **R. BENABID**.Mémoire de Magister en Electrotechnique Optimisation Multiobjectif de la Synthèse des FACTS par les Particules en Essaim pour le Contrôle de la Stabilité de Tension des Réseaux Electriques.Université Amar Telidji, Laghouat.2007

[11] **T**echnique de branch and bound. Chapiter 06:la méthode de Branche and Bound. Cours hiver 2005.

[12] **Y. OULD-SAADI** . Conception d'une Métaheuristique Réactive. Pour l'obtention du diplôme de Magister en Informatique. Universite des sciences et de la technologie d'ORAN Mohamed Boudiaf,2012.

[13] **Z. KHERROUBI Zakiya. R. BEN AHMED.** Optimisation d'un problème d'ordonnancement de type job shop avec contrainte de transport. Memoire du master.Université Abou bekr Belkaid Tlemcen 2017

[14] https://fr.wikipedia.org/wiki/Optimisation_(math%C3%A9matiques)

الملخص

في هذه المذكرة قمنا بدراسة طرق حلول بعض مسائل التحليل التوفيقي، وكمثال على ذلك مسألة الجدولة.

لحل هذه المسألة اقترحنا الة واحدة واقترحنا خوارزميتين تقريبيتين تظهر فعاليتهما من خلال اختلاف حركة الأنشطة على هذه الألة

الكلمات المفتاحية: التحليل التوفيقي، الجدولة، خوارزمية

Résumé

Dans ce mémoire, nous avons fait une étude des méthodes de résolution de certains problèmes d'optimisations combinatoire, par exemple les problèmes d'ordonnancement.

Pour résoudre ces problèmes, nous avons proposé une machine et nous avons suggéré deux algorithmes approchés qui montrent leur efficacité à travers les différents mouvements d'activités sur cette machine.

Mots clés : optimisation combinatoire, ordonnancement, algorithme

Abstract

In this thesis, we have studied the methods of solving some combinatorial optimization problems, as an example, the scheduling problems.

To solve these problems, we have proposed a machine and suggested two approximate algorithms which show their efficiency through the different movements of activities on this machine.

Keywords: combinatorial optimization, scheduling, algorithm