



UNIVERSITE MOHAMED BOUDIAF - M'SILA
FACULTE DES MATHEMATIQUES ET
DE L'INFORMATIQUE



DEPARTEMENT DES MATHEMATIQUES

MEMOIRE de fin d'étude

Présenté pour l'obtention du diplôme de MASTER

Domaine : Mathématiques et Informatique

Filière : Mathématiques

Spécialité : Mathématiques discrètes

Par

Bakour Lyazid

SUJET

Sur la méthode de recuit simulé

Soutenu publiquement le : 04/06/2016 devant le jury composé de :

Mr. Lamiche Chaabane

Mr. Belouadah Hocine

Dr. Bounab Noura

MCA. Université de M'sila

Prof. Université de M'sila

MCB. Université de M'sila

Président

Rapporteur

Examinatrice

Promotion : 2015 /2016

Remerciements

Je tiens à exprimer toute ma reconnaissance à mon Directeur de mémoire PHD Belouadah Hocine, je la remercie de m'avoir encadré, orienté, aidé et conseillé. Je le remercie pour ses encouragements continuels qui ne cessaient de me remonter le moral pendant les moments difficiles et pour l'énorme soutien scientifique et moral qu'il a su m'accorder pendant cette période.

Je remercie Docteur Lamiche Chaabane pour avoir accepté de présider mon jury. Je remercie également Dr Bounab Noura, membre du jury d'avoir accepté l'examination et l'évaluation de ce travail.

J'adresse aussi mes vifs remerciements à mon cher père et à ma chère mère pour leur encouragement et le soutien affectif et matériel qu'ils m'ont apporté tout au long de mon existence.

Je remercie aussi mes frères, mes sœurs, mes chers, mes collègues, surtout mes amis Fradj Farid et Bachiri taki, mes enseignants et tout le personnel de la faculté MI, ainsi que toutes les personnes qui m'ont apporté un soutien moral de loin ou de près.

Résumé

Les problèmes NP-complets d'optimisation combinatoire sont caractérisés par une complexité exponentielle ou factorielle, par conséquent, il est impossible d'énumérer toutes les solutions possibles car cela dépasse la capacité de calcul de n'importe quel ordinateur. Il est donc très difficile de trouver la solution optimale. Pour palier à ces problèmes, les chercheurs ont introduit des méthodes approchées appelées heuristiques, elles présentent l'avantage d'un temps de calcul réduit mais ne donnent aucune information sur la qualité de la solution trouvée, de plus elles ne sont en général applicables qu'à un seul type de problèmes, par exemple la méthode de la descente. Ce qui a poussé les chercheurs à proposer de nouvelles méthodes générales applicables à la plupart des problèmes d'optimisation appelées métaheuristiques, dont la méthode du recuit simulé, conçu pour rechercher un optimum global parmi plusieurs minimums (ou maximums) locaux.

Abstract

The NP-complete problems combinatorial optimization are characterized by an exponential or factorial complexity, therefore, it is impossible to enumerate all possible solutions because it exceeds the computing capacity of any computer. It is therefore very difficult to find the optimal solution. To overcome these problems has, the researchers introduced methods approximate called heuristics, they have the advantage of a time calculation reduced but give no information on the quality of the solution found, they are more generally applicable has only one type of problems, for example the method of the descent. This has led researchers to propose new general methods applicable to most optimization problems called metaheuristics, the method of simulated annealing, designed to search for global optimum of several minimum (or maximum) premises.

ملخص

مشاكل التحسين التوافقي صعبة الحل تتميز بشدة تعقيد أسية أو عاملية، بحيث لا نستطيع إيجاد حل مثالي لهذه المشاكل لأنها تحتاج الى وقت ومساحة كبيرين تفوق قدرات أي حاسوب في الوقت الراهن.

لهذا لجأ الباحثون الى إيجاد طرق تقريبية مختصة لحل مشكل معين، لكن للأسف لا تعطي حلول جيدة كطريقة البحث المحلي، مما جعل الباحثين لإيجاد نوع آخر من الطرق التقريبية تتوافق مع معظم المشاكل التوافقية الصعبة بحيث تعطي حلول تقريبية أحسن من الأولى، كطريقة محاكاة الصلب.

Table des matières

Introduction générale	1
1 Généralités sur les problèmes d'optimisation	3
1.1 Introduction	3
1.2 Notions générales en optimisation	4
1.3 La complexité d'un problème d'optimisation	6
1.3.1 Notions sur la complexité	6
1.3.2 Les problèmes NP-Complets versus les problèmes NP-Difficiles	7
1.3.3 La relation entre les problèmes P et NP	8
1.4 Exemples de problèmes d'optimisation	9
1.4.1 Le problème d'ordonnancement	9
1.4.2 Le problème du sac à dos	9
1.4.3 Le problème de tournées de véhicules	11
1.4.4 Le problème du voyageur de commerce	12
1.4.4.1 Historique de problème	12
1.4.4.2 Complexité du problème	13
1.4.4.3 Problèmes réels se modélisent sous forme du TSP	14
1.4.4.4 Problèmes symétrique et asymétrique	14
1.4.5 Le problème de coloration des graphes	15
1.5 Conclusion	16
2 Les méthodes de résolution : Le recuit simulé	17
2.1 Introduction	17
2.2 Les méthodes de résolution de problèmes d'optimisation	18
2.2.1 Les méthodes approchées	18
2.2.1.1 La heuristique	18
2.2.1.2 Les métaheuristiques	19
2.3 Le recuit simulé	20
2.3.1 La méthode de recherche locale	20
2.3.1.1 L'algorithme général de la descente	20
2.3.2 L'algorithme de Metropolis	22
2.3.3 L'algorithme du recuit simulé	22
2.3.3.1 Principes généraux	24
2.3.3.2 Etat initial de l'algorithme	24
2.3.3.3 La probabilité d'acceptation	24
2.3.3.4 Paramètre de température	25
2.3.3.5 Condition d'arrêt	26

2.3.3.6	Avantages et inconvénients	26
2.4	Conclusion	27
3	Etude de cas : Application de la méthode du recuit simulé sur le TSP et le KP	28
3.1	Introduction	28
3.2	Programmation et implémentation	28
3.2.1	Le choix du langage de programmation	28
3.2.2	Les structures des données	29
3.2.3	Le choix de voisinage	29
3.2.4	Phase expérimentale	30
3.2.5	Tests et résultats	30
3.2.5.1	Le cas du problème du voyageur de commerce	30
3.2.5.2	Le cas du problème de sac à dos	35
3.2.5.3	Discussions	37
3.3	Conclusion	37
	Conclusion générale	38
	Bibliographie	39

Table des figures

1.1	Courbe représentant les optimums locaux et les optimums globaux	6
1.2	la relation entre les différents classes du problème	8
1.3	Exemple de tournées avec 4 véhicules.	11
2.1	le schéma générale de la solution dans la méthode du descente.	21
2.2	Organigramme d'algorithme de la descente simple	21
2.3	le schéma générale de la solution dans la méthode du recuit simulé.	23
2.4	Organigramme d'algorithme du recuit simulé	23

Liste des tableaux

1.1	Nombre de trajets possibles et temps de calcul estimé	14
3.1	Le résultat d'exécution pour 200 villes.	31
3.2	Le résultat d'exécution pour 300 villes.	32
3.3	Le résultat d'exécution pour 500 villes.	33
3.4	Le résultat d'exécution pour 600 villes.	34
3.5	Le résultat d'exécution pour 10000 objets.	36
3.6	Le résultat d'exécution pour 10200 objets.	36
3.7	Le résultat d'exécution pour 10300 objets.	36
3.8	Le résultat d'exécution pour 10400 objets.	37
3.9	Le résultat d'exécution pour 10650 objets.	37

Introduction générale

Les problèmes d'optimisation apparaissent dans plusieurs domaines, telle que la conception de systèmes mécaniques, traitement d'image, l'électronique et la recherche opérationnelle. Résoudre un tel problème est de donner les bonnes valeurs des paramètres pour avoir une solution optimale. Ces valeurs doivent respecter les contraintes liées au problème.

La résolution d'un problème d'optimisation combinatoire nécessite l'utilisation d'un procédé algorithmique permettant la maximisation ou la minimisation d'une ou de plusieurs fonctions d'objectif en respectant les contraintes posées par le problème. Le nombre de fonctions d'objectif du problème à traiter permet de le classer en problème mono objectif ou problème multi objectifs. En fait, un problème mono objectif consiste à optimiser une seule fonction d'objectif. Tandis qu'un problème multi objectifs consiste à optimiser deux ou plusieurs fonctions d'objectif souvent contradictoires. La résolution de ce type de problèmes consiste à trouver un compromis entre les différents objectifs imposés.

La complexité d'un problème donné est liée à deux facteurs : le temps de calcul nécessaire pour sa résolution et l'espace mémoire requis. Ces derniers (temps et mémoire) augmentent avec la taille du problème traité. La théorie de la complexité permet de classer les problèmes selon leur complexité en deux classes essentielles : les problèmes P et des problèmes NP . Un problème P est un problème facile à résoudre. Il possède une solution qui peut être trouvée dans un temps polynomial avec la taille de l'instance traitée. Tandis qu'un problème NP est un problème difficile à résoudre. Il possède une solution qui peut être trouvée dans un temps exponentiel ou pire que exponentiel par rapport à la taille de l'instance traitée.

les méthodes de résolution de problèmes ont été classées en deux catégories : les méthodes exactes et les méthodes approchées. 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. Tandis que, les méthodes approchées sont connues par le fait qu'elles ne garantissent pas l'optimalité de la solution, mais elles nécessitent des coûts de recherche raisonnables. Par conséquent, les méthodes exactes sont plus pratiques pour la résolution de problèmes faciles. En revanche, les méthodes approchées sont plus pratiques dans le cas où on cherche une solution de bonne qualité dans un bref délai. Particulièrement, pour la résolution de problèmes difficiles dont la résolution exacte est très coûteuse.

Les méthodes approchées sont classées en deux catégories : les heuristiques et les métaheuristiques, une métaheuristique à base de population de solutions manipule un ensemble de solutions parallèlement. Elle lance la recherche avec une population de solutions et essaye d'améliorer leurs qualités au cours des itérations dans le but de fournir la ou les meilleures solutions trouvées. La majorité de métaheuristiques sont inspirées de systèmes naturels, nous pouvons citer à titre d'exemple : le recuit simulé qui est inspiré d'un processus métallurgique, les algorithmes génétiques qui sont inspirés des principes de l'évolution Darwinienne et de la biologie, la recherche tabou qui s'inspire de la mémoire des êtres humains, les algorithmes basés sur l'intelligence par essaim comme l'algorithme d'optimisation par essaim de particules, l'algorithme de colonies de fourmis, l'algorithme de colonies d'abeilles, la recherche coucou.

Nous présentons notre travail de mémoire en 3 chapitres. Dans le premier chapitre, nous essayons d'aborder quelques notions de base en optimisation. Dans le deuxième chapitre, nous présentons la méthode de recuit simulé pour résoudre un problème d'optimisation. Dans le troisième chapitre est consacré au problème de voyageur de commerce et le problème de sac à dos (étude de cas), la méthode de recuit simulé.

Chapitre 1

Généralités sur les problèmes d'optimisation

1.1 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 afin d'en extraire une solution optimale parmi un ensemble fini de solutions, 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 ou la minimisation d'une fonction objectif en respectant certaines contraintes rendant infaisable une partie de l'espace de recherche. Malgré que la définition de problèmes d'optimisation soit une tâche souvent facile, la résolution exacte de ce type de problème est une tâche souvent difficile et prohibitive en termes de temps de calcul et de l'espace mémoire requis. Ces derniers, augmentent avec la taille du problème traité. Plus elle est grande, plus l'algorithme de résolution est coûteux.

La théorie de la complexité permet de mesurer la complexité des problèmes en considérant le temps de calcul nécessaire et l'espace mémoire requis pour en résoudre. En effet, elle classe les problèmes en deux classes essentielles : la classe P et la classe NP. Par conséquent, elle permet d'aider au choix général de l'algorithme de résolution d'un problème donné en fonction de la difficulté intrinsèque du problème, qui selon son degré permet de partager les problèmes en deux types : faciles et difficiles à résoudre. En plus du classement des problèmes imposé par leur complexité, les problèmes d'optimisation peuvent aussi être classés en deux autres classes en fonction du nombre de critères à satisfaire. On parle de problèmes mono-objectifs et de problèmes multi-objectifs. outre la classification selon la complexité et selon le nombre de critère à satisfaire, les problèmes d'optimisation peuvent aussi être classés en problèmes industriels ou en problèmes académiques.

1.2 Notions générales en optimisation

Deux types de problèmes d'optimisation sont distingués : des problèmes de minimisation et des problèmes de maximisation. Un problème d'optimisation est défini par un ensemble de données et un ensemble de contraintes. Un ensemble de solutions S est associé au problème d'optimisation. Parmi les solutions S , un sous-ensemble $X \subseteq S$ représente des solutions réalisables respectant les contraintes C du problème, à chaque solution s est associée une valeur $f(s)$ qui représente sa qualité. La résolution du problème d'optimisation consiste à trouver une solution $s \in X$ qui minimise ou maximise la valeur $f(s)$.

Definition 1.2.1. Une instance d'un problème est un couple (X, f) . Où $X \subseteq S$ est un ensemble fini de solutions potentielles admissibles et f est une fonction du coût (fonction d'objectif) à minimiser ou maximiser, $f : X \rightarrow \mathbb{R}$. L'objectif est de trouver $s^* \in X$ tel que $f(s^*) \leq f(s)$ (Au cas de maximisation : $f(s^*) \geq f(s)$) pour n'importe quelle solution $s \in X$.

Definition 1.2.2. Un problème d'optimisation combinatoire est défini par l'ensemble S des solutions possibles d'un problème. $X \subseteq S$ est l'ensemble des solutions réalisables. $f : X \rightarrow \mathbb{R}$, une fonction que l'on nomme fonction d'objectif. La résolution du problème consiste à minimiser (maximiser) la valeur $f(s)$, où $s \in X$.

Definition 1.2.3. Un problème mono-objectif est décrit comme suit :

$$\text{Opt } f(s)$$

$$\text{Avec } \begin{cases} C_i(s) \leq 0 & i = 1 \dots m \\ H_j(s) = 0 & j = 1 \dots p \\ s \in S \subseteq \mathbb{R}^n \end{cases}$$

$$\text{Tel que } \text{Opt} = \begin{cases} \text{Min} & \text{au cas de problème de minimisation} \\ \text{Max} & \text{au cas de problème de maximisation} \end{cases}$$

$$\Omega \text{ est remplacé par } \begin{cases} \leq & \text{au cas de problème de minimisation} \\ \geq & \text{au cas de problème de maximization} \end{cases}$$

- f est la fonction du coût .
- C_i sont des contraintes d'inégalité et H_i sont des contraintes d'égalité.
- S est l'ensemble de solutions possibles (l'espace de recherche).
- s est une solution admissible, elle respecte les contraintes du problème.

Definition 1.2.4. Une solution d'un problème d'optimisation est un ensemble de quantités souvent numériques pour lesquelles des valeurs sont à choisir. L'ensemble de ces valeurs est généralement regroupé dans un vecteur représentant une solution. Supposant un problème de taille n , le vecteur représentant la solution s est représenté par : $(s_1, \dots, s_n)$.

Definition 1.2.5. Une contrainte d'un problème est une restriction imposée par la nature et les caractéristiques du problème sur les solutions proposées.

Definition 1.2.6. L'espace de recherche S d'un problème est composé de l'ensemble de valeurs pouvant être prises par les variables $(s_1, \dots, s_n)$ qui construisent la solution s .

Definition 1.2.7. Le voisinage $V(s)$ d'une solution s est un sous ensemble de S , dont les membres sont des solutions proches (voisines) de la solution s . En effet, on dit qu'une solution s' est une voisine de s , si elle peut être obtenue en modifiant légèrement la solution s .

Definition 1.2.8. L'optimum local est la meilleure solution appartenant à un voisinage $V(s)$ de la solution s . Une solution s' (appartenant à S) est un optimum local de la structure du voisinage $V(s)$ de la solution s si elle vérifie la condition suivante :

$$f(s') \leq f(s) \quad \forall s \in V(s)$$

Pour un problème de maximisation l'inégalité est inversée, c-à-d la condition sera remplacée par :

$$f(s') \geq f(s) \quad \forall s \in V(s)$$

Definition 1.2.9. Une solution optimale (L'optimum global) : Une solution s^* est dite optimale si les deux contraintes suivantes sont vérifiées :

- Elle est réalisable : tirée de l'ensemble de solutions possibles (l'espace de recherche S) et respecte toutes les contraintes du problème posé.
- $f(s^*) = \text{opt}_{s \in X} \{f(s)\}$

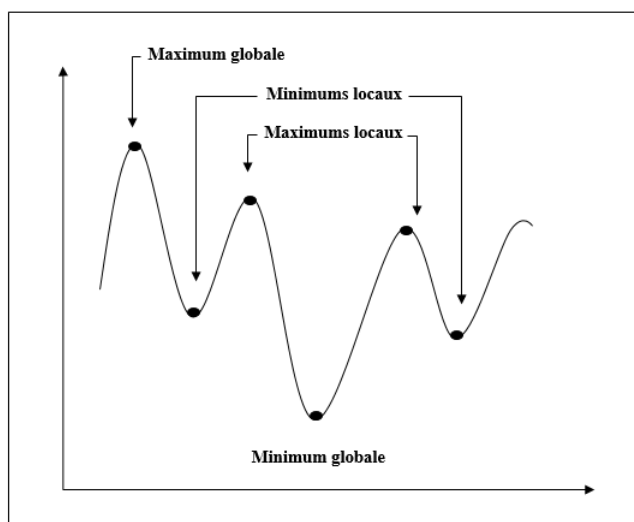


FIGURE 1.1 – Courbe représentant les optimums locaux et les optimums globaux

1.3 La complexité d'un problème d'optimisation

La complexité d'un problème donné est discutée sur deux cotés : coté temporel (complexité temporelle) et coté spatial (complexité spatiale). La complexité temporelle consiste à évaluer le temps de calcul nécessaire pour résoudre un problème donné. Tandis que la complexité spatiale permet d'estimer les besoins en mémoire (l'espace mémoire requis) pour la résolution d'un problème donné. La complexité d'un problème donné est estimée en fonction du nombre d'instructions permettant d'aboutir à la solution du problème posé. Elle est influencée par la taille du problème en question. En effet, elle exprime un rapport entre la taille du problème, le temps de calcul nécessaire et l'espace mémoire requis. Les problèmes sont classés selon leur complexité en deux classes principales : la classe P (Polynomial) et la classe NP (Non Polynomial). Et les problèmes de la classe NP sont partagés en deux sous classes : NP-Complet et NP-Difficile.

1.3.1 Notions sur la complexité

Définition 1.3.1. Une fonction $f(n)$ est $O(g(n))$ ($f(n)$ est de complexité $g(n)$), s'il existe un réel $c > 0$ et un entier positif n_0 tel que pour tout $n \geq n_0$ on a $|f(n)| \leq c.g(n)$.

Définition 1.3.2. Un algorithme en temps polynomial est un algorithme dont le temps de la complexité est en $O(p(n))$, où p est une fonction polynomiale et n est la taille de l'instance (ou sa longueur d'entrée). Si k est le plus grand exposant de ce polynôme en n , le problème correspondant est dit être résoluble en $O(n^k)$ et appartient à la classe P.

Exemple 1.3.1. *Le calcul du plus grand diviseur commun est un exemple d'un problème appartenant à la classe P .*

Definition 1.3.3. *La classe NP contient les problèmes de décision qui peuvent être décidés sur une machine non déterministe en temps polynomial. C'est la classe des problèmes qui admettent un algorithme polynomial capable de tester la validité d'une solution du problème. Intuitivement, les problèmes de cette classe sont les problèmes qui peuvent être résolus en énumérant l'ensemble de solutions possibles et en les testant à l'aide d'un algorithme polynomial.*

Exemple 1.3.2. *Le problème du sac à dos, le problème du voyageur de commerce, les problèmes d'ordonnancement, le problème de coloration de graphes sont des exemples de problèmes appartenant à la classe NP .*

Definition 1.3.4. *On dit qu'un problème de recherche P_1 se réduit polynomialement à un problème de recherche P_2 par la réduction de Turing s'il existe un algorithme A_1 pour résoudre P_1 utilisant comme sous programme un algorithme A_2 résolvant P_2 , de telle sorte que la complexité de A_1 est polynomiale, quand on évalue chaque appel de A_2 par une constante.*

Definition 1.3.5. *La classe NP -complet : parmi l'ensemble des problèmes appartenant à NP , il en existe un sous ensemble qui contient les problèmes les plus difficiles : on les appelle les problèmes NP -complets. Un problème NP -complet possède la propriété que tout problème dans NP peut être transformé (réduit) en celui-ci en temps polynomial. C'est à dire qu'un problème est NP -complet quand tous les problèmes appartenant à NP lui sont réductibles. Si on trouve un algorithme polynomial pour un problème NP -complet, on trouve alors automatiquement une résolution polynomiale de tous les problèmes de la classe NP .*

Exemple 1.3.3. *Le premier problème qui a été démontré comme étant NP -Complet a été celui de la satisfiabilité d'une formule logique binaire (SAT). Cela a été établi en 1971 par Stephen Cook dans son théorème publié dans [1] .*

Definition 1.3.6. *La classe NP -difficile : un problème est NP -difficile s'il est plus difficile qu'un problème NP -complet, c'est à dire s'il existe un problème NP -complet se réduisant à ce problème par une réduction de Turing.*

1.3.2 Les problèmes NP -Complets versus les problèmes NP -Difficiles

Les problèmes NP -Complets et les problèmes NP -Difficiles sont deux types de problèmes difficiles. Aucun algorithme polynomial permettant leur résolution n'a été trouvé. En fait, les algorithmes proposés dans la littérature sont de complexité exponentielle ou pire qu'exponentielle. La différence entre les problèmes de ces deux types de problèmes réside dans le type de la solution attendue. Les problèmes NP -Complets sont des problèmes décisionnels dont la réponse attendue pourra être « oui ou non ».

Exemple 1.3.4. *Le problème du cycle hamiltonien qui consiste à répondre à la question : Est-ce qu'il existe un cycle permettant de parcourir tous les sommets d'un graphe non orienté en passant par chacun une seule fois ? Ici on s'attend à une des deux réponses : oui il existe ou non il n'existe pas.*

Par contre, les problèmes NP-Difficiles concernent beaucoup plus les problèmes d'optimisation ou on cherche la solution optimale.

Exemple 1.3.5. *Le problème du voyageur de commerce dont on cherche le plus court chemin permettant de parcourir un ensemble de sommets (villes) en passant par chacun une seule fois.*

1.3.3 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 oui, 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 $NP \not\subseteq P$. 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 [2] .

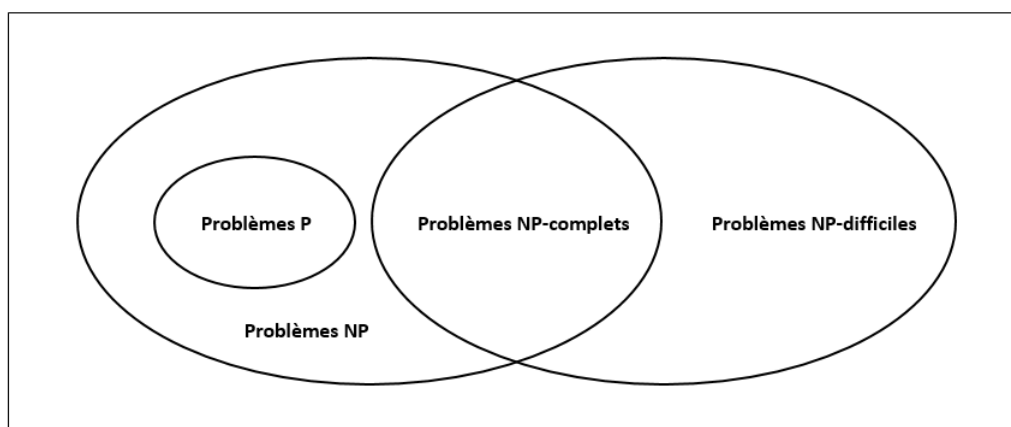


FIGURE 1.2 – la relation entre les différents classes du problème

1.4 Exemples de problèmes d'optimisation

Les problèmes d'optimisation peuvent être des problèmes académiques ou industriels. En fait, les problèmes industriels sont des projections des problèmes académiques sur le plan de la pratique. Parmi les problèmes d'optimisation académiques.

1.4.1 Le problème d'ordonnancement

Un problème d'ordonnancement consiste à planifier la réalisation des tâches en utilisant des ressources et en tenant compte de deux contraintes indispensables et souvent inséparables : le temps de réalisation et la disponibilité des ressources utilisées. Une tâche est un ensemble d'opérations dont l'exécution peut être réalisée par morceaux ou sans interruption. L'ensemble des opérations constituant une tâche donnée est localisé dans le temps par des dates de début et des dates de fin. En outre, la réalisation de l'ensemble des opérations des différentes tâches s'effectue par l'intermédiaire des ressources avec certaine disponibilité. Les ressources sont des moyens permettant la réalisation des opérations. Une ressource peut être un homme, une machine, une matière première, l'argent, un équipement nécessaire... etc. Quelque soit la ressource utilisée, sa disponibilité n'est plus éternelle. Selon sa disponibilité, la ressource peut être renouvelable ou consommable (non renouvelable). Une ressource est renouvelable si elle est à nouveau disponible en même qualité après avoir été allouée à une ou plusieurs opérations, comme le cas où la ressource est un homme ou une machine. Dans le cas contraire, la ressource est dite consommable comme le cas où la ressource est une matière première ou un budget [3].

Un problème d'ordonnancement peut être exprimé comme suit : étant donné un ensemble de tâches à réaliser et un ensemble de machines disponibles pour la réalisation des différentes tâches, le problème consiste à trouver l'ordre de passage des tâches par les différentes machines et la date de début et la date de fin de chaque machine par chaque tâche. La plupart des problèmes d'ordonnancement sont des problèmes d'optimisation qui appartiennent à la classe des problèmes NP-Difficiles [4]. Dans la résolution d'un problème d'ordonnancement, on cherche souvent à optimiser plusieurs critères tels que : le temps total d'exécution, le temps moyen d'achèvement d'un ensemble de tâches, les différents retards (maximum, moyen, somme... etc.) ou avances par rapport aux dates limites fixées.

Il existe plusieurs types de problèmes d'ordonnancement, les plus connus sont : Le problème Job-Shop, le problème Flow-Shop et le problème Open-Shop

1.4.2 Le problème du sac à dos

Le problème du sac à dos est un des problèmes d'optimisation académiques les plus importants. Il est noté par KP (De l'anglais, Knapsack Problem), son

importance est liée en particulier au fait qu'il peut être utilisé en tant que sous problème dans de nombreux problèmes d'optimisation.

L'importance du problème du sac à dos a sollicité l'attention de plusieurs communautés de chercheurs, depuis longtemps et jusqu'aujourd'hui. En fait, il a fait l'objet de plusieurs travaux proposés dans la littérature. Plusieurs variantes du problème du sac à dos ont été proposées dans la littérature. Elles peuvent être classées en sous classes selon plusieurs critères y compris le type de variables, le nombre de contraintes, le nombre d'objectifs etc.

Dans sa version la plus simple, le KP peut être défini comme suit : Supposant que nous avons un sac à dos qui a une capacité maximale C . D'autre part, nous avons un ensemble de N objets. Chaque objet i a un poids w_i et un profit p_i . Le problème consiste à sélectionner un sous ensemble d'objets de manière à maximiser le profit total du sac à dos sans dépasser sa capacité maximale. Le problème peut être formulé comme suit :

$$\text{Maximiser } \sum_{i=1}^N p_i x_i$$

$$\text{Avec } \sum_{i=1}^N w_i x_i \leq C$$

$$\text{Et } x_i = \begin{cases} 1 & \text{si l'objet } i \text{ est sélectionné} \\ 0 & \text{sinon} \end{cases}$$

Il existe plusieurs types de problèmes du sac à dos, les plus connus sont : le problème du sac à dos unidimensionnel, Le problème du sac à dos multidimensionnel, le problème du sac à dos à choix multiple et Le problème du sac à dos quadratique.

Il est à noter que le problème du sac à dos unidimensionnel (appelé tout court : problème du sac à dos) est un problème d'optimisation combinatoire et qu'il appartient à la classe des problèmes NP-Difficiles [5] .

Exemple 1.4.1. *considérant une famille qui veut se déplacer vers une ville côtière pour passer des vacances estivales. Le papa a demandé aux membres de sa famille de préparer une liste des objets importants qui leur seraient utiles et il a préparé sa voiture pour voyager. La capacité de la voiture est limitée, elle a une capacité maximale qu'elle ne peut pas dépasser. Chacun des membres de la famille a préparé sa propre liste et ils ont construit une grande liste qu'ils ont présentée au papa. Outre leurs poids (i.e. Les poids des membres de la famille) non discutables, chaque objet de la liste préparée a un poids et une certaine importance (profit). De sa part, le papa a essayé de calculer le poids total des objets. Il a trouvé que la somme des poids des objets et des poids des membres de la famille dépasse la capacité maximale de la voiture. Le papa s'est trouvé devant le dilemme de satisfaire les besoins des membres de sa famille et de ne pas dépasser la capacité maximale de sa voiture. Par conséquent, il a pensé à*

une solution qui consiste à sélectionner un sous ensemble de l'ensemble d'objets proposés de façon à réduire le poids total des objets et maximiser le profit global sans dépasser la capacité maximale de la voiture [2] .

1.4.3 Le problème de tournées de véhicules

Le problème de tournées de véhicules (VRP : Vehicle Routing Problem) a été introduit pour la première fois en 1959 par Dantzig et Ramser [6] . Le VRP est un des problèmes qui ont sollicité l'attention de plusieurs équipes de recherches vu son importance et son utilité dans la résolution de nombreux problèmes rencontrés dans la vie quotidienne. Le VRP est un problème d'optimisation combinatoire, il appartient à la classe des problèmes NP-Difficiles [7] où il n'existe pas d'algorithme permettant la résolution d'un problème donné en un temps polynomial.

Le principe du problème de tournées de véhicules consiste à trouver les meilleurs trajets à parcourir par une flotte de M véhicules afin de visiter N clients (sites) pour des raisons de collecte ou de livraison. Tous les véhicules commencent et terminent leurs tournées à un site central appelé dépôt. Chaque client est affecté à une tournée et il doit être visité une seule fois par un seul véhicule. La capacité de transport de ce dernier pour une tournée est limitée et elle ne doit pas être dépassée. Tous les trajets proposés doivent respecter les différentes capacités des véhicules et satisfaire les quantités demandées par les clients ou à livrer à ces derniers [8] .

En général, l'objectif principal des problèmes de tournées de véhicules est de minimiser le coût (distance) total de parcours des trajets par un nombre minimum de véhicules. Outre son objectif principal, le VRP impose d'autres contraintes à respecter parmi lesquelles :

- Chaque client doit être servi une seule fois par un et un seul véhicule.
- Le client visité doit impérativement être quitté.
- Le nombre de véhicules est limité.
- Les véhicules doivent tous commencer leurs tournées du dépôt.

Exemple 1.4.2. La figure représente un exemple de tournée avec quatre véhicules qui assurent la visite de 21 clients tout en partant et revenant au dépôt.

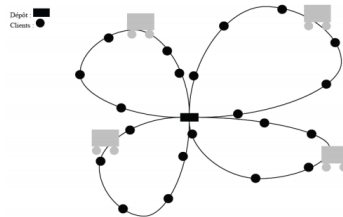


FIGURE 1.3 – Exemple de tournées avec 4 véhicules.

1.4.4 Le problème du voyageur de commerce

Le problème du voyageur de commerce (en anglais Travelling Salesman Problem : TSP) est un problème classique d'optimisation combinatoire. Le TSP est un cas particulier du problème de Tournées de Véhicules sans contrainte de capacité et avec un seul véhicule. Dans le problème du TSP, on suppose un voyageur de commerce qui doit visiter n villes données, en passant par chaque ville exactement une et une seule fois. Il commence par une ville quelconque et doit terminer en retournant à la ville du départ. Les distances entre les villes sont connues. Le problème consiste à déterminer le chemin permettant de minimiser la distance parcourue. La notion de distance peut être remplacée par d'autres notions comme le temps qu'on met ou l'argent qu'on dépense : dans tous les cas, on parle du coût. De même, la notion de ville peut être remplacée par d'autres notions comme la location, l'entrepôt... etc. Le problème du voyageur de commerce peut être formulé comme suit :

$$\text{Minimiser } \sum_{i=1}^{n-1} (D(v_i, v_{i+1})) + D(v_n, v_1)$$

Où $D(v_i, v_{i+1})$ est la distance entre les deux villes v_i et v_{i+1} .

1.4.4.1 Historique de problème

Des problèmes mathématiques liés au problème de voyager de commerce ont été étudiés dans les 1800's par le mathématicien Irlandais Sir William Rowan Hamilton et par le mathématicien Britannique Thomas Penyngton Kirkman. Hamilton a créé le jeu Icosian en 1857 qui nécessite un lecteur de terminer une tournée en utilisant seulement connecteurs spécifiés par 20 points . Dans les années 1930 le TSP est traité plus en profondeur par Karl Menger à Harvard. Il est ensuite développé à Princeton par les mathématiciens Hassler Whitney et Merrill Flood [9] .

- 1954 Solution du TSP pour 49 villes par Dantzig, Fulkerson et Johnson par la méthode du cutting-plane, les villes représentent 48 capitale état d'ETATS-UNIS plus Washington.
- 1975 Solution pour 100 villes par Camerini, Fratta and Maffioli .
- 1987 Solution pour 532, puis 2392 villes par Padberg et Rinaldi .
- 1998 Solution pour les 13 509 villes des Etats-Unis.
- 2001 Allemagne par Applegate, Bixby, Chvátal et Cook des universités de Rice et Princeton appliquent une version parallèle de la méthode de Danzig, Fulkerson et de Johnson sur une grande instance de TSP, Ils ont obtenu la solution optimale du TSP qui a 15112 villes (nœuds) en Allemagne, Le calcul a été exécuté sur un réseau de 110 processeurs situés et ils ont estimé tout le temps de calcul étaient de 22,6 ans, mesurés à un processeur d'alpha de Compaq EV6(21264) fonctionnant à 500MH .
- En mai 2004, le problème de représentant de commerce de visiter chacune des 24.978 villes en Suède a été résolu : une excursion de longueur approximativement 72.500 kilomètres a été trouvée et on l'a montrée qu'aucune excursion plus courte n'existe.

- En mars 2005, le problème de représentant de commerce de visiter chacun des 33.810 points dans une carte a été résolu en utilisant CONCORDE une excursion des unités de la longueur 66.048.945 a été trouvée et on l'a montrée qu'aucune excursion plus courte n'existe. Le calcul a pris approximativement 15,7 années d'unité centrale de traitement .

1.4.4.2 Complexité du problème

Ce problème appartient à la classe NP-difficile [10] et pour montrer la difficulté de ce problème supposant qu'on a 50 villes, donc ils existent $49!$ Solutions (tourne) possibles, i.e $6,08 * 10^{62}$ tournées possible. Prenons un ordinateur de π Millimètres calculant un milliard de solutions par seconde.

Sachant que le diamètre équatorial de la terre est de 12756 kilomètres, mettons $1,2756.10^{10}$ de ces ordinateurs les uns à la suite des autres sur l'équateur. On peut ainsi calculer $1,2576.10^{19}$ solutions par seconde Pour être certain de trouver la tournée la plus courte, il faut considérer toutes les tournées possibles. Il nous faudra alors $4,766.10^{43}$ secondes avec le super ordinateur que l'on vient de présenter. Note : $4,766.10^{43}$ est l'équivalent de $1,51.10^{34}$ siècle.

D'une manière formelle pour démontrer que TSP est NP-complet nous emploierons la méthode populaire de la réduction. Nous prendrons un problème, l'appelons H, on s'est déjà avéré qu'il est NP-complet. Puis, nous prouverons que vous pouvez résoudre H en le ramenant au TSP. Une fois que nous avons montré ceci, nous pouvons dire que si TSP est soluble dans un temps polynôme, alors nous pouvons résoudre H dans un temps polynôme. À quel point nous atteignons une contradiction nous concluons alors que TSP ne peut pas être résolu dans un temps polynôme parce qu'on l'a déjà montré que H ne peut pas être résolu dans le temps polynôme. D'ailleurs, TSP doit être NP-complet parce que H est NP-complet. En montrant que TSP est NP-complet, nous choisirons H pour être le problème populaire de cycle hamiltonien, qui est connu pour être NP-complet. À un regard étroit, il est évident que le problème de cycle hamiltonien soit un cas spécial de TSP. Tous simplement en modifiant le graphe d'entrée selon l'algorithme de TSP en plaçant les coûts de tous les bords existants pour être une certaine constante finie fixe. Puis, si n'importe quelle excursion est trouvée, cette excursion est un cycle hamiltonien. Ainsi, le TSP doit être NP-complet parce qu'on l'a montré que le problème de cycle hamiltonien est NP-complet.

Nombre de villes	Nombre de possibilités	Temps de calcul
5	12	12 μ
10	181440	0,18 s
15	43 milliards	12 heures
20	60×10^{15}	1928 ans
25	310×10^{21}	9,8 milliards d'années (!)

TABLE 1.1 – Nombre de trajets possibles et temps de calcul estimé

1.4.4.3 Problèmes réels se modélisent sous forme du TSP

Le TSP a des applications directes, notamment dans le transport et la logistique. Par exemple, trouver le chemin le plus court pour les bus de ramassage scolaire, ou dans l'industrie, pour trouver la plus courte distance que devra parcourir le bras mécanique d'une machine pour percer les trous d'un circuit imprimé. Le TSP peut aussi être utile en ordonnancement, si n commandes doivent être effectuées sur une seule machine et que le temps de mise en route dépend de l'ordre de passage, le TSP permettra de minimiser le temps de mise en route des commandes en déterminant l'ordre dans lequel celles-ci doivent être effectuées. Parmi ces problèmes, on cite :

- La tournée des avions.
- Le tour d'un supporteur de base-ball.
- Collection de monnaie des cabines téléphoniques publics.
- La chaîne d'ADN.
- Conception des réseaux à fibres optiques.

1.4.4.4 Problèmes symétrique et asymétrique

Le problème de voyageur de commerce se présente sous différentes formes selon la nature de la distance entre villes (symétrique, asymétrique, euclidien... etc.) ou l'ensemble des villes à visiter (tous les N villes ou bien un sous-ensemble de N ... etc.). Le TSP Symétrique (STSP : Symetric Traveling Salesman Problem) : Le TSP symétrique est le TSP le plus simple, car la distance entre deux villes quelconques x et y est égale à celle entre y et x .

1.4.5 Le problème de coloration des graphes

Il s'agit de colorier tous les sommets d'un graphe tel que chaque sommet ait une couleur différente de la couleur de ses sommets adjacents, tout en minimisant le nombre de couleurs. La coloration de graphe est appliquée dans multiples domaines tels que l'emploi de temps, la coloration des cartes, l'affectation de fréquence en téléphonie mobile, la gestion de chaînes logistiques, la gestion du trafic aérien, l'affectation des registres dans les compilateurs, etc.

Formellement, ce problème est défini comme suit : soit le graphe $G = (V, E)$ avec V l'ensemble des sommets et E l'ensemble des arêtes. Une coloration valide d'un graphe $G = (V, E)$ est une fonction $c : v \rightarrow c(v)$ associant à tout sommet $v \in V$ une couleur $c(v)$, en s'assurant que $c(v) \neq c(u)$ pour toute arête $[u, v] \in E$. Si le nombre de couleurs utilisé est k , la coloration de G est une k -coloration. La valeur minimale de k pour laquelle une k -coloration est réalisable est appelé nombre chromatique du graphe, noté $\chi(G)$. La résolution du problème de la coloration de graphe consiste à décider s'il y a une coloration avec k couleurs (problème de décision NP-complet), soit à trouver son nombre chromatique (problème d'optimisation NP-difficile).

1.5 Conclusion

Dans ce chapitre, nous avons essayé d'aborder quelques notions de base en optimisation. Nous avons présenté des définitions de notions confrontées souvent dans le domaine de l'optimisation en allant de la définition d'un problème d'optimisation, passant par les éléments du problèmes d'optimisation, jusqu'à la définition d'une solution optimale d'un problème donné. Ensuite, nous avons dévoilé le rôle de la théorie de la dans la classification des différents problèmes selon leurs complexités en problèmes P et problèmes NP.

Nous avons donné une vue générale sur les problèmes des deux classes et nous avons essayé d'éclaircir la relation entre les deux classes. Enfin, nous avons tenté d'élucider le principe de quelques problèmes d'optimisation académiques difficiles.

Dans le chapitre qui suit, nous présentons un état de l'art sur quelques méthodes de résolution de problèmes d'optimisation qui ont été proposées dans la littérature, en donnant une importance particulière à la méthode recherche locale et la méthode de recuit simulé.

Chapitre 2

Les méthodes de résolution : Le recuit simulé

2.1 Introduction

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 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 de différents problèmes en deux classes principales : la classe de méthodes exactes et la classe de méthodes approchées.

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 de problèmes faciles. La nécessité de disposer d'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 connues 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 de problèmes difficiles ou de 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.

La particularité qui différencie les méthodes métaheuristiques des méthodes heuristiques c'est que les métaheuristiques sont applicables sur de nombreux problèmes. Tandis que, les heuristiques sont spécifiques à un problème donné. De nombreuses métaheuristiques ont été proposées dans la littérature afin de faire face aux différents problèmes en minimisant le coût de la recherche. Les métaheuristiques peuvent être classées en deux catégories : les méthodes à base de solution unique et les méthodes à base de population de solutions. Les méthodes à base de solution unique se basent généralement sur une recherche locale, malheureusement la plupart de ces méthodes souffrent du problème de la convergence de la recherche vers l'optimum local.

Dans ce chapitre en donnant une importance particulière au méthode métaheuristique du recuit simulé.

2.2 Les méthodes de résolution de problèmes d'optimisation

Pour résoudre un problème d'optimisation, il existe plusieurs méthodes de résolution, ces dernières se partagent en deux classes principaux : la classe des méthodes exactes et la classe des méthodes approchées.

2.2.1 Les méthodes approchées

2.2.1.1 La heuristique

Les méthodes dites heuristiques sont des méthodes spécifiques à un problème particulier. Elles nécessitent des connaissances du domaine du problème traité. En fait, se sont des règles empiriques qui se basent sur l'expérience et les résultats acquis afin d'améliorer les recherches futures.

Definition 2.2.1. *Une heuristique (règle heuristique, méthode heuristique) est une règle d'estimation, une stratégie, une astuce, une simplification, ou tout autre type de dispositif qui limite considérablement la recherche de solutions dans des espaces problématiques importants. Les heuristiques ne garantissent pas des solutions optimales. En fait, elles ne garantissent pas une solution du tout. Tout ce qui peut être dit d'une heuristique utile, c'est qu'elle propose des solutions qui sont assez bonnes la plupart du temps [11] .*

2.2.1.2 Les métaheuristiques

Dans la vie pratique, on se retrouve souvent confronté à des problèmes de différentes complexités, pour lesquelles on cherche des solutions qui satisfont deux notions antagonistes : la rapidité et la qualité. Devant le coût de recherche prohibitif des méthodes exactes (particulièrement avec des problèmes de grande taille) et la spécificité des heuristiques au problème donné, les métaheuristiques construisent une solution moins exigeante. En fait, elles sont applicables sur une grande variété de problèmes d'optimisation de différentes complexités. En outre, elles permettent de fournir des solutions de très bonne qualité (pas nécessairement optimales) en temps de calcul raisonnable.

La majorité des métaheuristiques sont inspirées des systèmes naturels, nous pouvons citer à titre d'exemple : le recuit simulé qui est inspiré d'un processus métallurgique, les algorithmes évolutionnaires et les algorithmes génétiques qui sont inspirés des principes de l'évolution Darwinienne et de la biologie, la recherche tabou qui s'inspire de la mémoire des êtres humains, les algorithmes basés sur l'intelligence d'essaim comme l'algorithme d'optimisation par essaim de particules, l'algorithme de colonies de fourmis, l'algorithme de colonies d'abeilles, la recherche coucou et l'algorithme d'optimisation par coucou qui s'inspirent du comportement social de certaines espèces évoluant en groupe.

Definition 2.2.2. *Une métaheuristique est un processus itératif qui subordonne et guide une heuristique, en combinant intelligemment plusieurs concepts pour explorer et exploiter tout l'espace de recherche. Des stratégies d'apprentissage sont utilisées pour structurer l'information afin de trouver efficacement des solutions optimales, ou presque-optimales [12] .*

Les métaheuristiques à base de solution unique débutent la recherche avec une seule solution initiale. Elles se basent sur la notion du voisinage pour améliorer la qualité de la solution courante. En fait, la solution initiale subit une série de modifications en fonction de son voisinage. Le but de ces modifications locales est d'explorer le voisinage de la solution actuelle afin d'améliorer progressivement sa qualité au cours des différentes itérations. Le voisinage de la solution s'englobe l'ensemble des modifications qui peuvent être effectuées sur la solution elle-même. La qualité de la solution finale dépend particulièrement des modifications effectuées par les opérateurs de voisinages. En effet, les mauvaises transformations de la solution initiale mènent la recherche vers la vallée de l'optimum local d'un voisinage donné (peut être un mauvais voisinage) ce qui bloque la recherche en fournissant une solution de qualité insuffisante.

De nombreuses méthodes à base de solution unique ont été proposées dans la littérature. Parmi lesquelles : la descente, le recuit simulé, la recherche tabou, la recherche à voisinage variable, la recherche locale réitérée, la recherche locale guidée... etc.

2.3 Le recuit simulé

Le recuit simulé est une méthode métaheuristique de résolution de problèmes d'optimisation combinatoire. Elle appartient à la famille des algorithmes de recherche locale dits à seuil. La particularité du recuit simulé est d'être une méthode d'optimisation stochastique (basée sur la théorie des probabilités).

L'algorithme du recuit simulé a été proposé par Kirkpatrick, Gelatt et Vecchi [13]. Son principe se base sur la procédure du recuit des métaux utilisée par les métallurgistes. Ces derniers chauffent à blanc le métal, puis ils laissent l'alliage se refroidir très lentement afin d'aboutir à un alliage sans défauts. En fait, les thermodynamiciens ont remarqué qu'une baisse brutale de la température d'un liquide entraîne une reproduction d'un optimum local, i.e. une structure amorphe. Alors qu'une baisse progressive de la température du liquide permet d'aboutir à un optimum global, i.e. Une structure bien construite. C'est l'idée prise en considération par les métallurgistes qui savent que si le métal refroidit trop vite, il contiendra beaucoup de défauts microscopiques et s'il refroidit lentement ils obtiendront une structure bien ordonnée. La métaheuristique du recuit simulé s'inspire de l'algorithme de Métropolis.

En pratique, le recuit simulé fournit des solutions intéressantes à beaucoup de problèmes classiques d'optimisation combinatoire (voyageur de commerce, coloration de graphe...)

2.3.1 La méthode de recherche locale

La méthode de recherche locale la plus élémentaire est la méthode de descente. La recherche locale simple ou la descente est un algorithme d'amélioration très ancien. Son principe consiste à explorer le voisinage de la solution courante afin d'améliorer sa qualité progressivement. A chaque itération du processus d'amélioration, l'algorithme modifie un ensemble de composantes de la solution courante pour permettre le déplacement vers une solution voisine de meilleure qualité. Le processus est répété itérativement jusqu'à la satisfaction du critère d'arrêt. Il est à noter qu'il existe trois types de la descente : la descente déterministe, la descente stochastique et la descente vers le premier meilleur voisin.

2.3.1.1 L'algorithme général de la descente

On résume les étapes de l'algorithme général de la descente comme suit :

1. L'algorithme entame la recherche par la construction d'une solution initiale s et l'évaluation de sa qualité $f(s)$
2. Modifier s pour obtenir une nouvelle solution s' de meilleure qualité que s et qui appartient à son voisinage.
3. Evaluer la qualité $f(s')$ de la nouvelle solution s' .
4. Remplacer s par s' , si s' est de meilleure qualité.

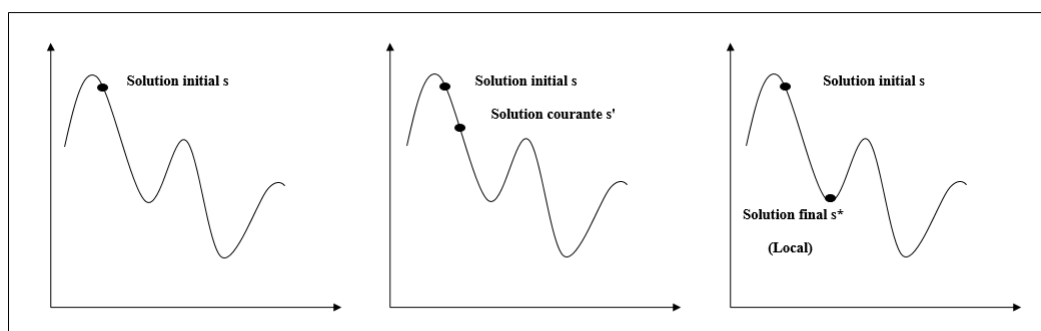


FIGURE 2.1 – le schéma générale de la solution dans la méthode du descente.

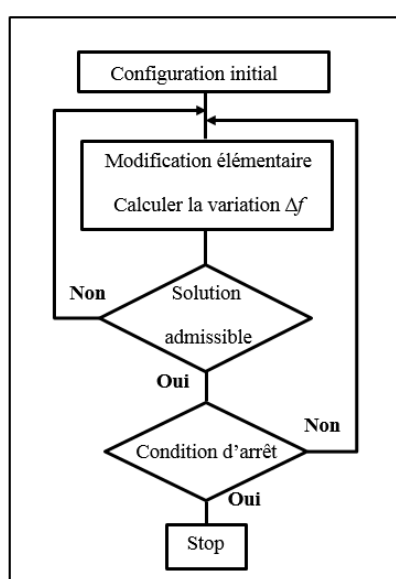


FIGURE 2.2 – Organigramme d'algorithme de la descente simple

Algorithm 1 descente simple

-
- 1: Engendrer une configuration initiale s_0 de $s : s \leftarrow s_0$
 - 2:
 - 3: **while** (la condition d'arrêt n'est pas vérifiée) **do**
 - 4: Engendrer un voisin aléatoire S' de S
 - 5: Calculer $\Delta f = f(s') - f(s_0)$
 - 6:
 - 7: **if** ($\Delta f \leq 0$) **then** (cas de maximisation : $\Delta f \geq 0$)
 - 8: $s \rightarrow s'$
 - 9: **else**
 - 10: s est rejetée
 - 11: **end if**
 - 12: **end while**
 - 13: Retourner la meilleure configuration trouvée
-

2.3.2 L'algorithme de Metropolis

Dans l'algorithme de Metropolis, on part d'une configuration donnée, et on lui fait subir une modification aléatoire. Si cette modification fait diminuer la fonction objectif (ou énergie du système), elle est directement acceptée. Sinon, elle n'est pas acceptée qu'avec une probabilité égale à $e^{-\Delta E/T}$ (avec E : énergie, et T : température), cette règle est appelé critère de Metropolis [14] .

2.3.3 L'algorithme du recuit simulé

Le recuit simulé applique itérativement l'algorithme de Metropolis, pour engendrer une séquence de configurations qui tendent vers l'équilibre thermodynamique. L'énergie du système est représentée par un réel arbitraire T , la température. A partir d'une recherche locale quelconque pour un problème, on obtient une méthode de recuit comme suit :

1. Choisir une température de départ T et une solution initiale $s = s_0$.
2. Générer une solution aléatoire dans le voisinage de la solution actuelle $s_0 \rightarrow s', s' \in V(s)$, au lieu de chercher la meilleur ou la première solution voisine améliorante comme dans une recherche locale classique.
3. On calcule la variation de coût $\Delta f = f(s') - f(s_0)$.
4. Si $\Delta f \leq 0$ (cas de maximisation : $\Delta f \geq 0$), le coût diminue et on effectue la transformation améliorante comme dans une recherche locale ($s_0 = s'$).
5. Si $\Delta f > 0$ (cas de maximisation : $\Delta f < 0$), le coût remonte, c'est un rebond, qu'on va pénaliser d'autant plus que la température est basse et que Δf est grand. Une fonction exponentielle a les propriétés désirées. On calcule une probabilité d'acceptation p , puis on tire au sort r dans $[0, 1]$. Si $p > r$, la transformation est déclarée acceptée, bien qu'elle dégrade le coût, et on fait $s_0 = s'$. Sinon, la transformation est rejetée : on garde s pour l'itération suivante.
6. Pour assurer la convergence (analogie de la balle qui rebondit de moins en moins), T est diminuée lentement à chaque itération, par exemple $T = k.T, k = 0.999$. On peut aussi décroître T par paliers. Pour être efficace, un recuit doit diminuer T assez lentement, en plusieurs milliers ou dizaines de milliers d'itérations. Il dure en tout cas beaucoup plus longtemps qu'une recherche locale, puisque les transformations améliorante de cette dernière sont diluées parmi de nombreux rebonds.
7. On s'arrête si la condition d'arrêt est vérifiée.

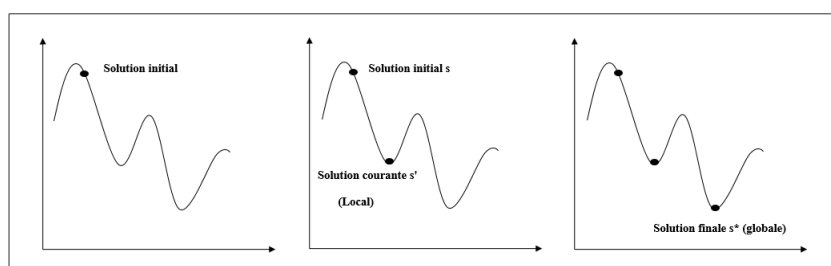


FIGURE 2.3 – le schéma générale de la solution dans la méthode du recuit simulé.

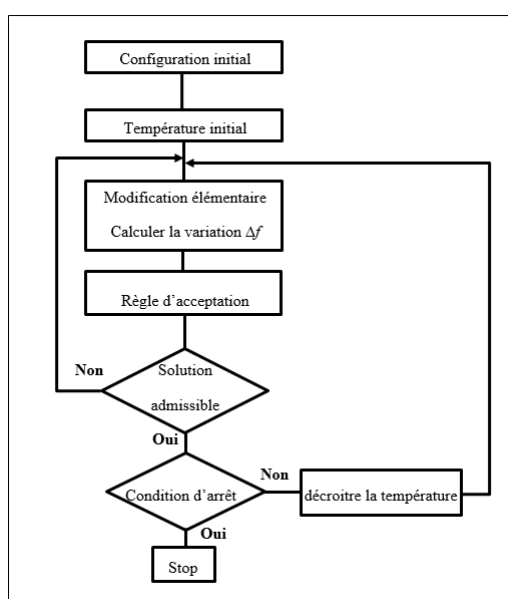


FIGURE 2.4 – Organigramme d'algorithme du recuit simulé

Algorithm 2 Recuit simulé

-
- 1: Engendrer une configuration initiale s_0 de s : $s \leftarrow s_0$
 - 2: Initialiser la température T en fonction du schéma de refroidissement
 - 3:
 - 4: **while** (la condition d'arrêt n'est pas vérifiée) **do**
 - 5: Engendrer un voisin aléatoire S' de S
 - 6: Calculer $\Delta f = f(s') - f(s_0)$
 - 7:
 - 8: **if** ($\Delta f \leq 0$) **then** (cas de maximisation : $\Delta f \geq 0$)
 - 9: $s \rightarrow s'$
 - 10: **else**
 - 11: tirer r dans $[0, 1]$
 - 12: **if** ($p > r$) **then** $s \leftarrow s'$
 - 13: **else**
 - 14: s' est rejetée et conserve s
 - 15: **end if**
 - 16: **end if**
 - 17: **end while**
 - 18: Mettre T à jour en fonction du schéma de refroidissement
 - 19: Retourner la meilleure configuration trouvée
-

2.3.3.1 Principes généraux

L'idée d'utiliser le recuit stimulé pour la résolution d'un problème d'optimisation combinatoire consiste à introduire un paramètre de contrôle jouant le rôle de la température qui conditionne le nombre d'états accessibles (solutions réalisables) et conduit vers un système figé ou vers l'état d'énergie minimale (fonction objectif minimale).

L'efficacité de l'algorithme du recuit simulé appliquée aux problèmes d'optimisation dépend crucialement du choix des paramètres suivants :

- le choix d'une solution initiale.
- le choix d'un voisinage.

La méthode du recuit simulé appliquée aux problèmes d'optimisations combinatoires, considère une solution initiale, le plus souvent inspiré par l'utilisation d'une méthode heuristique rapide de type Clarke et Wright, pour générer rapidement une solution initiale et recherche dans son voisinage une autre solution de façon aléatoire.

L'originalité de cette méthode est qu'il est possible de se diriger vers une solution voisine de moins bonne qualité avec une probabilité non nulle, ce qui permettra de sortir des optima- locaux; c'est-à-dire de bonnes solutions mais qui ne sont pas les meilleures des solutions possibles.

La performance du recuit simulé dépend de la règle de refroidissement c'est-à-dire de la décroissance du paramètre utilisé T , apparenté à la température. Un refroidissement trop rapide mènerait vers un optimum local pouvant être de très mauvaise qualité, un refroidissement trop lent serait très coûteux en temps de calcul.

2.3.3.2 Etat initial de l'algorithme

La solution initiale peut être prise au hasard dans l'espace des solutions possibles. A cette solution correspond une énergie initiale. Cette énergie est calculée en fonction du critère que l'on cherche à optimiser. Une température initiale élevée est également choisie. Ce choix est alors totalement arbitraire et va dépendre de la loi de décroissance utilisée.

2.3.3.3 La probabilité d'acceptation

A partir d'une solution acceptable S , du problème d'optimisation, une nouvelle solution acceptable S' est générée de manière stochastique. Si la nouvelle solution acceptable a pour effet de diminuer le critère, $f(S') < f(S)$ (cas de maximisation : $f(S') > f(S)$), le changement s'opère car l'état est plus stable. Si, au contraire le critère croît, la nouvelle solution S' peut être acceptée ou non selon une règle statistique. il existe deux protocoles classiques communément utilisés :

- la procédure de Metropolis [14] où l'acceptation de la nouvelle solution est réalisée avec une probabilité p telle que :

$$p = \begin{cases} \exp(-\Delta f/T) & \text{si } \Delta f \geq 0 \\ 1 & \text{si } \Delta f < 0 \end{cases}$$

- la variante proposée par [15], où la probabilité d'acceptation p est uniforme, et définie par :

$$p = (\exp(-\Delta f/T)) / (1 + \exp(-\Delta f/T))$$

2.3.3.4 Paramètre de température

Le paramètre T (température) est un réel positif, le choix de la température est primordial pour garantir l'équilibre entre l'intensification et la diversification des solutions dans l'espace de recherche. Premièrement, le choix de la température initiale dépend de la qualité de la solution de départ. Si cette solution est choisie aléatoirement, il faut prendre une température relativement élevée. Une étude [16] a montré qu'une valeur initiale de la température de l'ordre de celle de la fonction objectif initiale, conduit à de bons résultats. Ainsi, l'initialisation proposée est définie par : $T = f(S)/A$.

où $f(S)$ représente la valeur du critère pour la solution initiale S , et A un paramètre à fixer. La décroissance de la température peut également être réalisée (en d'autres termes, elle ne change qu'après un certain nombre d'itérations). une fonction qui spécifie l'évolution de la température est appelé le schéma de refroidissement, dans le recuit simulé standard la température décroît par paliers (en d'autres termes, elle ne change qu'après un certain nombre d'itérations). La longueur du palier, qui représente le nombre de solutions testées avant d'appliquer le schéma de décroissance de la température, est supposée constante pour l'ensemble des niveaux de température. Si cette longueur est trop faible, l'exploration de l'espace de recherche peut être trop partielle, alors qu'une valeur trop importante peut avoir pour effet de ralentir, voire même de bloquer la recherche.

On peut aussi utiliser d'autres schémas de refroidissement, il existe deux schémas classiques de décroissance de la température :

- la décroissance géométrique, proposée par [13] :

$$T_{k+1} = \alpha T_k \quad \text{avec } \alpha \in [0, 1]$$

- la procédure utilisée par [17] définie par :

$$T_{k+1} = \frac{T_k}{(1 + (T_k \ln(1 + \alpha) / 3\sigma))}$$

où σ désigne l'écart-type de l'ensemble des valeurs calculées de la fonction objectif à la température courante, et C le paramètre qui contrôle la vitesse du recuit ; plus C est faible, plus le recuit est lent.

le coefficient α de réduction de la température, qui doit être tel que : $0 < \alpha < 1$ Une valeur trop faible pour ce coefficient peut provoquer une exploration très

partielle de l'espace de recherche. Les valeurs classiques de ce coefficient se situent dans l'intervalle $[0.75, 0.95]$.

2.3.3.5 Condition d'arrêt

La procédure du recuit simulé est souvent arrêtée lorsque la température devient inférieure à une valeur donnée. Le choix de cette valeur limite, qui conditionne l'efficacité de la recherche, est très délicate car ce terme est étroitement lié aux autres paramètres de la méthode (température initiale, longueur des paliers, schéma de réduction de la température).

2.3.3.6 Avantages et inconvénients

Chaque méthode a des avantages et des inconvénients, mais ils diffèrent d'une méthode à d'autre. D'abord les avantages : on observe que la méthode du recuit simulé procure généralement une solution de bon qualité (minimum absolu ou bon minimum relatif de la fonction objectif). En outre, c'est une méthode générale, elle est applicable et facile à programmer pour tous les problèmes qui relèvent des techniques d'optimisation itérative, à conditions toutefois que l'on puisse évaluer directement et rapidement, après chaque transformation, la variation correspondante de la fonction objectif (le temps de calcul est souvent excessif si l'on ne peut éviter de recalculer complètement la fonction objectif, après chaque transformation).

Enfin, elle offre une grande souplesse d'emploi, car de nouvelles contraintes peuvent être facilement incorporées après coup dans le programme [18].

Citons ensuite les inconvénients. Les utilisateurs sont parfois rebutés par le nombre important de paramètres (température initiale, taux de décroissement de la température, durée des paliers et température, critère d'arrêt du programme...). Ces paramètres sont souvent choisis de manière empirique. Le deuxième défaut de la méthode est le temps de calcul, qui est excessif dans certaines applications.

2.4 Conclusion

Dans ce chapitre nous avons essayé de présenter un état de l'art sur les méthodes approchées de résolution de différents problèmes d'optimisation présentées dans la littérature. Nous avons présenté les types des méthodes approchées, ces dernières peuvent être partagées en deux classes : des méthodes heuristiques et des méthodes métaheuristiques. Une méthode heuristique est applicable sur un problème donné, tandis qu'une méthode métaheuristique est plus générique et elle peut être appliquée sur une panoplie de problèmes d'optimisation. Ensuite, nous avons pu constater que la méthode du recuit simulé, se basant sur la recherche locale (descente) pour trouver la solution du problème à traiter, passant par leur algorithme et le principe générale jusqu'à les paramètres principaux, ensuite nous avons dévoilé les avantages et les inconvénients de cette méthode.

Dans le chapitre qui suit, nous avons essayé de programmer l'algorithme du recuit simulé et implémenter ce programme sur le problème du voyageur de commerce et le problème de sac à dos.

Chapitre 3

Etude de cas : Application de la méthode du recuit simulé sur le TSP et le KP

3.1 Introduction

ce chapitre est consacré à l'application de la méthode de recuit simulé sur deux problèmes célèbres d'optimisation combinatoire à savoir le problème du voyageur de commerce (TSP) et le problème de sac à dos. Pour pouvoir juger mieux l'efficacité de l'algorithme un environnement adéquat a été établi où plusieurs et différentes façons de générer les données ont été effectuées pour les deux problèmes

3.2 Programmation et implémentation

La phase implémentation se caractérise généralement par un ensemble des choix faits par le concepteur ou le programmeur afin de répondre à un ensemble des critères liés à la méthode proposée, dans notre cas on va spécifier le langage de programmation ; par conséquent on va désigner l'outil ou la plateforme de programmation et d'exécution adaptée au langage choisi. Autres choix liés à la méthode et qui doivent être justifiés qui est la structure de données supportant une solution, supportant la mémoire locale et la mémoire globale, ainsi que le choix de type de voisinage d'une solution.

3.2.1 Le choix du langage de programmation

Il existe plusieurs langages de programmation pour implémenter les algorithmes mathématiques, mais pour des raisons de portabilité la méthode a été implémentée avec Fortran 90. La portabilité de Fortran 90 lui donne un bon avantage pour être un langage d'implémentation pour les applications sur les algorithmes mathématiques. Le langage Fortran 90 offre plusieurs mécanismes qui permettent d'une implémentation programme donné d'être exploité, et facilite la

manipulation des structures (objet de type dérivé), en offrant les possibilités suivantes :

- Il permet de définir des fonctions de manipulation de ces structures (surcharge des opérateurs $+$, $-$, $*$, de nouveaux opérateurs en les associant aux fonctions correspondantes, autres fonctions ou procédures agissant sur ces structures).
- Il permet d'encapsuler le type et les fonctions opérant sur les objets de ce type dans un module séparé. Ce qui permettra d'y accéder facilement (USE <module>) en diminuant les sources d'erreurs, dans toutes les unités de programme en ayant besoin.

3.2.2 Les structures des données

Pour la structure de données et pour des raisons de simplicité on a fait les choix suivants :

- Dans le cas du problème du voyageur de commerce comme suit :
 - Chaque solution est représentée à l'aide d'un tableau de la taille $n+1$ où n est le nombre des villes.
 - La matrice de la distances entre les villes est représentée à l'aide d'une matrice carrée et symétrique de dimension $n \times n$ où n est le nombre des villes.
- Dans le cas du problème de sac à dos comme suit :
 - Les profits et les poids de les objets sont représentés à l'aide d'une matrice de n colonne et deux lignes où n est le nombre d'objets.
 - Les objets sélectionnés représentés à l'aide d'un tableau de taille n d'éléments binaires, n est le nombre d'objets.

3.2.3 Le choix de voisinage

Pour l'implémentation de cette méthode on utilise le voisinage par permutation dans le cas du problème du voyageur de commerce. Donc à chaque itération, on génère un ensemble de voisinage par insertion de la solution courante par la permutation d'une villes avec chacune des autres de cette solution, ensuite on fait le choix de la meilleure solution pour qu'elle soit la solution courante de la prochaine itération de la recuit. On note que si on considère que n est la taille de donnée (dans notre cas le nombre de villes) l'ensemble de voisinage par insertion contient $n * (n - 1)/2$ solution voisine.

Exemple 3.2.1. *On considère un problème de voyageur de commerce de quatre ville 1, 2, 3 et 4. supposons qu'on a une tournée $s = (1, 2, 3, 4)$ comme solution courante. Alors l'ensemble des solutions voisines est représenté comme suit :*

$$V(1, 2, 3, 4) = \{(2, 1, 3, 4); (3, 2, 1, 4); (4, 2, 3, 1); (1, 3, 2, 4); (1, 4, 3, 2); (1, 2, 4, 3)\}$$

Dans le cas du problème de sac à dos on utilise un voisinage inspiré par la distance Hamming, on génère un ensemble des solutions voisines de la solution courante, tel que la distance Hamming entre les éléments de ce ensemble avec cette solution est égale à 1. On note que si on considère que n est la taille de donnée (dans notre cas le nombre de objets) l'ensemble de ce voisinage contient n solutions voisines.

Exemple 3.2.2. *On considère un problème de sac à dos de cinq objets 1, 2, 3, 4 et 5. supposons qu'on a un sélection $s = (0, 1, 1, 0, 1)$ comme solution courante. Alors l'ensemble des solutions voisines est représenté comme suit :*
 $V(0, 1, 1, 0, 1) = \{(1, 1, 1, 0, 1); (0, 1, 1, 1, 1); (0, 0, 1, 0, 1); (0, 1, 0, 0, 1); (0, 1, 1, 0, 0)\}$

3.2.4 Phase expérimentale

L'implémentation de l'application est fait avec l'outil Code : :Blocks 16.01, et l'exécution sur une plateforme logicielle Windows 8.1, sur une plateforme matérielle d'un ordinateur de configuration suivante :

- Processeur : Intel(R) core(TM) i5-3230M CPU @ 2.60GHz 2.60GHz.
- RAM : 4Go.

3.2.5 Tests et résultats

3.2.5.1 Le cas du problème du voyageur de commerce

Afin d'être plus juste de comparer les résultats en utilisant le programme RECUIT, nous avons générer 100 problèmes pour différents paramètres tel que $n = \{200, 300, 500, \text{ et } 600\}$ et les distances sont générés comme suit :

- Dans le premier problème la distance appartient l'intervalle [1.100].
- Dans le deuxième problème la distance appartient l'intervalle [20.100].
- Dans le troisième problème la distance appartient l'intervalle [30.100].
- Dans le quatrième problème la distance appartient l'intervalle [40.100].
- Dans le cinquième problème la distance appartient l'intervalle [50.100].

Le nombre d'itérations est compatible avec la température initiale comme suit :

- Si la température initial $T = 100$ nombre d'itérations est égale à 76.
- Si la température initial $T = 500$ nombre d'itérations est égale à 92.
- Si la température initial $T = 1000$ nombre d'itérations est égale à 98.
- Si la température initial $T = 5000$ nombre d'itérations est égale à 114.
- Si la température initial $T = 10000$ nombre d'itérations est égale à 120.

Le coefficient α de réduction de la température est égal à 0.9, et la température finale égale à 0.03.

- Test 1 : pour nombre de villes $n = 200$, la taille de l'ensemble de voisinage est égale à 19900 solution voisine.
Le résultat d'exécution de programme Classé dans le tableau suivant :

L'intervalle de distance	La température initiale	la distance minimale	temps d'exécution (s)
	100	2034.18	85,91
[01-100]	500	1771.68	106,72
	1000	1702.06	110,88
	5000	1673.58	129,37
	10000	1673.58	139,41
	100	5467,34	79,29
[20-100]	500	5257,34	95,77
	1000	5201,65	102,01
	5000	5178.86	118,51
	10000	5178.86	125,14
	100	7283,92	79,91
[30-100]	500	7100,18	96,01
	1000	7051,44	101,91
	5000	7031,51	118,41
	10000	7031,51	125,24
	100	9100,51	79,23
[40-100]	500	8943,01	95,74
	1000	8901,23	101,93
	5000	8884,14	118,45
	10000	8884,14	125,67
	100	10917,09	79,33
[50-100]	500	10785,84	95,78
	1000	10753.14	101,91
	5000	10736,79	118,51
	10000	10736,79	125,24

TABLE 3.1 – Le résultat d'exécution pour 200 villes.

- Test 2 : pour nombre de villes $n = 300$, la taille de l'ensemble de voisinage est égale à 44850 solution voisine.

Le résultat d'exécution de programme Classé dans le tableau suivant :

L'intervalle de distance	la température initiale	la distance minimale	temps d'exécution (s)
	100	3667.09	434,41
[01-100]	500	3115.80	522,91
	1000	2989.67	565,45
	5000	2699.36	647,11
	10000	2624.71	861,22
	100	8693,68	428,78
[20-100]	500	8252,64	516,01
	1000	8151,74	550,41
	5000	7919,49	646,03
	10000	7859,77	683,31
	100	11356,96	490,91
[30-100]	500	10971,06	550,32
	1000	10822,76	573,17
	5000	10679,54	651,06
	10000	10627,31	716,94
	100	14020,25	429,82
[40-100]	500	13689,48	518,29
	1000	13613,39	589,06
	5000	13439,61	645,68
	10000	13394,83	778,18
	100	16683,54	478,49
[50-100]	500	16407,9	534,25
	1000	16344,83	565,69
	5000	16199,67	645,07
	10000	16162,35	702,41

TABLE 3.2 – Le résultat d'exécution pour 300 villes.

- Test 3 : pour nombre de villes $n = 500$, la taille de l'ensemble de voisinage est égale à 124750 solution voisine.

Le résultat d'exécution de programme Classé dans le tableau suivant :

L'intervalle de distance	la température initiale	la distance minimale	Temps d'exécution (s)
	100	9771.77	3294,82
[01-100]	500	8153.41	4260,02
	1000	7674.67	4234,22
	5000	6627.67	4953,8
	10000	6300.95	5554,70
	100	17417.43	3220,72
[20-100]	500	16122.73	4230,45
	1000	15739,74	4351,70
	5000	14902,14	5048,40
	10000	14640,76	5316,50
	100	21490,24	3344,22
[30-100]	500	20357,38	4026,83
	1000	20022.26	4370,60
	5000	19289,36	5045,79
	10000	19060,65	5327,11
	100	25563,06	3472,99
[40-100]	500	24592,04	4145,02
	1000	24304,81	4366,19
	5000	23676,61	4924,41
	10000	23480,58	5114,02
	100	29635,87	3442,56
[50-100]	500	28826,72	3932,08
	1000	28587,33	4178,95
	5000	28063,23	4806,73
	10000	27900,47	5150,58

TABLE 3.3 – Le résultat d'exécution pour 500 villes.

- Test 4 : pour nombre de villes $n = 600$, la taille de voisinage est égale à 179700 solution voisine.
Le résultat d'exécution de programme Classé dans le tableau suivant :

L'intervalle de distance	la température initiale	la distance minimale	Temps d'exécution (s)
	100	14062.08	7494,43
[01-100]	500	11754.87	8375,57
	1000	11026.01	12648,33
	5000	9372.59	12120,50
	10000	8892.61	12920,04
	100	22769,66	6981,23
[20-100]	500	20923,88	8389,98
	1000	20340,79	8851,03
	5000	19018,07	10228,71
	10000	18634,07	10635,97
	100	27423,44	6855,41
[30-100]	500	25808,41	8313,49
	1000	25298,19	8756,86
	5000	24140,81	10142,70
	10000	23804,80	10740,91
	100	32077,25	6880,87
[40-100]	500	30692,92	8299,95
	1000	30255,61	8729,15
	5000	29263,56	10130,44
	10000	28975,56	10580,68
	100	36731,03	6801,43
[50-100]	500	35577,42	8202,80
	1000	35213,01	8610,12
	5000	34386,29	10144,11
	10000	34146,29	10530,39

TABLE 3.4 – Le résultat d'exécution pour 600 villes.

3.2.5.2 Le cas du problème de sac à dos

Afin d'être plus juste de comparer les résultats en utilisant le programme RECUIT, nous avons générer 25 problèmes pour différents paramètres tel que $n = \{10000, 10200, 10300, 10400 \text{ et } 10650\}$ et les capacités maximaux de sac sont générés comme suit :

- Dans le premier problème la capacité maximale de sac est égale à 5000.
- Dans le deuxième problème la capacité maximale de sac est égale à 20000.
- Dans le troisième problème la capacité maximale de sac est égale à 30000.
- Dans le quatrième problème la capacité maximale de sac est égale à 40000.
- Dans le cinquième problème la capacité maximale de sac est égale à 50000.

Le nombre d'itérations est compatible avec Le coefficient α de réduction de la température comme suit :

- Si Le coefficient $\alpha = 0.75$ nombre d'itérations est égale à 36.
- Si Le coefficient $\alpha = 0.80$ nombre d'itérations est égale à 46.
- Si Le coefficient $\alpha = 0.85$ nombre d'itérations est égale à 64.
- Si Le coefficient $\alpha = 0.90$ nombre d'itérations est égale à 98.
- Si Le coefficient $\alpha = 0.95$ nombre d'itérations est égale à 203.

la température initiale est égale à 1000, et la température finale est égale à 0.03.

Dans les tableaux suivants, Wmax représente le poids des objets sélectionnés, et Pmax représente la valeur de leur importance.

- Test 1 : pour nombre d'objets $n = 10000$, le résultat d'exécution de programme Classé dans le tableau suivant :

α	Pmax	Wmax	Temps d'exécution (s)
0,75	3762	1862	295,78
0.80	4752	2352	352,37
0,85	6534	3234	512,43
0,90	9900	4900	772,96
0,95	14098	4998	1614,28

TABLE 3.5 – Le résultat d'exécution pour 10000 objets.

- Test 2 : pour nombre d'objets $n = 10200$, le résultat d'exécution de programme Classé dans le tableau suivant :

α	Pmax	Wmax	Temps d'exécution (s)
0,75	5662	2812	296,88
0,80	7152	3552	406,97
0,85	9834	4884	534,11
0,90	14877	7400	866,57
0,95	20057	9990	1727,17

TABLE 3.6 – Le résultat d'exécution pour 10200 objets.

- Test 3 : pour nombre d'objets $n = 10300$, le résultat d'exécution de programme Classé dans le tableau suivant :

α	Pmax	Wmax	Temps d'exécution (s)
0,75	11362	7562	365,32
0.80	14344	9549	471,97
0,85	19708	13122	649,39
0,90	29818	19857	946,94
0,95	44939	29929	1866,91

TABLE 3.7 – Le résultat d'exécution pour 10300 objets.

- Test 4 : pour nombre d'objets $n = 10400$, le résultat d'exécution de programme Classé dans le tableau suivant :

α	Pmax	Wmax	Temps d'exécution (s)
0,75	13258	9460	410.22
0.80	16738	11946	462.43
0,85	23000	16416	635.392
0,90	34798	24841	1071.27
0,95	58203	39927	1965.28

TABLE 3.8 – Le résultat d'exécution pour 10400 objets.

- Test 5 : pour nombre d'objets $n = 10650$, le résultat d'exécution de programme Classé dans le tableau suivant :

α	Pmax	Wmax	Temps d'exécution (s)
0,75	13258	11359	378.34
0.80	16738	14341	485.09
0,85	23000	19707	667.76
0,90	34798	29816	1002.15
0,95	58303	49961	1942.70

TABLE 3.9 – Le résultat d'exécution pour 10650 objets.

3.2.5.3 Discussions

Dans le cas du problème du voyageur de commerce, les quatre tests d'implémentation, on remarque qu'à chaque fois qu'on diminué la température initiale on obtient une solution meilleure, malheureusement ceci est au dépend de l'augmentation du temps d'exécution. Bien sur le temps d'exécution augmente chaque fois que la nombre des villes augmente. Dans le cas de sac à dos, les cinq tests d'implémentation, on remarque qu'à chaque fois qu'on diminué Le coefficient α de réduction de la température on obtient une solution meilleure, malheureusement ceci est au dépend de l'augmentation du temps d'exécution. Bien sur le temps d'exécution augmente chaque fois que la nombre des objets augmente.

3.3 Conclusion

Dans ce chapitre on a implémenté une programme de la méthode du recuit simulé, l'implémentation était réalisée sur le problème du voyageur de commerce, utilisant le langage Fortran 90. Les résultats montrent que la méthode du recuit simulé converge donne une solution optimale, ce dernier en relation avec la solution et la température initiaux.

Conclusion générale

Nous sommes intéressés dans ce mémoire au la méthode de recuit simulé pour des problèmes d'optimisation académiques difficiles. Notre objectif est appliquer la méthodes pour la résolution de ce type de problèmes. Afin de réaliser notre but, nous avons organisé nos recherches en deux phases. Dans la première phase, nous nous sommes concentrés sur l'établissement d'un état de l'art sur les problèmes d'optimisation et les méthodes de recherche locale et recuit simulé proposées dans la littérature. Nous avons essayé de récolter des notions de bases en optimisation, comprendre le rôle de la théorie de la complexité qui permet d'évaluer la complexité de problèmes et les classer en problèmes P ou NP. Nous avons ensuite essayé d'aborder quelques problèmes d'optimisation académiques difficiles à résoudre. Ensuite, nous avons établi une recherche sur les méthodes (la recherche locale et recuit simulé) : leurs origines, leurs principes, leurs étapes, leurs avantages, leurs lacunes et leurs applications. Dans la deuxième phase, appliquer l'algorithme de recuit simulé sur le problème du voyageur de commerce symétrique et le problème de sac à dos, nous avons programmé ce algorithme en la langage de programmation (Fortran 90) et implémenter ce programme et sauvegarder les résultats dans des tableaux, ces derniers prouvent que la méthode de recuit simulé efficace tel que donnée une solution optimale approchée et applicable à plusieurs problèmes d'optimisation académiques. Il est recommandé de chaque fois le schéma de décroissance de la température afin de voir s'il y à une possibilité d'améliorer la performance de notre algorithme. Aussi il faut penser à utiliser d'autre style de voisinage tels que la méthode de 2 ou 3 opt et finir une étude comparative pour pouvoir déterminer le meilleure algorithme du recuit simulé que performe le mieux.

Bibliographie

- [1] S. Cook. The Complexity of Theorem-Proving Procedure. Proc. 3 rd ACP Symp. On Theory of Complexity, Association of Computing Machinery. New York. pp. 151 - 158, 1971.
- [2] A. Gherboudj. Méthodes de résolution de problèmes difficiles académiques. Thèse de doctorat. Université de Constantine2. Algérie. 2013
- [3] S. Bourazza. Variantes d'algorithmes génétiques appliquées aux problèmes d'ordonnancement. Thèse de doctorat. Université du Havre, France. 2006.
- [4] P. Pongchairerks. Particle swarm optimization algorithm applied to scheduling problems. ScienceAsia. Vol. 35, pp. 89-94, 2009.
- [5] D. Pisinger. Where are the hard knapsack problems? Computers and Operations Research. Vol. 32, pp. 2271-2284, 2005.
- [6] G. B. Dantzig, J. H Ramser. The Truck dispatching problem. Lehrstuhl für Wirtschaftsinformatik und Operations Research. Universitat zu Koln, Mgmt Sci. Vol. 6, pp 80-91, 1959.
- [7] M.W.P. Savelsbergh. Local Search for Routing Problems with Time Windows. Annals of Operations Research. Vol. 4, pp. 285-305. 1995.
- [8] R. Kammarti. Approches Evolutionnistes Pour La Resolution Du 1 - PDPTW Statique Et Dynamique. Thèse de doctorat. Université de Lille, France. 2006.
- [9] A. Schrijver. On the history of combinatorial optimization, 1960.
- [10] R. M. Garey et D. S. Johnson. Computers and Intractability. A guide to the Theory of NP-Completeness. W.H.Freeman and CO, San Francisco. 1979.
- [11] E. A. Feigenbaum, j. Feldman, (Edirors). Computers and thought. McGraw-Hill Inc. pp.6. New York, 1963.

-
- [12] I.H. Osman, G. Laporte. Metaheuristics : A bibliography. Ann.Oper. Res. Vol. 63, pp. 513-623, 1996.
 - [13] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Simulated annealing : theory and applications, IBM Research Report RC. pp. 9355, 1982.
 - [14] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E.Teller. Equation of state calculations by fast computing machines. Journal of Chem. Physics. Vol. 21, pp. 1087-1092, 1953.
 - [15] R.J. Gauber, Time-dependent statistics of the Ising Model, J. Math. Phys. Vol. 4, pp. 294, 1963
 - [16] G. Athier, P. Floquet, L. Pibouleau, S. Domenech, Synthesis of optimum heat exchanger networks by simulated annealing, AIChE J. Vol 11, pp. 3007–3020, 1997.
 - [17] E.H.L. Aarts, P.J. Van Laarhoven, Statistical cooling : a general approach to combinatorial optimization problems, Philips J. Res. vol. 40, pp. 193, 1985.
 - [18] P. Siarry, J. Dréo, A. Pétrowski, E. Taillard. Métaheuristiques pour l'optimisation difficile. Eyrolles, pp. 40, 2011.