Université de M'sila

FACULTÉ DES MATHÉMATIQUES ET INFORMATIQUES

Département de Mathématiques

Mémoire de fin d'etude

Présenté pour l'obtention du diplôme de **Master**

Domaine :Mathématiques et Informatique

Filière :Mathématiques

Spécialité : Mathématiques Discrètes

Par

Hanane Souyah

Sujet

Problèmes d'optimisation combinatoire et métaheuristiques à base solution unique

Président

Belouadah Hocine

Promotion: 2014/2015

Remerciements

Je remercie tout d'abord mon Dieu qui m'a donné la force pour terminer ce modeste travail.

Je tiens à remercier mes promoteurs : Mr le professeur Belouadah Hocine pour la confiance qu'il m'a témoignée en me proposant ce sujet, ses encouragements et sa patience.

Je remercie l'étudiant Karima seghiri qui ma aussi aider dans ce travail.

Je remercie aussi tous les membres du Jury pour l'honneur qu'ils m'ont fait, en acceptant de juger ce modeste travail.

Je ne peux pas clôturer mes remerciements sans se retourner vers les êtres qui me sont les plus chers ; ma famille qui ont eu un rôle essentiel et continu dans ma réussite.

Merci

Table des matières

Introduction

L'optimisation combinatoire occupe une place très importante en recherche opérationnelle, en mathématiques discrètes et en informatique. Son importance réside dans la difficulté des problèmes d'optimisation lorsque leurs tailles sont grandes.

Un problème d'optimisation comme un problème de recherche qui consiste à explorer un espace contenant l'ensemble de toutes les solutions potentielles réalisables, dans le but est de trouver la solution optimale, sinon la plus proche possible de l'optimum, permettant de minimiser ou maximiser une fonction dite objectif. Nous avons donné quelques problèmes d'optimisation: le problème d'ordonnancement, le problème du sac à dos, le problème du coloration de graphe et le problème du voyageur de commerce.

Il existe au moins deux méthodes de résolution des problèmes d'optimisation combinatoire. Les méthodes exactes qui permettent d'obtenir une solution optimale exacte, mais avec un temps de calcul très long et les méthodes approchées appelées heuristiques qui permettent quant à elles d'obtenir rapidement une solution approchée.

Dans notre mémoire on s'est penché sur ces métaheuristiques à base solution unique qui est un partie de méthodes approchées présentées pour résoudre des problèmes combinatoires, Elle possède une grande efficacité de fournir des solutions approchées de bonne qualité pour un grand nombre de problèmes avec un temps de calcul raisonnable. Parmi ces métaheuristiques nous pouvons citer les méthodes: Descente, Descente Stochastique, Recuit Simulé et Tabou.

Ce mémoire est composé en trois chapitres:

Dans le premier chapitre on va définir l'optimisation combinatoire, le problème d'optimisation, le voisinage, la théorie de la complexité, et on va cité quelques problèmes d'optimisation ainsi que les méthodes de résolution.

Dans le deuxième chapitre on va définir quatre métaheuristique à base solution unique (méthode Descente, méthode Descente Stochastique, méthode de Recuit Simulé et méthode de Tabou). On applique chacune à un exemple illustratif pour éclaircir leur mécanismes.

En fin, dans le troisième chapitre on choisi parmi les métaheuristique étudiées méthode Descente Stochastique. On l'applique sur le problème du voyageur de commerce pour déterminer une solution approchée. L'algorithme générale est programmer en utilisant le langage Java pour une étude empirique pour juger la performance de la métaheuristique.

Chapitre 1

L'optimisation combinatoire

1.1 Introduction

L'optimisation combinatoire est un domaine très important en mathématiques discrètes. 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 dites $NP - difficiles$ et ne possèdent donc pas à ce jour de solution algorithmique efficace valable pour toutes les données.

Nous présentons dans ce chapitre l'optimisation combinatoire, problème d'optimisation, le voisinage, la théorie de la complexité, quelque problèmes d'optimisation et la méthode de résolution d'un problème.

1.2 Notion d'optimisation combinatoire

" **Combinatoire** " désigne la discipline des mathématiques concernée par les structures " discrètes " ou " finies ". Citons quelques branches de cette discipline: la théorie des graphes, la combinatoire énumérative, les problèmes de dénombrement, la combinatoire polyèdrale,

l'optimisation combinatoire, etc. Les frontières entre ces branches ne sont pas hermétiques, les différentes branches expriment plutôt des orientations méthodologiques différentes.

L'" **optimisation** ", ou " **programmation mathématique** " sont des termes utilisés pour recouvrir toutes les méthodes qui servent à déterminer l'optimum d'une fonction avec (ou sans) contraintes. Cette fonction modélise le choix optimal. On optimise déjà à l'école, quand on apprend comment déterminer l'optimum (minimum ou maximum) d'une fonction différentiable. Les ingénieurs, les économistes, la nature, font souvent des choix optimaux (ou quasi-optimaux), d'où l'importance de l'optimisation dans les sciences, qu'elles soient techniques, mathématiques, physiques, informatiques, économiques, naturelles, etc.

L' " **optimisation combinatoire** " consiste à trouver le meilleur entre un nombre fini de choix. Autrement dit, minimiser une fonction, avec contraintes, sur un ensemble fini. Quand le nombre de choix est exponentiel (par rapport à la taille du problème), mais que les choix et les contraintes sont bien structurés, des méthodes mathématiques doivent et peuvent intervenir, comme dans le cas " continu ", pour permettre de trouver la solution en temps polynomial (par rapport à la taille du problème). Alors, l'optimisation combinatoire consiste à développer des algorithmes polynomiaux et des théorèmes sur des structures discrètes qui permettent de résoudre ces problèmes [19].

1.3 Problème d'optimisation

Un problème d'optimisation combinatoire est défini par un ensemble d'instances. A chaque instance du problème est associé un ensemble discrète de solutions S, un sous-ensemble X de S représentant les solutions admissibles (réalisables) et une fonction coût f (ou fonction objectif) qui assigne à chaque solution $s \in X$ le nombre réel (ou entier) $f(s)$. Résoudre un tel problème (plus précisément une telle instance du problème) consiste à trouver une solution $s^* \in X$ optimisant la valeur de la fonction coût f. Une telle solution s^* s'appelle une *solution optimale* ou un *optimum global*.

Nous avons donc la définition suivante:

Définition 1.3.1

Une instance I d'un problème de minimisation est un couple (X, f) où $X \subseteq S$ est un ensemble fini de solutions admissibles, et $f : X \rightarrow \mathbb{R}$ une fonction coût (ou objectif) à minimiser. Le problème est de trouver $s^* \epsilon X$ tel que $f(s^*) \leq f(s)$ pour tout élément $s \epsilon X$. Notons que d'une manière similaire, on peut également définir les problèmes de maximisation en remplaçant simplement $\leq$ par $\geq$. L'optimisation combinatoire trouve des applications dans des domaines aussi variés tels que la gestion, l'ingénierie, la conception, la production, les télécommunications, les transports, l'énergie, les sciences sociales et l'informatique [9].

Remarque 1.3.1 *puisque dans cet mémoire nous allons nous concentré sur l'étude des méthodes approchées telles que les métaheuristiques pour déterminer des solutions locales nous allons définir la notion de voisinage qui est liée directement au fonctionnement de ces méthodes.*

1.4 Définition de voisinage

Le voisinage de s est un sous-ensemble de configurations de X, directement atteignables à partir d'une transformation donnée de s. Il est noté $V(s)$ et une solution $\acute{s} \in V(s)$ 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 l'on s'autorise, celle-ci étant définie comme une application $V : X \rightarrow V(X)$. Chaque structure de voisinage fournit un voisinage, ou un ensemble de solutions, précis via la transformation définie [14].

1.4.1 Génération de voisinage

Pour le problème l'ordonnancement il y'a

- Le voisinage par permutation de deux tâches (adjacentes - au pas)
- Le voisinage par insertion des tâches

Pour le problème du voyageur de commerce il y'a

- Le voisinage par $2-Opt$
- Le voisinage par $3-Opt$
- Le voisinage par $\lambda-Opt$

1.4.2 Optimum local

Une solution $\acute{s} \in X$ est un optimum local si:

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

avec $V(s)$ 1'ensemble des solutions voisines de s.

1.4.3 Optimum global

Une solution $\acute{s} \in X$ est un optimum global si:

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

1.5 La théorie de la complexité

La théorie de la complexité s'intéresse à l'évaluation de la difficulté des problèmes via l'étude de la complexité de solutions algorithmiques proposées. Elle associe une fonction de complexité à chaque algorithme résolvant un problème donné et mesure les ressources nécessaires pour la résolution d'un problème posé. En effet, elle classe l'ensemble des problèmes selon deux critères qui sont le temps de calcul nécessaire et l'espace mémoire requis. Bien que la théorie de la complexité se concentre sur des problèmes de décision, elle peut être étendue aux problèmes d'optimisations. Elle classe les problèmes selon leur complexité en deux classes principales: la classe P et la classe NP. En outre, elle partage les problèmes de la classe NP en deux sous classes: $NP-complet$ et $NP-difficile$ [6].

1.5.1 La classe P

P est la classe de tous les problèmes de décision qui peuvent être résolus par un algorithme polynomial (Un problème de décision est un problème dont la réponse est oui ou non).

1.5.2 La classe NP

Un problème de décision est dans la classe NP si on peut vérifier en temps polynomial qu'une solution proposée permet d'affirmer que la réponse est «oui» pour cette instance.

1.5.3 Le problème NP complet

Un problème est dit $NP-complet$ s'il est dans classe NP, et que si problème de NP peut se réduire polynomialement à lui.

1.5.4 Le problème NP difficile

Un problème $NP-difficile$ est un problème au moins aussi difficile qu'un problème $NP-complet$, si le problème de décision associé à un problème d'optimisation est $NP-complet$, alors le problème d'optimisation est $NP-difficile$.

1.5.5 Le problème ouvert

Un problème est ouvert si on n'a pas pu lui déterminer un algorithme polynomial pour le résoudre et où n'a pas pu démontrer qu'il est équivalent à un problème $NP-difficile$.

1.5.6 Est-ce que P = NP ?

Depuis le début de l'introduction, j'utilise les mots « difficile » ou « facile » pour décrire des problèmes informatiques. La classification des problèmes en ces deux catégories n'est pas arbitraire: elle correspond à des critères très précis. On dit qu'un problème est dans la classe P (facile) si on peut le résoudre en temps polynomial. Il suffit par contre qu'on sache vérifier la solution d'un problème pour qu'il soit dans NP (difficile). Bien entendu, si un

problème est dans P, il est également dans NP: si je dois vérifier une solution d'un problème de P, je peux commencer par trouver la solution, et je compare les deux. Inversement, on peut se poser la question de savoir si les problèmes de NP sont également dans P. Ce problème est ouvert depuis 1971, il est devenu en 2001 un des 7 problèmes du millénaire, et est donc mis à prix pour un million de dollars [15].

1.5.7 L'optimisation difficile

L' « **optimisation combinatoire** » consiste à trouver la meilleure solution entre un nombre fini de choix. Autrement dit, à minimiser une fonction, avec ou sans contraintes, sur un ensemble fini de possibilités. Quand le nombre de combinaisons possibles devient exponentiel par rapport à la taille du problème, le temps de calcul devient rapidement critique.

Ce temps de calcul devient si problématique que pour certains problèmes, on ne connaît pas d'algorithme exact polynomial, c'est-à-dire dont le temps de calcul soit proportionnel à N^n, où N désigne le nombre de paramètres inconnus du problème, et où n est une constante entière. Lorsqu'on conjecture qu'il n'existe pas une telle constante n telle qu'un polynôme de degré n puisse borner le temps de calcul d'un algorithme, on parle alors d'optimisation difficile, ou de problèmes $NP - difficiles$ (c'est tout l'objet de la théorie de la $NP-$ *complétude*) [1].

1.6 Quelques problèmes d'optimisation combinatoire

Nous présentons quatre problèmes classiques d'optimisation combinatoire: le problème d'ordonnancement, le problème du sac à dos, le problème du coloration de graphe, le problème du voyageur de commerce.

1.6.1 Le problème d'ordonnancement

Le problème d'ordonnancement consiste à organiser dans le temps la réalisation de tâche, compte tenu de contraintes temporelles (délais, contraintes d'enchainement,...) et de contraintes portant sur l'utilisation et la disponibilité des ressources requises.

Un ordonnancement constitue une solution au problème d'ordonnancement. Il décrit l'exécution des tâches et l'allocation des ressources au cours du temps et vise à satisfaire un ou plusieurs objectifs. De manière plus précise, on parle d'ordonnancement lorsqu'on fixe les dates de début ou de fin de chacune des tâches, alors qu'on réserve le terme de séquencement au cas ou seul l'ordre relatif des tâches (séquence) est fixé, indépendamment des dates d'exécution [5].

1.6.2 Le problème du sac à dos

Le problème du sac à dos (**Knapsack Problem**, KP) est un problème classique d'optimisation combinatoire appartenant à la classe des problèmes $NP - difficiles$ [17]. L'énoncé de ce problème est simple: étant donné un ensemble de n objets, où chaque objet i est caractérisé par un poids w_i et un profit p_i on cherche le sous-ensemble d'objets à charger dans un sac de capacité c afin de maximiser la somme des profits. Ainsi, le problème du sac à dos se présente sous la forme mathématique suivante:

$$(KP) \begin{cases} \max \sum_{i=1}^{n} p_i \cdot x_i, \\ \sum_{i=1}^{n} w_i \cdot x_i \leq c, \\ x_i \in \{0,..,\, n\} \end{cases} \qquad (II.2)$$

Les poids w_i et les profits p_i ainsi que la capacité c sont des entiers positifs $i \in \{1, ..., n\}$. La variable x_i est la variable de décision; elle prend la valeur 1 si l'objet i est chargé dans le sac, sinon elle prend la valeur 0. Résoudre ainsi le problème du sac à dos revient à trouver le vecteur solution $x^* = (x_1^*, ..., x_n^*)^T$ qui optimise (maximise dans ce cas) la fonction objectif définie en $(II.2)$ [18].

1.6.3 Le problème du coloration de graphe

Etant donné un graphe non orienté, colorer ses sommets de façon à ce que deux sommets adjacents soient de couleur différents. Si k couleurs sont nécessaires, on parle d'une k-coloration. Le nombre chromatique du graphe est le k minimum pour lequel il admet une k coloration, en général trouver le nombre chromatique est un problème $NP - complet$ [18].

1.6.4 Le problème du voyageur de commerce

Le problème du voyageur de commerce (**Traveling Salesman Problem,** TSP) doit visiter n villes données en passant par chaque ville exactement une fois. Il commence par une ville quelconque et termine en retournant à la ville de départ. Les distances entre les villes sont connues. Il faut trouver le chemin qui minimiser la distance parcourue. La notion de distance peut-être remplacée par d'autres comme le temps qu'il met ou l'argent qu'il dépense: dans tous les cas, on parle de coût [12].

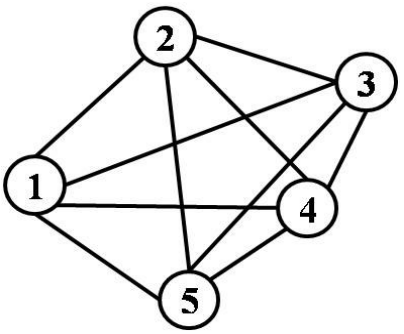

Figure 1.1. Problème du voyageur de commerce

Remarque 1.6.1

- On dit que le problème du voyageur de commerce est symétrique si $d_{ij} = d_{ji}$.
- On dit que le problème du voyageur de commerce est asymétrique si $d_{ij} \neq d_{ji}$.

1.7 Les méthodes de résolution d'un problème d'optimisation

Il existe deux méthodes pour de résolution le problème d'optimisation: méthode exacte et méthode approchées, la figure **1.2**. met en parallèle les méthodes représentatives développées en recherche opérationnelle

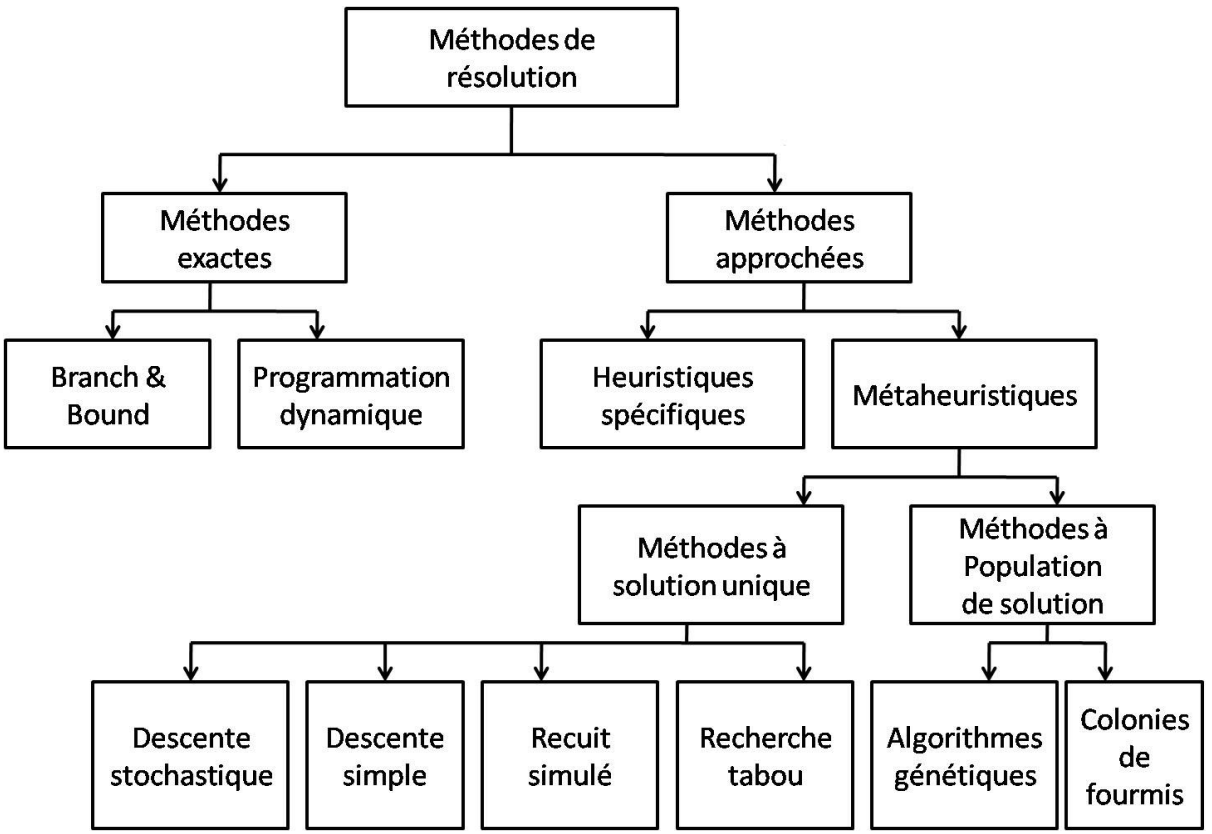

Figure 1.2. Méthodes de résolution le problème d'optimisation

1.7.1 Les méthodes exactes

les méthodes exactes sont très connues par le fait qu'elles nécessitent un coût de recherche souvent prohibitif en termes de ressources requises. En effet, le temps de recherche et/ou l'espace mémoire nécessaire pour l'obtention de la solution optimale par une méthode exacte sont souvent trop grands, notamment avec des problèmes de grandes tailles. De ce fait, la complexité de ce type d'algorithme croit exponentiellement avec la taille de l'instance à traiter, elle devient très importante face à des problèmes comprenant plusieurs variables, fonctions objectifs et/ou critères.

Il existe de nombreuses algorithmes exactes y compris la programmation dynamique, les algorithmes de séparation et évaluation (Branch and Bound) ... etc [7].

1.7.2 Les méthodes approchées

Une méthode approchée ou heuristique est un algorithme d'optimisation qui a pour but de trouver une solution raisonnable de la fonction objectif, mais sans garantie d'optimalité. Le principal avantage de ces méthodes est qu'elles peuvent s'appliquer à n'importe quelle classe de problèmes, faciles ou très difficiles, bien ou mal formulés, avec ou sans contrainte. En particulier, elles ne nécessitent pas une modélisation mathématique du problème. Elles semblent être tout à fait adaptées à l'optimisation de systèmes de production et c'est donc ce type de méthode qui sera utilisé dans la suite de ce travail. Ces méthodes sont souvent classées en deux catégories: des méthodes heuristiques et des méthodes métaheuristiques [9].

Les heuristiques

Une heuristique (méthode heuristique) est un algorithme qui fournit rapidement en temps polynomial une solution réalisable, pas nécessairement optimale, pour un problème d'optimisation $NP - difficile$ [11].

Les métaheuristiques

Les méthodes dites métaheuristiques sont des méthodes générales, des heuristiques polyvalentes applicables sur une grande gamme de problèmes. Elles peuvent construire une alternative aux méthodes heuristiques lorsqu'on ne connaît pas l'heuristique spécifique à un problème donné [7].

1.8 Conclusion

Nous avons défini dans ce chapitre les problèmes d'optimisation combinatoire ainsi que nous avons présenté quelques exemples ($NP - complet$), il existe de nombreuse méthodes de résolution. Chacune a ses avantages et ses inconvénients.

La résolution du problème d'optimisation combinatoire par une méthode exacte est pratiquement difficile et surtout pour le problème de grande taille. Pour cela, nous allons opté

pour les métaheuristiques qui servent à déterminer des solutions locales mais avec un coût (temps d'exécution, espace mémoire) minimiser.

Chapitre 2

Les métaheuristiques à base solution unique

2.1 Introduction

Les métaheuristiques, sont un paradigme très important des méthodes approchées destinée à la résolution des problèmes combinatoires. Grâce à elle, on peut proposer aujourd'hui de bonnes solutions pour le problèmes de plus grande taille, et pour de très nombreuses applications qu'il était impossible de traiter auparavant.

Nous présentons dans ce chapitre les principes fondamentaux de quatre métaheuristiques qui constituent l'objet de ce mémoire: Descente, Descente Stochastique, Recuit Simulé et Tabou. Nous donnons d'abord, un aperçu général sur les métaheuristiques. Ensuite, nous évoquons dans les quatre sections qui suivent, les notions fondamentales relatives aux quatre méthodes retenues en précisant notamment les principes généraux et le fonctionnement de chacune.

2.2 Présentation des métaheuristiques

Les problèmes d'optimisation comme le problème du voyageur de commerce, sont pour la majorité $NP-difficiles$, et ne possèdent pas à ce jour de solutions algorithmiques efficaces valables pour toutes le données. Le recours à des méthodes approchées ou heuristiques les

plus populaires, et également les plus efficaces, destinées à la résolution des problèmes d'optimisation combinatoire, sont des techniques générales, appelées métaheuristiques, qu'il s'agit d'adapter à chaque problème particulier [13].

2.2.1 Concept de métaheuristiques

Le terme métaheuristique introduit pour la première fois par Glover (1986), vient des deux mots grecs: heuriskein qui veut dire trouve et meta qui signifie au delà dans un niveau supérieur [3].

Une des définitions de métaheuristiques est donné en 1996 par Osman et Laporte comme suit: un processus itératif qui subordonne et qui 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 [16].

2.3 Caractéristiques principales des métaheuristiques

Les propriétés fondamentales des métaheuristiques sont résumés dans les points suivants:

- Les métaheuristiques sont des stratégies qui permettent de guider la recherche d'une solution optimale.
- Le but visé par les métaheuristiques est d'explorer l'espace de recherche efficacement afin de déterminer des solutions (presque) optimales.
- Les techniques qui constituent des algorithmes de type métaheuristique vont de la simple procédure de recherche locale à des processus d'apprentissage complexes [14].

2.4 Exemple de génération du voisinage

On considère un problème du voyageur de commerce de 6 villes. Calculons le voisinage de la tournée admissible $s = 012345$ en utilisant la méthode de $2 - Opt$:

- Je supprime les deux arêtes $(2, 3)$ et $(4, 5)$
- Je lie les deux extrémités initiales des deux arêtes supprimées

- Je lie les deux extrémités finales des deux arêtes supprimées et je change les directions de l'arête de l'extrémité finale du première arête à l'extrémité initiale du deuxième arête, alors on obtient $\acute{s} = 012435$.

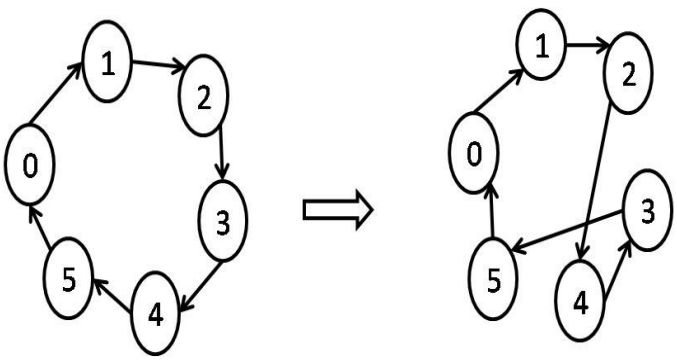

Figure 2.1.Voisinage $2-Opt$

2.5 Méthode Descente

Le principe de la méthode de Descente (dite aussi **basic local search**) consiste à partir d'une solution s et à choisir une solution $\acute{s}$ dans un voisinage de s, telle que $\acute{s}$ améliore la recherche (généralement telle que $f(\acute{s}) < f(s)$).

On peut décider soit d'examiner toutes les solutions du voisinage et prendre la meilleure de toutes (ou prendre la première trouvée), soit d'examiner un sous-ensemble du voisinage.

La méthode de recherche locale la plus élémentaire est la méthode de Descente [1]. On peut la schématiser comme suit:

2.5.1 L'algorithme

Algorithme: méthode Descente

1. Construire une solution initiale s ;
2. **Répéter :**
3. Choisi $\acute{s}$ dans un voisinage $V(s)$ de s ;
4. Si $f(\acute{s}) < f(s)$ alors $s \leftarrow \acute{s}$;
5. Jusqu'à ce que $f(\acute{s}) \geq f(s), \forall\ \acute{s} \in X$;
6. **Fin.**

Algorithme 2.1. Pseudo-code de la Descente

2.6 Avantages et inconvénients

En général, l'efficacité des méthodes de recherche locale simples (Descente) est très peu satisfaisante. D'abord, par définition, la recherche s'arrête au premier minimum local rencontré, c'est là leur principal défaut. Pour améliorer les résultats, on peut lancer plusieurs fois l'algorithme en partant d'un jeu de solutions initiales différentes, mais la performance de cette technique décroît rapidement.

En revanche, autoriser de temps à autre une certaine dégradation des solutions trouvées, afin de mieux explorer tout l'espace des configurations, a conduit au développement des deux méthodes que nous explorons aux paragraphes suivants, à savoir le Recuit Simulé et la méthode Tabou.

Le principal avantage de la recherche locale simple est évidemment sa grande simplicité de mise en œuvre: la plupart du temps, elle ne fait que calculer $f(s+i)-f(s)$, où i correspond à un déplacement élémentaire, et si cette expression peut se simplifier algébriquement, alors on pourra évaluer très rapidement cette différence.

Il est important de remarquer également l'importance du choix de la fonction de voisinage V: un minimum local pour une certaine structure de voisinage ne l'est pas forcément pour une autre. C'est d'ailleurs ce constat qui est à l'origine de la méthode dite de recherche par voisinage variable, qui repose sur la construction de solutions s parmi plusieurs voisinages V_i, plutôt que dans un seul [1].

2.6.1 Domaines d'applications

La méthode de Descente peut être appliquer dans plusieurs domaines citons quelques uns:

- Les mathématiques.
  - Problèmes d'optimisation combinatoire
- Technologie nouvelle.
- Biologie.

2.6.2 Exemple

Appliquer l'algorithme de Descente pour déterminer une solution locale pour le problème du voyageur de commerce de 5 villes $ABCDE$ dont la matrice des distances est donnée par:

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 3 & 1 & 2 & 2 \\ 3 & 0 & 1 & 2 & 3 \\ 5 & 1 & 0 & 1 & 1 \\ 2 & 3 & 1 & 0 & 4 \\ 2 & 2 & 1 & 4 & 0 \end{pmatrix} \end{array}$$

Itération 1:

On choisi au hasard une tournée s de l'espace de solutions admissibles tq:

$|s| = \frac{1}{2}(n-1)\,\text{factor} = \frac{1}{2}(5-1)\,\text{factor} = 12$ tournées possibles.

$s = ACDBE$, donc $f(s) = 5+1+3+2+2 = 13$.

On applique la méthode $(2-Opt)$ pour déterminer toutes les solutions voisins de la solution s:

$|V(s)| = \frac{n}{2}(n-3) = \frac{5}{2}(2) = 5$

paires d'arêtes non adjacentes	tournées voisines	$f(s)$
$\binom{AC}{DB}$	$ADCBE$	8
$\binom{AC}{BE}$	$ABDCE$	10
$\binom{CD}{BE}$	$ACBDE$	15
$\binom{CD}{EA}$	$ACEBD$	14
$\binom{DB}{EA}$	$ACDEB$	15

On choisi le meilleur voisin parmi tous les voisins de s et on note $\acute{s} = ADCBE$

$f(\acute{s}) \prec f(s)$ alors $s \leftarrow \acute{s}$

Itération 2:

Calculer le voisinage de $s = ADCBE$ par la méthode de $(2-Opt)$:

paires d'arêtes non adjacentes	tournées voisines	$f(s)$
$\binom{AD}{CB}$	$ECDBE$	13
$\binom{AD}{BE}$	$ABCDE$	11
$\binom{DC}{BE}$	$ADBCE$	9
$\binom{DC}{EA}$	$ADEBC$	14
$\binom{CB}{EA}$	$ADCEB$	9

On choisi le meilleur voisin parmi tous les voisin de s on note $\acute{s} = ADBCE$

$f(\acute{s}) \geq f(s)$ stop donc la meilleure solution est $s = ADCBE$, $f(s) = 8$.

2.7 Méthode Descente Stochastique

C'est la plus simple des méthodes Stochastiques (**Random walk**). Cette méthode consiste à tirer à chaque itération une solution au hasard. La fonction objective f est évaluée en ce point. La nouvelle valeur est comparée à la précédente. Si elle est meilleure que la précédente, cette valeur est enregistrée, ainsi que la solution correspondante, et le processus

continu. Sinon on repart du point précédent et on recommence le procédé, jusqu'à ce que les conditions d'arrêt soient atteintes. L'algorithme **2.2** présente un pseudo code de la recherche aléatoire dans le cas d'un problème de minimisation [20].

2.7.1 L'algorithme

Algorithme: méthode Descente Stochastique

1. Construire une solution initiale s_0 ;
   $f^* \leftarrow f(s_0)$;
   $s^* \leftarrow s_0$;
2. **Répéter :**
3. Calculer le voisinage de s^* ;
4. $\acute{s} \leftarrow$ solution aléatoire du voisinage ;
5. Si $(f(\acute{s}) < f(s^*))$ Alors
   $f^* \leftarrow f(\acute{s})$;
   $s^* \leftarrow \acute{s}$;
6. Fin Si ;
7. Jusqu'à conditions d'arrêt satisfaites ;
8. **Fin.**

Algorithme 2.2. Pseudo-code de la Descente Stochastique

2.7.2 Exemple

Applique cette algorithme au problème du voyageur de commerce donné au paragraphe précédent; pour déterminer une solution locale

Initialisation:

On choisi au hasard une tournée s de l'espace de solutions admissibles

$s = ACDBE$, avec $f(s) = 13, s^* \leftarrow ACDBE$, $f(s^*) = 13$

On pose $Nmax = 3$ (comme condition d'arrêt) $i = 0$,

Itération 1:

Calculer le voisinage de $s = ACEDB$ toujours par la méthode $(2 - Opt)$

paires d'arêtes non adjacentes	tournées voisines	$f(s)$
$\binom{AC}{DB}$	$ADCBE$	8
$\binom{AC}{BE}$	$ABDCE$	10
$\binom{CD}{BE}$	$ACBDE$	15
$\binom{CD}{EA}$	$ACEBD$	14
$\binom{DB}{EA}$	$ACDEB$	15

On choisi aléatoirement un voisin parmi tous les voisins de s et on note $\acute{s} = ACEBD$ avec $f(\acute{s}) = 14$

$f(\acute{s}) \succ f(s^*)$ donc $s^* \leftarrow ACDBE$ et $f(s^*) \leftarrow 13$

$s \leftarrow ACEBD$, $i = 1$, $i \prec N$ max donc je continue

Itération 2:

Calculer le voisinage de $s = ACEBD$

paires d'arêtes non adjacentes	tournées voisines	$f(s)$
$\binom{AC}{EB}$	$AECBD$	9
$\binom{AC}{BD}$	$ABECD$	9
$\binom{CE}{BD}$	$ACBED$	14
$\binom{CE}{DA}$	$ACDBE$	13
$\binom{EB}{DA}$	$ACEDB$	16

On choisi aléatoirement un voisin parmi tous les voisins de s et on note $\acute{s} = ACEDB$ avec $f(\acute{s}) = 16$

$f(\acute{s}) \succ f(s^*)$ donc $s^* \leftarrow ACEBD$ et $f(s^*) \leftarrow 13$

$s \leftarrow ACEDB$, $i = 2$, $i \prec N$ max donc je continue

Itération 3:

Calculer le voisinage de $s = ACEDB$

paires d'arêtes non adjacentes	tournées voisines	$f(s)$
$\binom{AC}{ED}$	$AECDB$	10
$\binom{AC}{DB}$	$ADECB$	11
$\binom{CE}{DB}$	$ACDEB$	15
$\binom{CE}{BA}$	$ACBDE$	15
$\binom{ED}{BA}$	$ACEBD$	13

On choisi aléatoire voisin parmi tous les voisins de s on note $\acute{s} = ADECB$ avec $f(\acute{s}) = 11$
$f(\acute{s}) \prec f(s^*)$ donc $s^* \leftarrow ADECB$ et $f(s^*) \leftarrow 11$

$s \leftarrow ADECB$, $i = 3$, $i = N$ max stop.
Donc la meilleure solution est $s = ADECB$, $f(s) = 11$.

2.8 Méthode de Recuit Simulé

2.8.1 Historique

Le principe du Recuit Simulé (**Simulated Annealing**) est connu depuis 1953 (Metropolis) simulation du refroidissement de matériaux (Thermodynamique). Mais la méthode à été utilise pour la première fois par (Kirkpatrick et al 1983) utilisation pour la résolution de problème d'optimisation [21].

2.8.2 Définition de la méthode

La méthode du Recuit Simulé s'inspire du processus du Recuit physique. Ce processus utilise en métallurgie pour améliorer la qualité d'un solide recherche un état d'énergie minimale qui correspond à une structure stable du solide.

En partant d'une température élevée à laquelle le solide est devenu liquide, la phase de refroidissement conduit la matière liquide à retrouver sa forme solide par une diminution progressive de la température. Chaque température est maintenue jusqu'à ce que la matière trouve un équilibre thermodynamique.

L'application de ce principe à l'optimisation commence par générer une solution initiale. A chaque itération, on calcule l'énergie de l'état (de la solution), et on diminue la température. Le passage de la solution actuelle à une nouvelle (même plus mauvaise) se fait avec une certaine probabilité qui dépend de la nature des deux solutions et l'avancement de la méthode [9].

Analogie entre un problème d'optimisation combinatoire et un système physique

- La fonction objectif (coût) du problème, étant analogue à l'énergie du métal. Elle est minimisée par l'introduction d'une température (paramètre de contrôle de l'algorithme).
- La température doit conduire vers:
  - L'état optimal si elle est baissée de façon lente et bien contrôlée (technique du Recuit).
  - Un minimum local si elle est baissée brutalement (trempe) [21].

Système physique	Problème d'optimisation
Energie	fonction objectif
Etat du système	solution
État de basse énergie	bonne solution
Température	paramètre de contrôle

Tableau 2.1. Comparaison entre système physique et problème d'optimisation

2.8.3 Principe de Recuit Simulé

Le processus du Recuit Simulé répète une procédure itérative qui cherche des configurations de coût plus faible tout en acceptant de manière contrôlée des configurations qui dégradent la fonction coût. On utilise une méthode Stochastique pour générer une suite d'états successifs du système en partant d'un état initial donné. Tout nouvel état est obtenu en faisant subir un déplacement (une perturbation) aléatoire au système.

Soit:

- $\Delta E = E_2 - E_1$: la différence d'énergie occasionnée par une telle perturbation,

- T: la température absolue du système,
- K: une constante physique appelée: constante de $Boltzmann$, $K = 1,3805.10^{-23} J/K$.

Le nouvel état est accepté si l'énergie du système diminue ($\Delta E \leq 0$); sinon, il est accepté avec une probabilité définie par: $P = e^{-\Delta E/KT}$.

A chaque nouvelle itération, un voisin $\acute{s} \in V(s)$ de la configuration courante s est généré de manière aléatoire. Selon les cas, ce voisin sera soit retenu pour remplacer celle-ci, soit rejeté.

- Si ce voisin est de performance supérieure ou égale à celle de la configuration courante, i.e. $f(\acute{s}) \leq f(s)$, il est systématiquement retenu;
- Dans le cas contraire, $\acute{s}$ est accepté avec une probabilité P qui dépend de deux facteurs:
  - De l'importance de la dégradation $\Delta f = f(\acute{s}) - f(s)$; les dégradations plus faibles sont plus facilement acceptées.
  - D'un paramètre de contrôle T (la température), une température élevée correspond à une probabilité plus grande d'accepter des dégradations[13].

La température est contrôlée par une fonction décroissante qui définit un schéma de refroidissement. Le schéma de refroidissement de la température est l'une des paramètres les plus difficiles à régler. Ce schéma est crucial pour l'obtention d'une implémentation efficace.

Sans être exhaustif, on rencontre habituellement trois grandes classes de schémas:

- **Réduction par paliers**: la température est maintenue constante pendant un certain nombre d'itérations, et décroît ainsi par paliers.

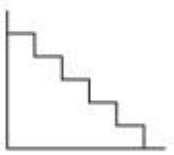

Figure 2.2. Réduction par paliers de la température

- **Réduction continue**: la température est modifiée à chaque itération.

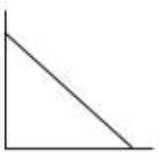

Figure 2.3. Réduction continue de la température

- **Réduction non-monotone**: la température décroît à chaque itération avec des augmentations occasionnelles [9].

Figure 2.4. Réduction non monotone de la température

Parmi les difficultés de la méthode, la détermination de certains paramètres: la valeur initiale de la température (T_0) et le coefficient de décroissance de la température (γ). Le réglage de ces paramètres est assez délicat et repose sur des essais. Certains utilisateurs de cet algorithme posent $\gamma \in [0,85;\ 0,95]$. Quant à la température initiale, celle-ci est déterminée empiriquement ou par des méthodes plus sophistiquées [13].

2.8.4 L'algorithme

Algorithme: méthode de Recuit Simulé

Initialisation

- Choisir $T_0,\ T_f,\ N\max$
- Générer aléatoirement une solution admissible initiale s_0
- Calculer $f_0 = f(s_0), s^* \leftarrow s_0,\ f^* \leftarrow f_0$
- $i \leftarrow 1$

Etape 1

- Choisir $s_1 \in Rand\ (V(s_0))$
- Calculer $\Delta f = f(s_1) - f(s_0)$
- Si $\Delta f \leq 0$ aller en *Etape* 3
- Sinon aller en *Etape* 2

Etape 2

- Choisir une valeur aléatoire $\theta \in [0,1]$ et calculer $p(\Delta f) = e^{-\Delta f/T}$
- Si $\theta \prec p(\Delta f)$ aller en *Etape* 3
- Sinon rejeter cette solution et aller en *Etape* 4

Etape 3

- Accepter la solution
- Poser $s_0 \leftarrow s_1, f_0 \leftarrow f_1$
- Aller en *Etape* 4

Etape 4

- Si $i \succ N\max$ stop
- Sinon diminuer la température en utilisant $T_{i+1} = \gamma * T_i$
- Poser $i \leftarrow i+1$ aller en *Etape* 1

Algorithme 2.3. Poseudo-code de la Recuit Simulé

Pour appliquer le Recuit Simulé, on devrait déterminer quelques paramètres choisis expérimentalement:

– La température initiale T_0.

– La configuration (solution) initiale s_0.

– La règle de transformation permettant de passer d'une configuration à une autre.

– La durée de relaxation (i.e.: le nombre d'itérations de la transformation par palier de température).

– La loi de décroissance de la température.

– Le(s) critère(s) d'arrêt.

2.8.5 Avantages et inconvénients

Avantages:

- Facile à programmer.
- Donne généralement de bonnes solutions par rapport aux algorithmes de recherche classiques.
- Peut être utilisé dans la plupart des problèmes d'optimisation.

Inconvénients:

- La difficulté de déterminer la température initiale et la loi de décroissance de la température.
- Très coûteuse en temps de calcul [4].

2.8.6 Domaines d'applications

Cette méthodes peut être appliquée a d'autre problème d'optimisation combinatoire citons quelques uns:

- Le problème du ordonnancement.
- Le problème du voyageur de commerce.
- Le problème du sac à dos [9].

2.8.7 Exemple

Utiliser la méthode de Recuit Simulé au problème d'ordonnancement de 6 tâches sur une seule machine où l'objectif est de minimiser la somme pondéré des retards $\sum_{i=1}^{6} w_i T_i$ avec ($T_i = \max(C_i - d_i, 0)$, et C_i est la date de fin de la tâche i), $\gamma = 0.95$

i	1	2	3	4	5	6
p_i	7	2	3	8	4	5
w_i	1	5	7	3	6	1
d_i	9	4	8	6	7	5

p_i: le temps d'exécution de tâche i.

w_i: le poids de la tâche i.

d_i: la date d'échue de la tâche i.

Le voisinage utilisé est par insertion

Initialisation:

$T_0 = 10$, $T_f = 1$ et $N\max = 5$

$s_0 = (526431)$

$f_0 = f(s_0) = 173$, $s^* \leftarrow s_0$, $f^* \leftarrow f_0$

$i \leftarrow 1$

Itération 1:

$s_1 \leftarrow (256431)$

$f(s_1) = 163$

$\Delta f = f(s_1) - f(s_0)$, $\Delta f = -10 \prec 0$ alors accepte la solution donc $s_0 \leftarrow s_1$, $f_0 \leftarrow f_1$

$T = 9.50$

$i \leftarrow 2$, $i \prec N\max$ donc je continue.

Itération 2:

$s_1 \leftarrow (652431)$

$f(s_1) = 204$

$\Delta f = f(s_1) - f(s_0)$, $\Delta f = 41 \succ 0$, alors $\theta = 0.20$ et $p(\Delta f) = 0.013$

$\theta \succ p(\Delta f)$ donc on rejette s_1

$T = 9.025$

$i \leftarrow 3$, $i \prec N$ max donc je continue.

Itération 3:

$s_1 \leftarrow (456231)$

$f(s_1) = 241$

$\Delta f = f(s_1) - f(s_0)$, $\Delta f = 78 \succ 0$, alors $\theta = 0.23$ et $p(\Delta f) = 0.00017$

$\theta \succ p(\Delta f)$ donc on rejette s_1

$T = 8.57$

$i \leftarrow 4$, $i \prec N$ max donc je continue.

Itération 4:

$s_1 \leftarrow (356421)$

$f(s_1) = 169$

$\Delta f = f(s_1) - f(s_0)$, $\Delta f = 6 \succ 0$, alors $\theta = 0.55$ et $p(\Delta f) = 0.4965$

$\theta \succ p(\Delta f)$ donc on rejette s_1

$T = 8.14$

$i \leftarrow 5$, $i = Nmax$ stop.

La meilleure solution est donc: $s^* = (256431)$ et $f^* = 163$.

2.9 Méthode Tabou

2.9.1 Historique

La méthode de Recherche avec Tabou, ou simplement Recherche Tabou ou méthode Tabou (**Tabu Search**), à été formalisée en 1986 par F. Glover**.** Cette méthode est une métaheuristique itérative qualifiée de recherche locale au sens large [4].

2.9.2 Définition de la méthode

La méthode Tabou, fait appel à un ensemble de règles et de mécanismes généraux pour guider la recherche de manière intelligente au travers de l'espace des solutions. A l'inverse du Recuit Simulé qui génère de manière aléatoire une seule solution voisine $\acute{s} \in V(s)$ à chaque itération, Tabou examine un échantillonnage de solutions de $V(s)$ et retient la meilleure $\acute{s}$

même si $\acute{s}$ est plus mauvaise que s. La Recherche Tabou ne s'arrête donc pas au premier optimum trouvé. Cependant, cette stratégie peut entraîner des cycles, par exemple un cycle de longueur 2 : $s \rightarrow \acute{s} \rightarrow s \rightarrow \acute{s}$... Pour empêcher ce type de cycle, on mémorise les k dernières configurations visitées dans une mémoire à court terme et on interdit tout mouvement qui conduit à une de ces configurations. Cette mémoire est appelée la liste Tabou, une des composantes essentielles de cette méthode. Elle permet d'éviter tous les cycles de longueur inférieure ou égale à k. La valeur de k dépend du problème à résoudre et peut éventuellement évoluer au cours de la recherche [9].

2.9.3 L'algorithme

Algorithme: méthode Tabou

1 : Initialisation

s_0 une solution initiale

$s \leftarrow s_0,\ s^* \leftarrow s_0,\ f^* \leftarrow f(s_0)$

$T = \emptyset$

2 : Générer un sous-ensemble de solutions au voisinage de s

$\acute{s} \in V(s)$ tel que $\forall x \in V(s),\ f(x) \geq f(\acute{s})$ et $\acute{s} \notin T$

Si $f(\acute{s}) \prec f^*$ alors $s^* \leftarrow \acute{s}$ et $f^* \leftarrow f(\acute{s})$

Mise-à-jour de T

3 : Si la condition d'arrêt n'est pas satisfaite retour à l'étape 2

Algorithme 2.4. Pseudo-code de la Recherche Tabou

Caractéristiques de l'algorithme

Voici l'explication des différentes variables du schéma:

s_0: solution initiale.

s^*: meilleure solution jusqu'à présent.

$\acute{s}$ et s: nouvelle solutions du voisinage de s^*.

$f(s)$: fonction objectif à minimiser.

$f^* = f(s^*)$: valeur de la meilleure solution.

2.9.4 Les mécanismes principaux de la méthode

Dans la Recherche Tabou, la mémorisation des solutions récemment visitées dans une liste des Tabous, afin d'éviter le cyclage est le mécanisme essentiel de la méthode. Mais d'autres mécanismes sont employées aussi, pour améliorer le processus de recherche.

La liste des Tabous

La liste des Tabous est une mémoire à court terme, qui évite le retour à une solution récemment visitée. Sa mise à jour se fait à chaque itération. La mémorisation de configurations entières serait trop coûteuse en temps de calcul et en place mémoire et ne serait sans doute pas la plus efficace. Il est courant de ne garder dans cette liste que des informations, soit sur les caractéristiques des solutions, soit sur les mouvements, au lieu de configurations complètes.

Redimensionnement des paramètres de la méthode Tabou

Dans certains cas, il est nécessaire d'ajuster la longueur de la liste Tabou selon la qualité de voisinage. Si la liste est trop courte, la Recherche finit par explorer un optimum local. A l'inverse, si la liste est trop longue, tous les mouvements peuvent devenir Tabous, et la Recherche se bloque. Le réglage de la longueur de la liste dépend essentiellement de la topologie de l'espace des solutions, donc de la qualité de voisinage [13].

2.9.5 Intensification et diversification

L'intensification et la diversification sont deux techniques (parmi d'autres) utilisées pour améliorer la recherche avec Tabou:

Intensification (exploitation): focaliser l'effort de recherche sur une zone particulière de l'espace de recherche (autour des meilleures solutions rencontrées).

Diversification (exploration): explorer des nouvelles zones prometteuses de l'espace de recherche pour trouver potentiellement d'autres bonnes solutions [2].

2.9.6 Avantages et inconvénients

Avantages:

- Grande efficacité.
- Fonctionnement simple à comprendre.

Inconvénients:

- Paramètres peu intuitifs.
- Demande en ressources importantes si la liste des Tabous est trop imposante.
- Aucune démonstration de la convergence [4].

2.9.7 Domaines d'applications

Citons quelques domaines d'applications:

- Le problème du voyageur de commerce.
- Le problème du sac à dos.
- Le problème d'ordonnancement.
- Le problème de coloration de graphes[8] .

2.9.8 Exemple

Appliquer la méthode Tabou pour déterminer un minimum local pour le problème du voyageur de commerce de 5 villes dont la matrice des coûts (des distances entre villes) est donnée ci-dessous

$$d_{ij} = \begin{pmatrix} 0 & 5 & 4 & 3 & 4 \\ 5 & 0 & 1 & 6 & 5 \\ 4 & 1 & 0 & 2 & 7 \\ 3 & 6 & 2 & 0 & 6 \\ 4 & 5 & 7 & 6 & 0 \end{pmatrix} \quad \text{telle que } (1 \leq i,\ j \leq 5),$$

$s_0 = (12435)$ générée aléatoiment et $f(s_0) = 24$

Initialisation:

$(N \max = 4)$, $t = 3$

$s \leftarrow s_0$, $s^* \leftarrow s_0 = (12435)$, $f^* = f(s_0) = 24$

liste tabou $\leftarrow \phi$

Itération 1: je calcule le $V(s)$ par $2 - Opt$

tournée voisine	(14235)	(13425)	(12345)	(12534)	(12453)
$f(s)$	17	21	18	22	28

donc: $\acute{s} \leftarrow (14235) = 17$ meilleur voisin non Tabou

liste Tabou $T = \{(14235)\}$

$f(\acute{s}) \prec f^*$ alors $s^* \leftarrow \acute{s}$, $f^* \leftarrow f(\acute{s})$

$s^* \leftarrow (14235)$, $i = 1, i \prec N \max$ donc je continue

Itération 2: je calcule le $V(s)$

tournée voisine	(12435)	(13245)	(14325)	(14532)	(14253)
$f(s)$	24	22	15	22	25

donc: $\acute{s} \leftarrow (14325) = 15$ meilleur voisin non Tabou

liste Tabou $T = \{(14235)\,,\ (14325)\}$

$f(\acute{s}) \prec f^*$ alors $s^* \leftarrow \acute{s}$, $f^* \leftarrow f\ (\acute{s})$

$s^* \leftarrow (14325)$, $i = 2, i \prec N \max$ donc je continue

Itération 3: je calcule le $V(s)$

tournée voisine	(13425)	(12345)	(14235)	(14523)	(14352)
$f(s)$	21	18	21	19	19

donc: $\acute{s} \leftarrow (12345) = 18$ meilleur voisin non Tabou

liste Tabou $T = \{(14235)\,,\ (14325)\,,(12345)\}$

$f(\acute{s}) \succ f^*$ alors $s^* \leftarrow (14325)$, $f^* \leftarrow 15$, $i = 3, i \prec N \max$ donc je continue

Itération 4: je calcule le $V(s)$

tournée voisine	(13245)	(14325)	(12435)	(12543)	(12354)
$f(s)$	21	15	24	22	22

on a (14325) meilleur voisine mais tabou donc (13245) meilleur voisin non Tabou

liste Tabou $T = \{(14325), (12345), (13245)\}$

$i = 4$, $i = Nmax$ stop.

Le programme de recherche s'arrête est la meilleur solution est $s^* = (14325)$ avec $f^* = 15$.

2.10 Quelle métaheuristique utiliser ?

Le premier problème pratique qui se pose à un utilisateur confronté à une application concrète est d'effectuer un choix parmi les différentes métaheuristiques disponibles. Ce choix est d'autant plus difficile qu'il n'existe pas de comparaison générale et fiable des différentes métaheuristiques. Cependant, il est possible de caractériser les métaheuristiques selon quelques critères généraux, ce qui pourrait faciliter ce choix. Le tableau de synthèse ci-dessous met en relation cinq critères avec trois métaheuristiques parmi les plus représentatives. Les indications sont présentées à titre purement indicatif et correspondent aux résultats publiés. Il doit être clair que le choix d'une métaheuristique appropriée ne constitue qu'une condition nécessaire. La qualité des solutions trouvées par une méthode peut être très variable selon l'implémentation réalisée.

	AI	RS	Tabou
Simplicité	0	−	−
Facilité d'adaptation	0	0	−
Connaissance	0	0	+
Qualité	0	+	++
Rapidité	0	−	−

Tableau 2.2. Comparaison générale des principales métaheuristiques

Dans ce tableau, les trois métaheuristiques comparées sont les suivantes:

- AI: amélioration itérative (Descente Stochastique),

- RS: Recuit Simulé,

- Tabou: méthode Tabou,

Les critères de comparaisons retenus sont les suivants:

- Simplicité de la métaheuristique, i.e. simplicité de la méthode elle-même,
- Facilité d'adaptation au problème,
- Possibilité d'intégrer des connaissances spécifiques du problème,
- Qualité des meilleures solutions trouvées,
- Rapidité, i.e., le temps de calcul nécessaire pour trouver une telle solution.

La méthode d'amélioration itérative est utilisée comme point de référence pour l'ensemble des méthodes : les signes $-$, 0, $+$ indiquent des performances respectivement inférieures, égales, supérieures à celles obtenues par l'amélioration itérative [1].

2.11 Conclusion

Nous avons défini dans ce chapitre les métaheuristiques à base solution unique sont Descente, Descente Stochastique, le Recuit Simulé et la Recherche Tabou. Ces quatre métaheuristiques étaient restent l'objet d'une recherche intense pour la résolution de divers problèmes $NP-difficiles$. Elles ont révélé leurs grandes efficacités de fournir des solutions approchées de bonne qualité pour un grande nombre de problèmes.

Pour chaque méthode on a présenté l'algorithme et on a défini la concept, le mécanisme et les caractéristiques principales. On a donné un tableau comparatif des critères d'utilisation de ces méthodes. De plus un exemple illustratif est donné pour éclaircir le fonctionnement de chacune.

Chapitre 3

Etude de cas

3.1 Introduction

Dans ce chapitre on va choisir une des métaheuristiques étudiées avant qui est la méthode Descente Stochastique et on va d'appliquer sur le problème du voyageur de commerce. Pour déterminer une solution locale. En fait on va présenter une étude numérique comparative utilisant le langage dit Java pour l'implémentation de l'algorithme.

3.2 Définition Java

Java est un langage de programmation orienté, développé par Sun Microsystems. Il très utilisé, notamment par un grand nombre de développeurs professionnels, ce qui en fait un langage incontournable actuellement. Le langage reprend en grande partie la syntaxe C++ [10].

NetBeans est un environnement de développement intégré, placé en open source par Sun en juin 2000 sous licence CDDL et GPLv2. En plus de Java, NetBeans permet également de supporter différents autres langages, comme C, C++ et HTML ...etc.

3.3 Génération des données

On a généré 10 problèmes différents chacun pour 5 villes où on a considèré dans l'intervalle $[1, 100]$ dix intervalles équitables. Les distances entre les villes sont générées (sur les lois

uniformes) comme ceci: pour le problème P_1 les $d_1 \in U(1,10)$, le problème P_2 les $d_2 \in U(11,20)$, le problème P_3 les $d_3 \in U(21,30)$, le problème P_4 les $d_4 \in U(31,40)$, le problème P_5 les $d_5 \in U(41,50)$, le problème P_6 les $d_6 \in U(51,60)$, le problème P_7 les $d_7 \in U(61,70)$, le problème P_8 les $d_8 \in U(71,80)$, le problème P_9 les $d_9 \in U(81,90)$ et le problème P_{10} les $d_{10} \in U(91,100)$. Alors que le nombre maximal d'itérations est fixé à 3 (Nmax = 3). Quant à la solution de départ est choisit au hasard.

Le problème 1:

s_0	$f(s_0)$	$V(s_0)$	$f(V(s_0))$	$\acute{s}$	$f(\acute{s})$	$f(\acute{s}) \prec f(s^*)$	s^*
03124	20	01324	27	02134	32	$32 \succeq 20$	03124
		02134	32				
		03214	21				
		03421	22				
		03142	20				
02134	32	01234	35	02431	30	$30 \succeq 20$	03124
		03124	20				
		02314	29				
		02431	30				
		02143	24				
02431	30	04231	27	03421	22	$22 \succeq 20$	03124
		03421	22				
		02341	31				
		02134	32				
		02413	20				

Le problème 2:

s_0	$f(s_0)$	$V(s_0)$	$f(V(s_0))$	$\acute{s}$	$f(\acute{s})$	$f(\acute{s}) \prec f(s^*)$	s^*
03124	75	01324	73	03214	72	$72 \prec 75$	03214
		02314	74				
		03214	72				
		03421	81				
		03142	73				
03214	72	02314	68	02314	68	$68 \prec 72$	02314
		01234	76				
		03124	75				
		03412	76				
		03241	75				
02314	68	03214	72	01324	73	$73 \succeq 68$	02314
		01324	73				
		02134	74				
		02413	73				
		02341	74				

Le problème 3:

s_0	$f(s_0)$	$V(s_0)$	$f(V(s_0))$	$\acute{s}$	$f(\acute{s})$	$f(\acute{s}) \prec f(s^*)$	s^*
03124	136	01324 02314 03214 03421 03142	128 132 121 131 126	02314	132	$132 \prec 136$	02314
02314	132	03214 01324 02134 02413 02341	121 128 132 126 115	01324	128	$128 \prec 132$	01324
01324	128	03124 02314 01234 01423 01342	136 120 125 119 130	01342	130	$130 \succeq 128$	01324

Le problème 4:

s_0	$f(s_0)$	$V(s_0)$	$f(V(s_0))$	$\acute{s}$	$f(\acute{s})$	$f(\acute{s}) \prec f(s^*)$	s^*
03124	185	01324 02134 03214 03421 03142	180 186 176 181 176	03421	181	$181 \prec 185$	03421
03421	181	04321 02431 03241 03124 03412	181 186 176 185 177	02431	186	$186 \succeq 181$	03421
02431	186	04231 03421 02341 02134 02413	180 181 172 186 176	02413	176	$176 \prec 181$	02413

Le problème 5:

s_0	$f(s_0)$	$V(s_0)$	$f(V(s_0))$	$\acute{s}$	$f(\acute{s})$	$f(\acute{s}) \prec f(s^*)$	s^*
03124	230	01324 02134 03214 03421 03142	220 235 227 228 226	03142	226	$226 \prec 230$	03142
03142	226	01342 04132 03412 03241 03124	226 225 233 218 230	03412	233	$233 \succeq 226$	03142
03412	233	04312 01432 03142 03214 03421	235 223 226 227 228	01432	223	$223 \prec 226$	01432

Le problème 6:

s_0	$f(s_0)$	$V(s_0)$	$f(V(s_0))$	$\acute{s}$	$f(\acute{s})$	$f(\acute{s}) \prec f(s^*)$	s^*
03124	277	01324 02134 03214 03421 03142	271 277 272 282 283	03421	282	$282 \succeq 277$	03124
03421	282	01342 04132 03412 03241 03124	282 272 283 277 277	04132	272	$272 \prec 277$	04132
04132	272	01432 03142 04312 04231 04123	277 283 277 271 272	03142	283	$283 \succeq 272$	04132

Le problème 7:

s_0	$f(s_0)$	$V(s_0)$	$f(V(s_0))$	$\acute{s}$	$f(\acute{s})$	$f(\acute{s}) \prec f(s^*)$	s^*
03124	323	01324 02134 03214 03421 03142	324 337 332 336 329	02134	337	$337 \succeq 323$	03124
02134	337	01234 03124 02314 02431 02143	329 332 332 326 334	02431	326	$326 \succeq 323$	03124
02431	326	04231 03421 02341 02134 02413	324 326 326 337 329	02413	329	$329 \succeq 323$	03124

Le problème 8:

s_0	$f(s_0)$	$V(s_0)$	$f(V(s_0))$	$\acute{s}$	$f(\acute{s})$	$f(\acute{s}) \prec f(s^*)$	s^*
03124	379	01324 02134 03214 03421 03142	375 382 379 377 380	01324	375	$375 \prec 379$	01324
01324	375	03124 02314 01234 01423 01342	379 380 377 375 378	01234	377	$377 \succeq 375$	01324
01234	377	02134 03214 01324 01432 01243	382 379 375 378 377	02134	382	$382 \succeq 375$	01324

Le problème 9:

s_0	$f(s_0)$	$V(s_0)$	$f(V(s_0))$	$\acute{s}$	$f(\acute{s})$	$f(\acute{s}) \prec f(s^*)$	s^*
03124	427	01324	429	03142	425	$425 \prec 427$	03142
		02134	431				
		03214	424				
		03421	424				
		03142	425				
03142	425	01342	430	03241	422	$422 \prec 425$	03241
		04132	429				
		03412	424				
		03241	422				
		03124	427				
03241	422	02341	426	02341	426	$426 \succeq 422$	03241
		04231	429				
		03421	424				
		03142	425				
		03214	424				

Le problème 10:

s_0	$f(s_0)$	$V(s_0)$	$f(V(s_0))$	$\acute{s}$	$f(\acute{s})$	$f(\acute{s}) \prec f(s^*)$	s^*
03124	482	01324	483	03142	481	$481 \prec 482$	03142
		02134	482				
		03214	482				
		03421	478				
		03142	481				
03142	481	01342	480	01342	480	$480 \prec 481$	01342
		04132	484				
		03412	479				
		03241	480				
		03124	482				
01342	480	03142	481	01243	478	$478 \prec 480$	01243
		04312	482				
		01432	480				
		01243	478				
		01324	483				

3.4 Discussion des résultats empiriques

En regardent les tableaux des résultats numériques ci-dessus, on a remarqué que:

- Plus de 50% des cas la méthode Descente Stochastique saute à une solution meilleur, ce qui justifie l'importance de la diversification de la méthode.

- Les valeurs des solutions locales sont très proches de celles des solutions optimale exactes.

3.5 Conclusion

Dans ce chapitre on a pris un cas à étudier. C'est l'application du métaheuristique: Descente Stochastique pour déterminer une solution approchée au problème du voyageur de commerce. Divers problèmes ont été générés pour une simulation numérique pour l'algorithme général. La solution initiale est choisi au hasard, le nombre maximal d'itérations est fixé à 3. Les distances sont générées uniformément aléatoires.

Les résultats finaux indiquent que plus de 50% des cas la méthode de Descente Stochastique choisi une solution meilleur que la solution en cours. Aussi la valeur de la fonction objectif est dans la majorité des cas est proche de la valeur de la solution optimale.

Conclusion

Notre projet est consacré à l'étude des métaheuristiques à solution unique: méthode Descente, Descente Stochastique, Recuit Simulé, Tabou. Ces méthodes sont appliquées aux problèmes d'optimisation combinatoire pour déterminer une solution locale là où les méthodes exactes, telles que la méthode par séparation et évaluation, la programmation dynamique échouent pour déterminer une solution exacte et surtout lorsque la taille du problème considéré est très grande. Dans cette condition ces métaheuristiques ont connu une attention particulière de la part des chercheurs dans ce domaine, par leurs efficacités et leurs efficiences, et par conséquent deviennent l'alternative incontournable des méthodes exactes. Dans notre manuscrit on a essayé de rappeler ces méthodes et on a essayé de les appliquer sur différents problèmes d'optimisation combinatoire: méthode Descente sur le problème du voyageur de commerce, Descente Stochastique sur le problème du voyageur de commerce, Recuit Simulé sur le problème d'ordonnancement, Tabou sur le problème du voyageur de commerce. Il est important de rappeler que la construction d'un voisinage pour une métaheuristique est un facteur déterminant pour son efficacité. Deux méthodes ont été données celle pour générer le voisinage de $\lambda - Opt$ pour le PVC et celle pour générer le voisinage par permutation et insertion du tâches pour le problème d'ordonnancement. Finalement une étude de cas à été établit pour résoudre le problème du voyageur de commerce par la méthode Descente Stochastique. Un programme de l'algorithme à été construit en utilisant le langage Java.

Bibliographie

[1] B. Autin. Les métaheuristiques en optimisation combinatoire. Mémoire de master, Université de Paris, 2006.

[2] S. Ben Ismail. Introduction à l'optimisation combinatoire. Telecom Bretagne, (2012).

[3] C. Blum, R. Audrea. Metaheuristics In Combinatorial Optimization: Overview And Conceptual Comparison. ACM Computing Surveys, vol 35, n°3, (2003), 268-308.

[4] J. Dréo, A. Pétrowski, P. Siarry, É. Taillard. Métaheuristiques pour l'optimisation difficile. Eyrolles, (2003).

[5] P. Esquirol, P. Lopez. L'ordonnancement. Economica, Paris, (1999).

[6] M. R. Garey, D. S. Johnson. Computers and Intractability: A guide to the theory of NP-completeness. W. H. Freeman and company, New York, (1979).

[7] A. Gherboudj. Méthode de résolution de problèmes difficiles Académiques. Thèse de doctorat, Université Constantine, 2013.

[8] F. Glover. Tabu search fundamentals and uses. University of Colorado, (1995).

[9] J. K. Hao, P. Galinier, M. Habib. Métaheuristique pour l'optimisation combinatoire et l'affectation sous contraintes. Revue d'Intellgence Artificielle, vol 13. n°2, (1999), 283-324.

[10] C. Herby. Apprenez à programmer en Java. Imprim'vert, France, (2011).

[11] http://deptinfo.unice.fr/∼regin/cours/cours/ResoPb/ResoPb.pdf.

[12] http://sis univ-tln.fr/tollari/ter/AlgoGen1/node4.htm.

[13] M. Kebabla. Utilisation des statégies Métaheuristiques pour l'ordonnancement des ateliers de type Job Shop. Mémoire magister, Université de Batna, 2008.

[14] S. Le Digabel. Introduction aux métaheuristiques. École Polytechniaue de Montréal, (2014).

[15] M. E. Marion. Recherche locale et optimisation combinatoire: de l'analyse structurelle d'un problème à la conception d'algorithme efficaces. Thèse doctorat, Université Lille 1, 2011.

[16] I. H. Osman, G. Laporte. Metaheuristics: a bibliography. Revue of Operations Research, vol 63, n°5, (1996), 511-623.

[17] D. Pisinger. Where are the hard knapsack problems? Computers and Operations Research. Vol 32, n° 9, (2005), 2271-2284.

[18] J. P, Préaux. Optimisation Combinatoire. Université de provence, (2010).

[19] A. Schrijver. Combinatorial optimization: polyhedra and efficiency. Springer Verlag Berlin Heidelberg, New York, (2003).

[20] I. Souici. Sécurisation évolutionnaire du transfert d'images. Thèse doctorat, Université Annaba, 2013.

[21] S. Verel. Métaheuristique: Recherches locales et Algorithmes evolutionnaires. Université Lille, (2012).