الجمهوريـــة الجزائـريـــة الديمقراطيـــة الشعبيـــة
République Algérienne Démocratique et Populaire
وزارة التعليـــم العـالـي والبحـث العلمـي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

N° Ref :.....

Centre Universitaire de Mila

Institut des sciences et de la technologie **Département de Mathématiques et Informatique**

L'UTILISATION DE LA MÉTHODE RECHERCHE TABOU POUR RÉSOUDRE UN PROBLÈMES D'OPTIMISATION

Mémoire préparé En vue de l'obtention du diplôme de Master en Mathématiques

préparé par : *MIMOUNI Mouna*
LALAOUI Fatima Zohra

Encadré par : *Mr . ZAIDI Ali*

Devant le jury :
Président : *Mr . BOURIDEH Adel*
Coordinateur : *Mr . ZAIDI Ali*
Examinateur : *Mr . AZI Mourad*

Filière : Mathématiques

Spécialité : Mathématiques Fondamentales et Appliqué

Année universitaire :2012/2013

Remerciement

Nous remercierons dieu le tout puissant pour nous avoir
Offert la patience durant toutes ces années
Nous tenons à remercier,tous ceux qui nous ont aidé a
Eclaircir dans accomplissement de cett tâche,et plus particulièrement

- Le profe « Zaidi Ali » pour diriger ce travail
- Les deux maitre de conférence Bourideh Adel et
  Talai Meriem

Nous remercierons aussi :
-Mimouni Abdelmadjid, Lalaoui Soumia, Nesrouche Hayat
et tous ceux qui ont supervisé avec beaucoup D'attention notre travail

*****MERCI A TOUT
Mouna et FATIMA

Résumé

Dans ce travail, on parle des utilisation de la méthode de Recherche Tabou pour la résolution d'un problème d'optimisation ,où s'intéresse dans le premier chapitre des problèmes d'optimisation mono_objectifs et c'est méthodes de résolution , et aussi des problèmes classiques d'optimisation combinatoire .
Ensuite , Le deuxième chapitre est consacré à l'étude de méthode de Recherche Tabou , lorsque nous définissons cette méthode et nous donnons son algorithme .
En fin, le $3^{ème}$ chapitre c'est la partir programmation de l'algorithme de Recherche Tabou lorsque nous avons programmé cette algorithme qui résoudre les problèmes du voyageur de commerce dans un langage de programation «Matlab».

Abstract

In this work, we talk about the use of Tabu Search method for solving an optimization problem, , which is interested in the first chapter of mono_objectifs optimization problems and it is solving methods, and also classical combinatorial optimization
problems.
Then the second chapter is devoted to the study of Tabu Search method, when we define this method and give its algorithm
In the end , 3^{th} chapter is from the programming algorithm Tabu Search when we programmed the algorithm solving the traveling salesman problem in the programming language "Matlab".

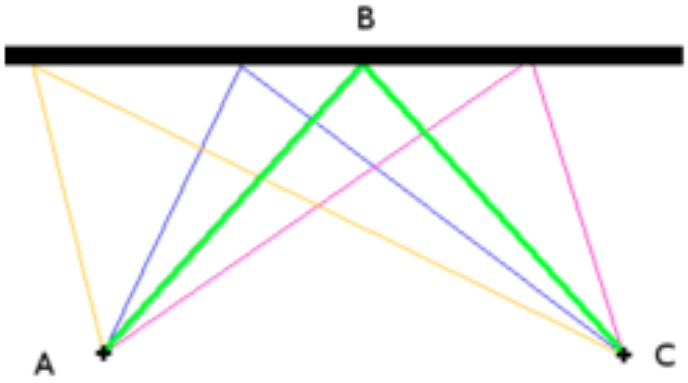
Table des matières

INTRODUCTION GÉNÉRAL

L'optimisation est une branche des mathématiques, cherchant à analyser et à résoudre analytiquement ou numériquement les problèmes qui consistent à déterminer le meilleur élément d'un ensemble, au sens d'un critère quantitatif donné. Ce mot vient du latinoptimum qui signifie le meilleur.

L'optimisation joue un rôle important en recherche opérationnelle (donc en économie et microéconomie), dans les mathématiques appliquées (fondamentales pour l'industrie et l'ingénierie), en analyse et en analyse numérique, en statistique pour l'estimation du maximum de vraisemblance d'une distribution, pour la recherche de stratégies dans le cadre de la théorie des jeux, ou encore en théorie du contrôle et de la commande.

Ils sont extrêmement variés : optimisation d'un trajet, de la forme d'un objet, d'un prix de vente, d'une réaction chimique, du contrôle aérien, du rendement d'un appareil, du fonctionnement d'un moteur, de la gestion des lignes ferroviaires, du choix des investissements économiques, de la construction d'un navire, etc. L'optimisation de ces systèmes permet de trouver une configuration idéale, d'obtenir un gain d'effort, de temps, d'argent, d'énergie, de matière première, ou encore de satisfaction. Les premiers problèmes d'optimisation auraient été formulés par Euclide, au IIIe siècle avant notre ère, dans son ouvrage historique Éléments. Trois cent ans plus tard, Héron d'Alexandrie dans Catoptrica énonce le principe du plus court chemin dans le contexte de l'optique.(voir figure)



Le plus court chemin pour aller de A à C en passant par un point B de la droite est obtenu lorsque l'angle d'incidence est égal à l'angle réfléchi (sur la figure, il s'agit du chemin vert).
Au XVIIe siècle, l'apparition du calcul différentiel entraîne l'invention de techniques d'optimisation, ou du moins en fait ressentir la nécessité. Newton met au point une méthode itérative permettant de trouver les extrémums locaux d'une fonction en faisant intervenir la notion de dérivée, issue de ses travaux avec Leibniz3. Cette nouvelle notion permet de grandes avancées dans l'optimisation de fonctions car le problème est ramené à la recherche des racines de la dérivée.
Durant le XVIIIe siècle, les travaux des mathématiciens Euler et Lagrange mènent au calcul des variations, une branche de l'analyse fonctionnelle regroupant plusieurs méthodes d'optimisation. Ce dernier invente une technique d'optimisation sous contraintes : Les multiplicateurs de Lagrange. Le XIXe siècle est marqué par l'intérêt croissant des économistes pour les mathématiques. Ceux-ci mettent en place des modèles économiques qu'il convient d'optimiser, ce qui accélère le développement des mathématiques. Depuis cette période, l'optimisation est devenue un pilier des mathématiques appliquées et le foisonnement des techniques est tel qu'il ne saurait être résumé en quelques lignes.

On peut tout de même évoquer l'invention de plusieurs méthodes itératives utilisant le gradient de la fonction, ainsi que l'utilisation du terme programmation mathématique, pour désigner des problèmes d'optimisation.

Historiquement, le premier terme introduit fut celui de programmation linéaire, inventé par George Dantzig vers 1947. Le terme programmation dans ce contexte ne

réfère pas à la programmation informatique (bien que les ordinateurs soient largement utilisés de nos jours pour résoudre des programmes mathématiques). Il vient de l'usage du mot programme par les forces armées américaines pour établir des horaires de formation et des choix logistiques, que Dantzig étudiait à l'époque. L'emploi du terme programmation avait également un intérêt pour débloquer des crédits en une époque où la planification devenait une priorité des gouvernements. L'expression programmation mathématique, qui requiert la longue explication ci-dessus, tend à être abandonnée. Par exemple, en juin 2010, la société savante internationale qui représente cette discipline a vu son nom précédent Mathematical Programming Society changé en Mathematical Optimization Society ; pour la même raison, on préfère aujourd'hui utiliser les locutions optimisation linéaire/quadratique/... au lieu de programmation linéaire/quadratique/....

Chapitre 1

Les problèmes d'optimisation mono-objectifs

1.1 Introduction sur les problèmes d'optimisation :

De nombreux secteurs de l'industrie sont concernés par les problèmes d'optimisation combinatoire . En effet, que l'on s'intéresse à l'optimisation d'un système de production, au traitement d'images, à la conception de systèmes, au design de réseaux de télécommunication ou à la bio-informatique nous pouvons être confrontés à des problèmes d'optimisation combinatoire . Plusieurs Problèmes ont été traités dans différents domaines : -design de systèmes dans les sciences d'ingénieurs (mécanique, aéronautique, chimie, etc.) : ailes d'avions [44], moteurs d'automobiles [29] ; -ordonnancement et affectation : ordonnancement en productique [31], localisation d'usines, planification de trajectoires de robots mobiles [30], etc. -agronomie : programme de production agricole, etc. -transport : gestion de containers [10], design de réseaux de transport [46], tracé autoroutier, etc. -environnement : gestion de la qualité de l'air [47], distribution de l'eau [9], etc. -télécommunications : design d'antennes [11], affectation de fréquences [3], radiotéléphonie mobile [18], etc.
a) Un problème d'optimisation est défini par :
-un espace de recherche (de décision) : ensemble de solutions ou de configurations constitué des différentes valeurs prises par **les variables de décision**.
-une ou plusieurs **fonction(s)** dite **objectif(s)**, à optimiser (minimiser ou maximiser).
-un ensemble de **contraintes** à respecter.
Dans la plupart des problèmes, l'espace d'état (décision) est fini ou dénombrable. **Les variables** du problème peuvent être de nature diverse (réelle, entier,

booléenne, etc.) et exprimer desdonnées qualitatives ou quantitatives.
La fonction objectif représente le but à atteindre pour ledécideur.
L'ensemble de contrainte définit des conditions sur l'espace d'état que les variables doivent satisfaire. Ces contraintes sont souvent des contraintes d'inégalité ou d'égalité et permettent en général de limiter l'espace de recherche (solutions réalisables). La résolution optimale du problème consiste à trouver le point ou un ensemble de points de l'espace de recherche qui satisfait au mieux la fonction objectif. Le résultat est appelé **valeur optimale** ou **optimum** . Néanmoins en raison de la taille des problèmes réels, la résolution optimale s'est souvent montrée impossible dans un temps raisonnable. Cette impossibilité technique impose la résolution approchée du problème, qui consiste à trouver une solution de bonne qualité (la plus proche possible de l'optimum). Il est vital pour déterminer si une solution est meilleure qu'une autre, que le problème introduise un critère de comparaison **(une relation d'ordre)**. La plupart des problèmes d'optimisations appartiennent à la classe des problèmes **NP-difficile** classe où il n'existe pas d'algorithme qui fournit la solution optimale en temps polynomial en fonction de la taille du problème et le nombre d'objectifs à optimiser. Dans la littérature il existe des **problèmes académiques** utilisés comme des benchmarks : sac à dos, les fonctions de schaffer, voyageur de commerce, Flowshop, ... et des problèmes réels (applications industrielles) : télécommunications, transport, environnement,...
b) Un problème d'optimisation est caractérisé par :
-le domaine des variables de décision : soit Continu et on parle alors de **problème continu**, soit discret et on parle donc de problème combinatoire ;
-la nature de la fonction objectif à optimiser : soit linéaire et on parle alors de **problème linéaire**, soit non linéaire et on parle donc de **problème non linéaire** ;
-le nombre de fonctions objectifs à optimiser : soit une fonction scalaire et on parle alors de problème **mono-objectif**, soit une fonction vectorielle et on parle donc de **problème multi objectif** ;
-la présence ou non des contraintes : on parle de **problème sans contrainte** ou **avec contrainte**
-sa taille : **problème de petite** ou **de grande taille** ;
-l'environnement :**problème dynamique** (la fonction objectif change dans le temps).
c) Face à un problème d'optimisation :
-Elaborer un modèle (mathématiques) : l'expression de l'objectif à optimiser et les contraintes à respecter.
-Développer un algorithme de résolution.
-Evaluer la qualité des solutions produites.

1.2 Les problème d'optimisation mono-objectif

Lorsqu'un seul objectif (critère) est donné ,le problème d'optimisation est mono-objectif.Dans ce cas la solution optimale est clairement définie , c'est celle qui a le coût optimal (minimal,maximal). De manière formelle, à chaque instance d'un tel problème est associé un ensemble Ω des solutions potentielles
respectant certaines contraintes et une fonction d'objectif $f : \Omega \longrightarrow \Psi$ qui associe à chaque solution admissible $s \in \Psi$ une valeur $f(s)$. Résoudre l'instance (Ω, f)du problème d'optimisation consiste à trouver la solution optimale $s^* \in \Omega$ qui optimise (minimise ou maximise) la valeur de la fonction objectif f. Pour le cas de la minimisation : le but est de trouver $s^* \in \Omega$ tel que $f(s^*) \leq f(s)$ pour tout élément $s \in \Omega$. Un problème de maximisation peut être défini de manière similaire

1.2.1 Variables de décision Les variables de décision sont des quantités numériques pour les quelles des valeurs sont à choisir. Cet ensemble de n variables est appelé vecteur de décision :$(x_1, x_2, ...x_n)$ Les différentes valeurs possibles prises par les variables de décision $x_i$constituent l'ensemble des solutions potentielles.

1.2.2 Espace décisionnel et espace objectif Deux espaces Euclidiens sont considérés en optimisation :

- L'espace décisionnel, de dimension n, n étant le nombre de variables de décision. Cet espace est constitué par l'ensemble des valeurs pouvant être prise par le vecteur de décision.
- L'espace objectif : l'ensemble de définition de la fonction objectif, généralement défini dans Â. La valeur dans l'espace objectif d'une solution est appelée coût, ou fitness.

1.2.3 Contraintes

Dans la plupart des problèmes d'optimisation, des restrictions sont imposées par les caractéristiques du problème. Ces restrictions doivent être satisfaites afin de considérer une solution acceptable. Cet ensemble de restrictions, appelées **contraintes** décrit les dépendances entre les variables de décision et les paramètres du problème.On formule usuellement ces contraintes c_j par un ensemble d'inégalités, ou d'égalités de la forme : $c_j(x_1, x_2, ...x_n) \geq 0$ [33].
un problème d'optimisation mono-objectif est plus souvent donnée sous la forme suivante :

$$\begin{cases} \text{Minimiser } f(\vec{x}) \\ \text{tel que } \vec{g}(\vec{x}) \leq 0 \\ \text{avec } \vec{x} \in R^n \ , \ \vec{g} \in R^q \end{cases}$$

tel que f est la fonction objectif et appellé aussi fonction de coût ou critére d'optimisation, $\vec{x}$ est l'ensemble des variables du problème ,$\vec{g}$ sont les contraintes.
En effet un problème de maximisation peut être aisément transformé en problème de minimisation en considérant l'équivalence suivante :
$maximiser f(\vec{x}) \Longleftrightarrow minimiser - f(\vec{x})$

1.2.1 Méthodes de résolution des problèmes d'optimisation mono-objectif

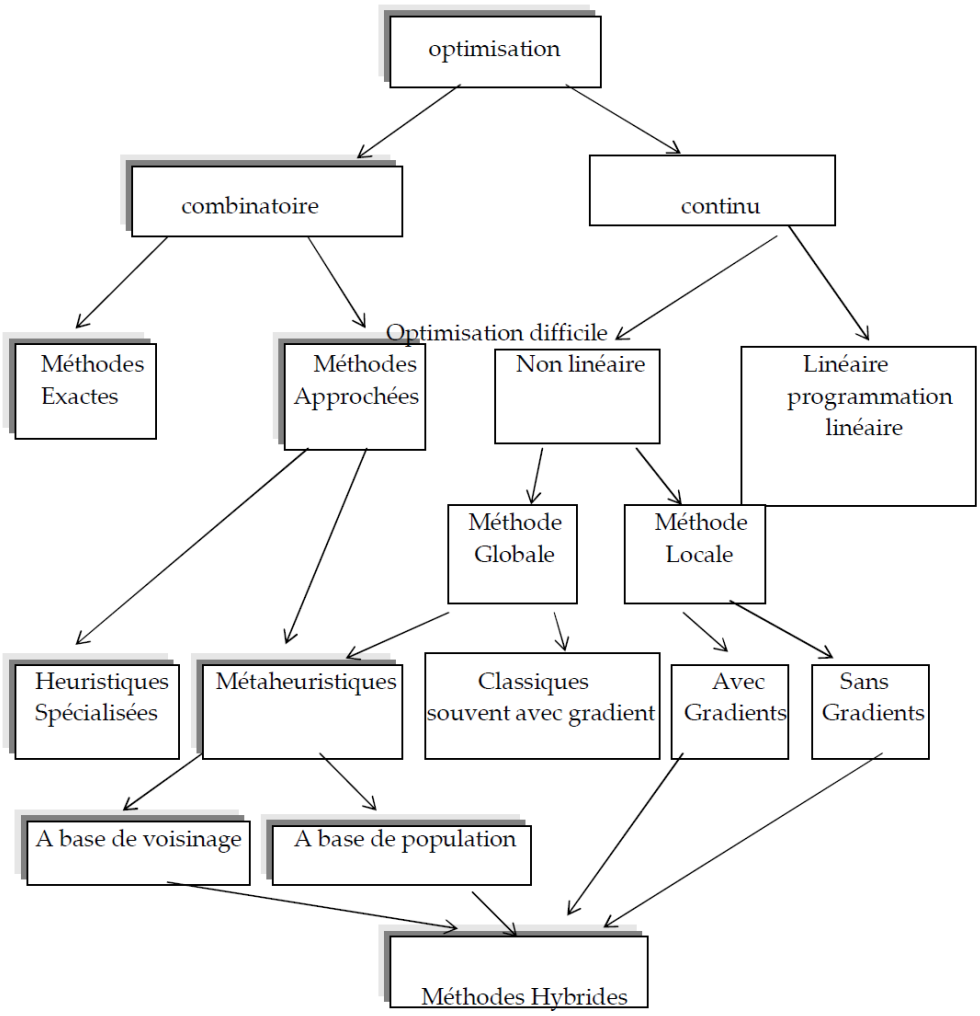


Fig 1 :Classification des méthodes d'optimisation mono-objectif [49].

Le schéma précédent présente une classification générale des méthodes de résolution des problèmes d'optimisation mono-objectif. On distingue en premier lieu l'optimisation continue de l'optimisation combinatoire. Pour l'optimisation continue, on sépare le cas linéaire du cas non linéaire, où l'on retrouve le cadre de l'optimisation difficile, dans ce cas on fait appel à une méthode locale qui exploite ou non les gradients de la fonction objectif. Si le nombre de minimums locaux est très élevé, le recours à une méthode globale s'impose : on retrouve alors les métaheuristiques. Pour l'optimisation combinatoire, on utilise les méthodes exactes. Lorsqu'on est confronté à un problème difficile on a recours aux méthodes approchées, dans ce cas le choix est parfois possible entre une heuristique spécialisée, dédiée au problème considéré, et une métaheuristique. Parmi les métaheuristiques, on peut différencier les métaheuristiques à base de voisinage, et les métaheuristiques à base de population. Enfin les méthodes hybrides associent souvent une métaheuristique et une méthode locale.

Les méthodes exactes

Les méthodes exactes sont des méthodes qui garantissent la complétude de la résolution autrement dit ces méthodes donnent à tous les coups la solution optimale. Le temps de calcul nécessaire de telles méthodes augmente en général exponentiellement avec la taille du problème à résoudre. On distingue dans ce cas l'approche constructive qui est probablement la plus ancienne et occupe traditionnellement une place très importante en optimisation combinatoire. Une méthode constructive construit pas à pas une solution de la forme $s = (< V_1, v_1 >< V_2, v_2 > \ldots < V_n, v_n >)$ en partant d'une solution partielle initialement vide $s = 0$, elle cherche à étendre à chaque étape la solution partielle $s = (< V_1, v_1 > \ldots < V_{i-1}, v_{i-1} >)(i <= n)$ de l'étape précédente. Pour cela, elle détermine la prochaine variable v_i, choisit une valeur v_1 dans Di et ajoute $< V_i, v_i >$ dans s pour obtenir une nouvelle solution partielle $s = (< V_1, v_1 > \ldots < V_n, v_n >< V_i, v_i >)$.Ce processus se répète jusqu'à ce que l'on obtienne une solution complète.
Durant la recherche d'une solution, la méthode constructive fait intervenir des heuristiques pour effectuer chacun des deux choix : le choix de la variable suivante et le choix de la valeur pour la variable. Les méthodes de cette classe diffèrent entre elles selon les heuristiques utilisées. En général,les heuristiques portent plus souvent sur le choix de variables que sur le choix de valeurs car les informations disponibles concernant le premier choix semblent souvent plus riches. La performance de ces méthodes dépend largement de la pertinence des heuristiques employées, c'est à dire,de leur capacité d'exploiter les connaissances du problème.

Un premier type de méthodes constructives est représenté par les méthodes gloutonnes. ces méthodes consistent à fixer à chaque étape la valeur d'une variable sans remettre en cause les choix effectués précédemment.
Un deuxième type de méthodes constructives est représenté par les méthodes avec retour arrière. Ces méthodes de retour arrière avec une stratégie de recherche en profondeur d'abord consistent à fixer à chaque étape la valeur d'une variable. Aussitôt qu'un échec est détecté, un retour arrière est effectué, c'est à dire, une ou plusieurs instanciations déjà effectuées sont annulées et de nouvelles valeurs recherchées. Les méthodes avec retour arrière sont en général complètes et de complexité exponentielle. Pour réduire le nombre de retour arrière (et le temps de recherche), on utilise des techniques de filtrage afin d'anticiper le plus tôt possible les échecs. Par exemple : ALICE, PROLOG III sont des systèmes de programmation sous contraintes fondés sur le principe de retour arrière.
Un troisième type de méthodes constructives est représenté par de nombreux algorithmes basés sur le principe de séparation et évaluation progressive, qui ont pour principe la construction d'un arbre de recherche dont le problème initial (problème de minimisation) est la racine. On divise le problème en sous problèmes (en deux ou plus) en introduisant par exemple une contrainte supplémentaire, qui peut être satisfaite ou non. L'optimum peut appartenir à l'un quelconque de ces sous problèmes. Tout sous problème infaisable sera éliminé. Si possible on calcule la solution du problème, sinon, on calcule une borne inférieure, si elle est supérieure de la meilleure solution déjà obtenue on élimine le sous-problème. Dans le cas restant, on subdivise à nouveau le sous problème. Pour améliorer l'efficacité de la recherche , on utilise des techniques variées pour calculer des bornes permettant d'élaguer le plus tôt possible des branches conduisant à un échec. Parmi ces techniques on peut citer : la relaxation de base en programmation linéaire et la relaxation lagrangienne .
Une autre méthode exacte, la méthode de programmation dynamique, c'est une méthode découverte par Bellman en 1956, elle c'est avérée une méthode très efficace pour la résolution des problèmes d'optimisation combinatoire, elle est conçu sur le modèle de l'algorithme du plus court chemin dans un graphe. Elle consiste à décomposer la résolution du problème initial en une suite de problèmes plus simples, la résolution du n-ème se déduisant de celle du (n-1)-ème par une équation récurrence.[36]
Plusieurs chercheurs ont tenté de résoudre les problèmes NP-difficiles par les méthodes exactes par exemple dans notre cas : pour la résolution du problème de l'emploi du temps en utilisant les approches basées sur la théorie des graphes ,on constate que ces approches partagent en commun le fait de s'appuyer sur les notions :
du nombre chromatique, l'indice chromatique ou le nombre de stabilité. Il s'agit alors

de modéliser le problème réel en un problème de coloration ou de recherche de sous graphes stables :

- Modélisation par coloration de graphe :

il s'agit de colorier les sommets d'un graphe avec un nombre minimum de couleurs (nombre chromatique du graphe) tels que deux sommets adjacents quelconques n'ont pas la même couleur. Dans ce cas les sommets correspondent aux enseignements et deux enseignements sont mis en correspondance s'ils ne peuvent, pour une raison ou une autre, se dérouler en même temps.Le nombre chromatique qu'on peut trouver correspond au minimum de périodes nécessaire pour la programmation de tous les enseignements. Chaque couleur correspondra alors à une période donnée.

- Modélisation par ensembles de stables :

un graphe est alors construit comme précédemment où les sommets représentent les enseignements et les arêtes les contraintes de non simultanéité. Il s'agit dans ce cas, de dégager un partitionnement des sommet du graphe en sous graphes stables de cardinalités inférieures à L. L désigne le nombre de locaux disponibles. Chaque sous graphe stable correspond à un sous ensemble d'enseignements qui doivent être planifiés à la même période.

Les approches de résolution se basant sur la théorie des graphes soufrent cependant de plusieurs lacunes, la plus importantes réside dans l'impossibilité de modéliser l'ensemble de toutes les contraintes.

D'autres méthodes exactes ont tenté aussi de résoudre le problème de l'emploi du temps tels que : les méthodes constructives [40]et de relaxation lagrangienne [48] .

Cependant, malgré les progrès réalisés au niveau des méthodes exactes, qui ont aidé à résoudre les problèmes de manière optimale, ces méthodes rencontrent généralement des difficultés face aux instances de taille importantes car la recherche d'une solution optimale peut être totalement inappropriée dans certaines applications pratiques en raison de la dimension du problème, de la dynamique qui caractérise l'environnement de travail, du manque de précision dans la récolte des données, de la difficulté de formuler les contraintes en terme explicites ou de la présence d'objectifs contradictoires.

Compte tenu de ces difficultés, la plupart des spécialistes de l'optimisation combinatoire ont orienté leur recherche vers le développement de méthodes heuristiques qui exploitent au mieux la structure du problème considéré. Cela a conduit à une avancée importante pour la résolution pratique de nombreux problèmes.

Les méthodes approchées

Contrairement aux méthodes exactes,les méthodes approchées ne procurent pas forcément une solution optimale, mais seulement une bonne solution (de qualité raisonnable) en un temps de calcul aussi faible que possible.
Une partie importante des méthodes approchées est désignée sous le terme de métaheuristiques. Plusieurs définitions d'une métaheuristique ont été proposées dans la littérature[50], cette définition est celle adoptée par le « *Metaheuristics Network* »[20] : « *A metaheuristics is a set of concepts that can be used to define heuristic méthods that can be applied to a wide set of different problems* ».
Plusieurs classifications des métaheuristiques ont été proposées ; la plupart distinguent globalement deux catégories : les méthodes à base de solution courante unique, qui travaille sur un seul point de l'espace de recherche à un instant donné, appelées méthodes à base de voisinage comme les méthodes de recherche locale(méthode de la descente), de recuit simulé et de recherche tabou, et les méthodes à base de population, qui travaillent sur un ensemble de points de l'espace de recherche, comme les algorithmes évolutionnaires et les algorithmes de colonies de fourmis.
a) - Les méthodes à base de voisinage :
Dans les problèmes d'optimisation où l'on cherche à optimiser une fonction objectif sur un espace de recherche donné, une petite perturbation sur un point de cet espace induit souvent une petite variation des valeurs de la fonction objectif en ce point. On déduit que les bonnes solutions ont tendance à ce trouver à proximité d'autres bonnes solutions, les mauvaises étant proches d'autres mauvaises solutions. D'où l'idée qu'une bonne stratégie consisterait à se déplacer à travers l'espace de recherche en effectuant de petits pas (petits changements sur le point courant) dans des directions qui améliorent la fonction objectif. Cette idée est la base d'une grande famille d'algorithmes appelée méthodes à base de voisinage ou de recherche locale.

Les méthodes de voisinage (ou méthodes de recherche locale) s'appuient toutes sur un même principe : elles résolvent le problème d'optimisation de manière itérative. Elles débutent avec une configuration initiale (souvent un tirage aléatoire dans l'espace des configurations), et réalisent ensuite un processus itératif qui consiste à effectuer un mouvement choisi par le mécanisme d'exploration en tenant compte de la fonction de coût. ce processus s'arrête et retourne la meilleure configuration trouvée quand la condition d'arrêt est réalisée. Cette condition d'arrêt peut porter sur le nombre d'essais effectués, sur une limite temporelle ou sur le degré de qualité de la meilleure configuration courante. Cette versatilité permet de contrôler le temps de calcul, la qualité de la solution optimale trouvée s'améliorant au cours du temps. De manière générale les opérateurs de recherche locale s'arrêtent quand une solution localement

optimale est trouvée, c'est à dire quand il n'existe pas de meilleure solution dans le voisinage.
Les méthodes de voisinage diffèrent essentiellement entre elle par le voisinage utilisé et la stratégie de parcours de ce voisinage.

b) - Les méthodes à base de population :

Les sciences de la vie et les processus naturels ont de tout temps fasciné les ingénieurs. Ces derniers n'hésitent pas à s'inspirer des structures et des mécanismes du monde vivant pour développer des objets artificiels utilisables dans des contextes variés. Dans le domaine de l'optimisation combinatoire, la complexité des phénomènes naturels a servi de modèle pour des algorithmes toujours plus sophistiqués constituant ainsi la base d'un nouveau champ de la programmation informatique en pleine effervescence. On peut distinguer deux grandes classes de techniques :

. les algorithmes évolutinnaires qui sont inspirés par des concepts issus de la théorie de l'évolution naturelle de Darwin .

. les algorithmes de colonies de fourmis qui sont inspirés de l'éthologie.
Ces deux techniques appartiennent donc à la classe des méthodes à base de population.

Les méthodes à base de population comme leur nom l'indique, travaillent sur une population de solutions et non pas sur une solution unique comme dans les méthodes à base de voisinage.

Les méthodes hybrides

Le mode d'hybridation qui semble le plus fécond concerne la combinaison entre les méthodes de voisinage et les méthodes évolutives. L'idée essentielle de cette hybridation consiste à exploiter pleinement la puissance de recherche de méthodes de voisinage et de recombinaison des algorithmes évolutionnaires sur une population de solutions. Un tel algorithme utilise une ou plusieurs méthodes de voisinage sur les individus de la population pendant un certain nombre d'itération ou jusqu'à la découverte d'un ensemble d'optima locaux et invoque ensuite un mécanisme de recombinaison pour créer de nouveaux individus.
Les algorithmes hybrides sont considérés parmi les méthodes les plus puissantes. Cette puissance réside dans la combinaison des deux principes de recherche fondamentalement différents comme on a vu dans le paragraphe précédent. Le rôle de la

méthode de voisinage est d'explorer en profondeur une région donnée de l'espace de recherche alors que la méthodes évolutive introduit des règles de conduite générales dans le but de guider la recherche au travers de l'espace de recherche. Dans ce sens, les opérateurs de combinaison ont un effet diversificateur bénéfique à long terme.

Cette approche d'hybridation a permis de produire beaucoup de travaux dans la littérature par exemple :
- les algorithmes mémétiques
- la méthode GRASP
- L'algorithme MA/PM (Memetic Algorithm with Population Management) [35]

1.3 problèmes classiques d'optimisation combinatoire

Un problème d'optimisation consiste à chercher une instanciation d'un ensemble de variables soumises à des contraintes, de façon à maximiser ou minimiser un critère. Lorsque les domaines de valeurs des variables sont discrets, on parle alors de problèmes d'optimisation combinatoire. Nous présentons rapidement ici quatre problèmes classiques d'optimisation combinatoire : le problème du sac-à-dos, le problème d'affectation, le problème du voyageur de commerce et le problème d'ordonnancement.

1.3.1 Problème du sac-à-dos

Le "problème du sac-à-dos" est un problème de sélection qui consiste à maximiser un critère de qualité sous une contrainte linéaire de capacité de ressource. Il doit son nom à l'analogie qui peut être faite avec le problème qui se pose au randonneur au moment de remplir son sacà-dos : il lui faut choisir les objets à emporter de façon à avoir un sac le plus "utile" possible, tout en respectant son volume.

Plus formellement, on peut le décrire de la façon suivante. Soit un ensemble de n éléments et une ressource disponible en quantité limitée , b .Pour j= 1à n,on note p_j le profit associé à la sélection de l'élément j et on note a_j la quantité de ressource que nécessite l'élément j, s'il est sélectionné. Les coefficients p_j et a_j prennent des valeurs positives pour tout $j = 1$ à n.Le problème du sac-à-dos consiste à choisir un sous-ensemble des n éléments qui maximise le profit total obtenu, en respectant la quantité de ressource disponible.[17]

On associe à chaque élément j une variable de sélection,x_j ,binaire, égale à 1 si j est sélectionné,égale à 0 sinon. Le profit total obtenu peut alors s'écrire

comme la somme :$\sum_{j=1}^{n} p_j.x_j$ et la quantité totale de ressource utilisée comme la somme :$\sum_{j=1}^{n} a_j.x_j$.Le problème du sacà-dos se modélise donc sous la forme :

$$\max \quad \sum_{j=1}^{n} p_j.x_j$$

$$s.c \sum_{j=1}^{n} a_j.x_j \leq b$$

$$x_j \in \{o, 1\} \forall j = 1..n$$

La contrainte de ressource est appelée "contrainte de sac-à-dos" ; on la retrouve dans des problèmes d'optimisation, issus de nombreux domaines d'application, qui mettent en jeu des ressources à capacité limitée.

Dans le cas où l'on a plusieurs contraintes de ce type (par exemple, le randonneur peut considérer non seulement un volume maximal, mais également un poids maximal, que son sac peut supporter), on parle de problème de sac-à-dos "multidimensionnel".

Le problème du sac-à-dos a fait l'objet de différents travaux proposant des méthodes exactes de résolution. Un état de l'art détaillé de ces approches est présenté dans [43]. Les algorithmes proposés relèvent de trois principaux types de méthodes. Premièrement, des algorithmes de type séparation et évaluation ont été proposés dans les années 70, permettant de traiter efficacement des instances de petite taille. Ces performances ont par la suite été améliorées par l'adjonction de contraintes supplémentaires pour renforcer les bornes dans l'arbre de recherche. Deuxièmement, des algorithmes se basant sur l'identification d'une variable critique et d'un sous-ensemble associé de variables, sur lequel on applique une recherche arborescente tronquée, ont permis, à partir des années 80,d'augmenter la taille des instances pouvant être résolues (jusqu'à $n = 100000$). Troisièmement, des algorithmes efficaces de programmation dynamique ont été proposés.En particulier, dans[42] la programmation dynamique est combinée avec l'identification d'une variable critique et l'utilisation de techniques de renforcement des bornes.

Concernant le problème de sac-à-dos multidimensionnel, nous renvoyons à la lecture de[8] qui détaille les différentes approches existantes.

1.3.2 Problème d'affectation

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

On considère m tâches et n agents, avec n≥m. Pour tout couple (i,j) (i= 1 à m ,j = 1 à n),l'affectation de la tâche i à j entraîne un coût de réalisation noté $c_{i,j}$ ($c_{i,j} \geq 0$).Chaque tâche doit être réalisée exactement une fois et chaque agent peut réaliser au plus une tâche. Le problème consiste à affecter les tâches aux agents, de façon à minimiser le coût total de réalisation et en respectant les contraintes de réalisation des tâches et de disponibilité des agents.[17]

À tout couple tâche/agent (i,j),on associe une variable d'affectation,$x_{i,j}$,binaire, qui prend la valeur 1 si la tâche i est affectée à l'agent j et 0 sinon. Le coût total de réalisation des tâches s'exprime alors par la somme : $\sum_{j=1}^{m}\sum_{j=1}^{n} c_{i,j}.x_{i,j}$ .Le nombre d'agents réalisant la tâche i est donné par :$\sum_{j=1}^{n} x_{i,j}$,pour tout $i = 1$ à m et le nombre de tâches réalisées par l'agent j est donné par :$\sum_{i=1}^{m} x_{i,j}$,pour tout $j = 1$ à n. On peut donc modéliser le problème d'affectation sous la forme :

$$\min \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{i,j}.x_{i,j}$$

$$s.c. \sum_{j=1}^{n} x_{i,j} = 1 \qquad \forall i = 1..m$$

$$\sum_{i=1}^{m} x_{i,j} \leq 1 \quad \forall i = 1..m, \forall j = 1..n$$

$$x_{i,j} \in \{o, 1\} \quad \forall i = 1..m, j = 1..n$$

Les contraintes de ce problème se retrouvent dans de nombreuses applications mettant en jeu des problèmes d'allocation de ressources. Elles sont généralement appelées "contraintes d'affectation".

En théorie des graphes, on peut se ramener à un "problème de couplage dans un graphe biparti"[14].On dit d'un graphe G qu'il est biparti si l'on peut diviser les sommets en deux ensembles X_1 et X_2 de telle sorte que toutes les arêtes dans le graphe joignent un sommet de X_1 à un sommet de X_2.
Un "couplage" dans un graphe biparti est un ensemble d'arêtes qui n'ont, 2 à 2, aucun sommet commun dans G.

En associant X_1 à l'ensemble des tâches, de cardinalité m et X_2 à l'ensemble des agents, de cardinalité n, une arête (i, j) dans le graphe G (avec $i \in X_1$ et $j \in X_2$) représente la possibilité d'affecter la tâche i à l'agent j ; on associe le poids $c_{i,j}$ à chaque arête (i, j) de G. Le poids d'un couplage étant défini comme la somme des poids de ses arêtes, le problème d'affectation revient alors à chercher un couplage de cardinalité m de poids minimal dans le graphe G.

Le cas particulier où X_1 et X_2 sont de même cardinalité (correspondant au cas $n = m$ pour le problème d'affectation) est fréquemment étudié ; on s'intéresse alors à la recherche d'un couplage de cardinalité maximale. Si on considère des ensembles X_1 et X_2 de cardinalité n et s'il existe n^2 arêtes dans le graphe G (le graphe biparti est complet), alors le couplage maximal est de cardinalité n et il est appelé "couplage parfait". On peut étendre ce problème à celui de la recherche d'un couplage maximal de poids minimal dans G.

Le problème d'affectation, ou de couplage dans un graphe biparti, peut être modélisé comme un problème de flot maximal à coût minimal dans lequel les capacités des arcs sont toutes égales à 1. C'est un problème classique de la théorie des graphes qui revient à chercher à faire passer un débit maximal à travers un réseau, pour un moindre coût .Des algorithmes simples et efficaces existent pour résoudre ce problème ; en particulier l'algorithme de Busaker et Gowen qui part d'un flot nul et qui l'augmente progressivement par recherche de "chaînes augmentantes" (i.e., chemins sur les arcs desquels on peut systématiquement augmenter le flot), de coût minimal.

La "méthode Hongroise", proposée par *Kuhn* en 1955, est un algorithme dual qui s'appuie sur une modélisation du problème d'affectation sous forme d'un programme linéaire, mais qui peut être vu comme une variante de l'algorithme de *Busaker* et *Gowen*, spécialisée pour la structure biparti du graphe. Du fait de sa grande efficacité sur ce type de problème, c'est l'algorithme de référence en Recherche Opérationnelle pour résoudre le problème d'affectation. Son principe est basé sur le fait que les couplages de poids minimal dans le graphe du problème primal sont exactement les couplages de cardinalité maximale dans le graphe du problème dual (voir par exemple [14] ou [34] pour une présentation détaillée de la méthode).

1.3.3 Problème d'ordonnancement

Le "problème d'ordonnancement" consiste à séquencer et placer dans le temps un ensemble d'activités (entités élémentaires de travail), compte tenu de contraintes temporelles (délais, contraintes d'enchaînement, . . .) et de contraintes portant sur l'utilisation et la disponibilité des ressources requises par les activités[37].Posé ainsi, il s'agit d'un problème de satisfaction de contraintes qui trouve ses applications dans divers domaines (gestion de projets, ateliers de production, . . .) et qui fait l'objet de travaux de recherche d'un point de vue de l'aide à la décision, notamment par des approches par contraintes[38]. Dans un contexte d'optimisation, on cherche de plus à minimiser (ou maximiser) un critère, comme par exemple la durée totale de réalisation des activités (minimisation du *Makespan*).

1.3.4 Problème du voyageur de commerce

Le "problème du voyageur de commerce", ou TSP (pour *Traveling Salesman Problem*), est le suivant : un représentant de commerce ayant n villes à visiter souhaite établir une tournée qui lui permette de passer exactement une fois par chaque ville et de revenir à son point de départ pour un moindre coût, c'est-à-dire en parcourant la plus petite distance possible. C'est un des problèmes les plus anciennement et largement étudiés en optimisation combinatoire. Ses applications sont nombreuses. Par exemple, des problèmes de séquencement de processus de fabrication ou d'optimisation de parcours en robotique peuvent s'exprimer directement sous forme d'un TSPet certains problèmes, comme les problèmes de transport, sont plus complexes que le TSPmais présentent une structure sous-jacente de type TSP.

Soit $G = (X, U)$, un graphe dans lequel l'ensemble X des sommets représente les villes à visiter, ainsi que la ville de départ de la tournée, et$U$, l'ensemble des arcs de G, représente les parcours possibles entre les villes. À tout arc $(i, j) \in U$, on associe la distance de parcours $d_{i,j}$ de la ville i à la ville j. La longueur d'un chemin dans G est la somme des distances associées aux arcs de ce chemin. Le TSP se ramène alors à la recherche d'un circuit hamiltonien (*i.e.*, un chemin fermé passant exactement une fois par chacun des sommets du graphe) de longueur minimale dans G. Dans le cas où il existe certains arcs $(i, j) \in U$ pour lesquels $d_{i,j} \neq d_{j,i}$, on parle de TSP asymétrique.[17]

On peut formuler le TSP de manière équivalente en associant à chaque couple (i,j) de villes à visiter ($i = 1$ à n, $j = 1$ à n et $i \neq j$) une distance $\delta_{i,j}$ égale à $d_{i,j}$ s'il existe un moyen d'aller directement de i à j (*i.e.*, $(i, j) \in U$ dans G), fixée à 1 sinon et une variable de succession, $x_{i,j}$, binaire, qui prend la valeur 1 si la ville j est visitée immédiatement après la ville i dans la tournée et qui prend la valeur 0 sinon. Le TSP est alors modélisé par :

$$\begin{array}{lcl} \min & \sum_{i=1}^{n} \sum_{j=1}^{n} \delta_{i,j} x_{i,j} & \\ s.c. & \sum_{j=1}^{n} x_{i,j} = 1 & \forall i = 1..n \\ & \sum_{i=1}^{n} x_{i,j} = 1 & \forall j = 1..n \\ & \sum_{i \in S, j \notin S} x_{i,j} \geq 2 & \forall S \subset X, S \neq \emptyset \\ & x_{i,j} \in \{o, 1\} & \forall i = 1..n, \forall j = 1..n \end{array}$$

Les deux premières contraintes traduisent le fait que chaque ville doit être visitée exactement une fois ; la troisième contrainte interdit les solutions composées de sous-tours disjoints, elle est généralement appelée contrainte d'élimination des sous-tours.

Des algorithmes de Programmation Linéaire en Nombres Entiers ont été développés pour résoudre de façon exacte le TSP.En particulier, des méthodes de recherche par séparation et évaluation sont renforcées efficacement par l'adjonction de coupes

(algorithmes de *branch*and*cut*) et de techniques de propagation de contraintes dans l'arbre de recherche. Des algorithmes de programmation dynamique pour la recherche de circuits hamiltoniens dans un graphe, ainsi que la Programmation Par Contraintes (PPC) fournissent également de bons résultats pour des problèmes allant jusqu'à une centaine de noeuds [14]. De plus, on dispose de procédures heuristiques et de techniques d'optimisation locale efficaces (par exemple, l'algorithme de *Lin* et *Kernighan*,voir[14]), ainsi que de calculs de bornes inférieures, basés par exemple sur une relaxation lagrangienne du TSP, permettant de fournir un bon encadrement de l'optimum.

Il existe de nombreuses variantes du TSP , obtenues soit par adjonctions de contraintes comme le TSP avec fenêtres de temps (TSPTW pour *Traveling Salesman Problem with Time Windows*) dans lequel la visite de chaque ville doit se faire dans un intervalle de temps donné, soit par modification, comme par exemple les problèmes de tournée de véhicules (VRPpour *Vehicule Routing Problem*) dans lesquels on ne considère plus un unique représentant de commerce pour visiter les villes mais une équipe (une flotte de véhicules) et qui peuvent être vu comme des problèmes de flot. Pour une présentation détaillée des différentes variantes du TSP, ainsi que des méthodes de résolution existantes, on peut se référer à un ouvrage spécialisé comme [16], ou à [15] pour une présentation plus synthétique.

• Les mèthodes de rèsolution le problrème du voyageur de commerce

Considérons un graphe non orienté et connexe, sur lequel deux sommets ont un rôle particulier : – source (ou origine) :O ; – puits (ou destination) : T. A chaque arête $\{i, j\}$, nous associons une distance $c_{ij} \geq 0$. Nous cherchons le chemin non orienté (ou chaîne) le plus court reliant O à T. Par chemin le plus court, nous désignons celui dont la distance totale (somme des distances des arêtes du chemin) est minimale parmi tous les chemins de O à T. Afin de procéder, nous affectons à chaque noeud du graphe une étiquette, représentant la meilleure distance connue à un moment donné, de l'origine à ce noeud.

· Algorithme de Dijkstra

Il s'agit d'une méthode itérative A chaque itération, nous choisissons le sommet j le plus près de O et nous fixons $d(j)$, la variable calculant la distance entre 0 et j (le sommet j est dit marqué). Au départ, seul O est marqué et $d(0) = 0$. Le sommet le plus près est choisi parmi les sommets non marqués reliés à au moins un sommet

marqué. Le sommet choisi j est celui qui atteint

$$\min_{\text{sommets } k \text{ non marqués}} \left\{ \min_{\text{sommets } i \text{ marqués}} d(i) + c_{ik} \right\}$$

$d(j)$ est fixée à cette valeur. Nous nous arrêtons lorsque T est marqué si nous cherchons à connaître le chemin allant de O à T spécifiquement, ou sinon jusqu'à avoir marqué tous les sommets. Par la suite, nous supposerons dans la description que nous voulons marquer tous les sommets. Plus spécifiquement, nous avons l'algorithme ci-dessous. L'algorithme de Dijkstra s'applique aussi sur un graphe orienté.[12]

Algorithme de Dijkstra

Soient $G = (N, A)$ un graphe connexe, de longueur d'arêtes non-négative (la longueur des arêtes est donnée par une fonction $\ell : A \longrightarrow R^+$, une source $O \in N$. Désignons par S l'ensemble des sommets marqués.

1. Initialisation. Pour tout $u \in N$, faire – $dist(u) := \infty$; – $pred(u) := NULL$. où $pred(u)$ indique le prédécesseur de u sur le plus court chemin de O à u. Le noeud origine prend un rôle particulier, et comme il est évident que le plus court chemin de O à O a une longueur nulle, nous posons $dist(O) = 0$. Comme à ce stade, aucun noeud n'a encore été marqué, nous posons $S = \emptyset$.

2. Tant que $S \neq N$, prenons u tel que

$$dist(u) \leq dist(v); \forall v \in N \backslash S.$$

Posons $S := S \cup \{\}$, et, pour tout $arc(u, v)$ existant, si

$$dist(v) < dist(u) + \ell(u, v)$$

posons

$$dist(v) := dist(u) + \ell(u, v)$$
$$pred(v) = u.$$

Il est possible de montrer qu'une fois un noeud u marqué, $dist(u)$ donne la longueur du plus court chemin de O à u.[12]

Exemple : Un étudiant en quête d'une université projette de visiter les campus de trois universités du Maine au cours d'un voyage unique, débutant et finissant à l'aéroport de *Portland*. Les trois établissements sont dans les villes de *Brunswick, Lewiston, et Waterville*, et l'étudiant ne veut visiter chaque ville qu'une seule fois, tout en maintenant le trajet total le plus court possible. Les distances entre ces villes sont données dans la Table 1.1.

ville	portland	Brunswick	Lewiston	Waterville
portland	0	26	34	78
Brunswick	26	0	18	52
Lewiston	34	18	0	51
Waterville	78	52	51	0

Table 1.1– Distances entre les villes (miles)

L'étape la plus importante dans la construction d'un modèle est le choix des variables qui vont entrer en jeu. Dans le présent cas, puisque n'importe quel trajet consiste en une série de petits déplacements entre deux villes, il est raisonnable d'assigner des variables aux décisions de partir ou non d'une ville vers une autre. Pour plus de faciliter, numérotons les villes comme suit : 1 pour Portland, 2 pour Brunswick, 3 pour Lewiston et 4 pour Waterville. Ainsi, nous aurons une variable $x_{1,2}$ égale à 1 si l'étudiant voyage de Portland à Brunswick au cours de son parcours total, et 0 sinon. Puisqu'il n'y a pas de voyage d'une ville vers cette même ville, nous avons d'ores et déjà les contraintes

$$x_{i,i} = 0, i = 1, ..., 4. \tag{1.1}$$

Une fois les variables choisies, nous pouvons essayer de formuler le problème. Ce processus est en fait souvent une manière utiles pour guider le choix des variables.

Chaque ville ne devant être visitée qu'une seule fois, elle ne peut apparaître qu'une seule fois comme ville d'arrivé. En d'autres termes, pour j fixé, $x_{i,j}$ ne peut être non-nul que pour un i donné, avec $i \neq j$. Une manière plus simple d'encoder cette information est d'écrire, pour $j = 1, ..., 4$,

$$x_{1,j} + x_{2,j} + x_{3,j} + x_{4,j} = 1$$

ou de manière plus concise.

$$\sum_{i=1}^{4} x_{i,j} = 1, j = 1, ..., 4. \tag{1.2}$$

Les contraintes formulées jusqu'à présent ne garantissent aucune forme de trajet ayant même départ et arrivée. Par exemple, l'affectation $x_{1,2} = 1, x_{1,3} = 1, x_{1,4} = 1, x_{2,1} = 1$, et toutes les autres variables égales à 0, satisfont les contraintes (1) et (2). Cette solution décrit toutefois un schéma de visites impossible puisque Portland est l'origine de tous les déplacements aux trois autres villes universitaires, mais

n'est destination que depuis *Brunswick*. Nous avons évidemment aussi besoin des contraintes

$$\sum_{j=1}^{4} x_{i,j} = 1, i = 1, ..., 4. \tag{1.3}$$

afin d'assurer que chaque ville ne serve d'origine que pour exactement un déplacement vers une autre ville. Finalement, afin d'obtenir un véritable trajet ayant même origine et départ, nous devons rejeter les affectations qui décrivent des groupes déconnectés de petits déplacements comme $x_{1,2} = x_{2,1} = 1, x_{3,4} = x_{4,3} = 1$, avec toutes les autres variables égales à 0. Nous pouvons forcer ceci avec les contraintes

$$x_{i,j} + x_{j,i} \leq 1, i = 1, ..., 4, etj = 1, ..., 4.$$

Cette contrainte exclut tout *mini-cycle*.

Les contraintes définies, nous devons décrire la distance totale associé à n'importe quel parcours autorisé. Puisque nos variables ont seulement comme valeurs possibles 0 ou 1, nous pouvons multiplier chacune d'elle par la distance correspondante entre les deux villes indexées, et les additionner :

$$\sum_{i=1}^{4} \sum_{j=1}^{4} x_{i,j} a_{i,j}.$$

Notre modèle mathématique consiste à minimiser cette fonction, dite fonction objectif par rapport aux variables $x_{i,j}$, tout en satisfaisant les contraintes préalablement décrites :

$$\begin{array}{cc} \min_x & \sum_{i=1}^{4} \sum_{j=1}^{4} x_{i,j} a_{i,j}, \\ \text{s.c.} & x_{i,i} = 0, i = 1, ..., 4, \\ & \sum_{i=1}^{4} x_{i,j} = 1, j = 1, ..., 4, \\ & \sum_{j=1}^{4} x_{i,j} = 1, i = 1, ..., 4, \\ & x_{i,j} + x_{j,i} \leq 1, i = 1, ..., 4, etj = 1, ..., 4, \\ & x_{i,j} \in \{o, 1\}, i = 1, ..., 4, etj = 1, ..., 4. \end{array}$$

Ici, $x = (x_{i,j})_{i=1,...,4,j=1,...,4}$, et s.c., "sous les contraintes". Le problème d'optimisation ainsi construit constitue un programme mathématique.

Le problème de visites d'universités est assez petit que pour être résolu explicitement, sans recourir à des méthodes d'optimisation numérique. Puisqu'il y a seulement trois parcours significativement différents, la distance totale associée à chacun

d'eux pourrait être facilement calculée, et nous choisissons le parcours de longueur minimale, qui est ici

$$Portland \longrightarrow Brunswick \longrightarrow Waterville \longrightarrow Lewiston \longrightarrow Portland.$$

avec une distance totale de 163 miles. Il est cependant clair qu'une telle stratégie de résolution ne fonctionne plus comme le nombre de villes augmente.

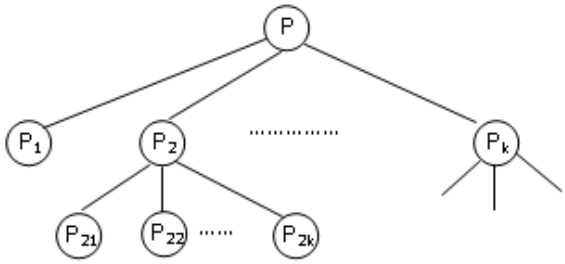
1.4 La Mèthode de Branch & Bound

1. **introduction**

   Il s'agit essentiellement de diviser (divide and conquer) l'ensemble de toutes les solutions réalisables (problème initial) en sous-ensembles plus petits (sous problèmes) et mutuellement exclusifs. C'est la phase « séparation » (Branch). Puis divers critères sont utilisés pour identifier les sous-ensembles qui peuvent contenir la solution optimale et les sous-ensembles qui ne doivent pas être explorés plus à fond car ils ne peuvent pas contenir la solution optimale. C'est la phase « évaluation » (Bound).

   Cette méthode consiste donc à faire une énumération intelligente de l'espace des solutions, puisqu'elle arrive à éliminer des solutions partielles qui ne mènent pas à la solution optimale.

   Pour ce faire, cette méthode se dote d'une fonction qui permet de mettre une borne inférieure (en cas de min) sur certaines solutions pour soit les exclure soit les maintenir comme des solutions potentielles. Bien entendu, la performance d'une méthode de branch and bound dépend, entre autres, de la qualité de cette fonction (de sa capacité d'exclure des solutions partielles tôt).



Le processus de séparation d'un sous-ensemble P_i s'arrête dans l'un des cas suivants (cas Min) :

- Lorsque la borne inférieure de P_i est $\geq$ à la meilleure solution trouvée jusqu'à maintenant pour le problème initial.
- Lorsque P_i n'admet pas de solution réalisable.
- Lorsque P_i admet une solution complète du problème initial.

2. **Algorithme général (cas Min)**

   A chaque instant, on maintient :

   - une liste de sous problèmes actifs P_i
   - Le coût Z de la meilleure solution obtenue jusqu'à maintenant (Initialisé à $+\infty$ ou à celui d'une solution initiale connue).

   A une étape typique :

   - Sélectionner un sous problème actif P_i
   - Si P_i n'est pas réalisable, le supprimer (stop branch) sinon calculer sa borne inférieure $Z_{inf}(P_i)$
   - Si $Z_{inf}(P_i) \geq Z$ supprimer P_i (stop branch)
   - Si $Z_{inf}(P_i) < Z$ soit résoudre P_i directement, soit créer de nouveaux sous problèmes et les ajouter à la liste des sous problèmes actifs.

   En pratique, il reste quelques petits détails à régler pour pouvoir appliquer cette méthode, en particulier :

   - Comment déterminer la borne d'évaluation (borne inférieure en cas de min) ?

   Une possibilité est de calculer la solution du PLC obtenue par relaxation linéaire du PLNE. Cette méthode donnera une bonne évaluation, mais pourra être coûteuse et longue à calculer. Suivant le problème on pourra essayer de trouver une méthode heuristique/astucieuse faisant un compromis entre vitesse d'obtention de la borne et sa fiabilité.

   - Quel sommet explore t'on à chaque étape de la recherche ?

   Là, il n'y a pas de bonne méthode, c'est suivant le problème en question. Les principales stratégies utilisées sont : - En profondeur (DFS)

   - En largeur (BFS)
   - Meilleur d'abord. (Best First)
   - Suivant quel xi construit-on l'arbre ?

   Ce choix peut être déterminant pour la rapidité de la solution optimale. Dans le cas du problème du sac à dos par exemple, il est plus efficace de prendre l'ordre des xi par coût décroissant.

1.5 La Méthode des coupes de Gomory (Gomory cutting planes)

La méthode des coupes (on dit également algorithme du plan sécant ou méthode des troncatures) développé par Ralph E. Gomory (1958) fournit une méthode de résolution des PLNE. L'idée principale est d'ajouter des contraintes qui n'excluent aucun point entier admissible. La méthode consistera à ajouter de telles contraintes linéaires une par une, jusqu'à ce que la solution optimale de la relaxation soit entière. Les contraintes ajoutées sont appelées troncatures ou coupes. On peut résumer la méthode comme suit :

- Etant donnée un PLNE, résoudre le PLC correspondant à l'aide du simplexe. Si la solution optimale obtenue contient uniquement des valeurs entières, la résolution est terminée.

- Si une ou plusieurs variables de base dans la solution optimale du PLC ne sont pas entières, on doit alors générer, à partir d'une des lignes du tableau (celle dont la partie fractionnaire est la plus élevée) une contrainte supplémentaire dite coupe de Gomory (qui n'exclue aucune solution réalisable entière).

- Après ajout de cette contrainte, on détermine une nouvelle solution à l'aide du dual simplexe. Le processus est répété jusqu'à ce qu'une solution optimale entière soit obtenue.

– **Génération de la Coupe de Gomory**

Notation :étant donné un réel $Y, [Y]$ désigne le plus grand entier inférieur ou égal à Y. Ainsi $Y = [Y] + f$, f étant la partie fractionnaire de y. Par exemple :

$$\begin{array}{ccc} Y = 2.7 & [Y] = 2 & f = 0.7 \\ Y = 16/5 & [Y] = 3 & f = 1/5 \\ Y = -8.1 & [Y] = -9 & f = 0.9 \\ Y = 2 & [Y] = 2 & f = 0 \end{array}$$

Pour générer la nouvelle contrainte (coupe de Gomory), on remplace tout simplement tous les coefficients dans la contrainte obtenue du tableau optimal par leur partie fractionnaire correspondante (f_{ij}) et déclarons par la suite que l'expression résultante doit être $\geq$ à la partie fractionnaire du second membre (f_i). On obtient la contrainte :

$$\sum f_{ij} x_j \geq f_i$$

Remarque : Plusieurs autres méthodes plus ou moins sophistiquées ont été conçues pour générer des coupes permettant d'obtenir plus ou moins rapidement la solution optimale du PLNE.

Chapitre 2

La méthode tabou pour résoudre les problèmes d'optimisation

2.1 introduction

Certains problèmes d'optimisation peuvent être résolus par des méthodes exactes simples, qui permettent de trouver la solution optimale. Mais la plupart des problèmes étudiés en optimisation appartiennent à la classe des problèmes NP-difficiles. La résolution de cette classe de problèmes via l'utilisation d'une méthode exacte n'est pas possible en un temps raisonnable. Dans ce cas, il est généralement plus judicieux de faire appel à des méthodes heuristiques pour les résoudre.[32] Une heuristique d'optimisation est une méthode approchée se voulant simple, rapide et adapté à un problème donné [22]. Sa capacité à optimiser un problème avec un minimum d'informations est contrebalancée par le fait qu'elle n'offre aucune garantie quant à l'optimalité de la meilleure solution trouvé.

Du point de vue de la recherche opérationnelle, ce défaut n'est pas toujours un problème, tout spécialement quand seule une approximation de la solution optimale est recherchée.Des heuristiques générales, appelées Méta-heuristiques sont conçues par différentes approches. Apparues dans les années 80, les Métaheuristiques forment un ensemble d'algorithmes, c'est-à-dire, une suite d'instructions effectuant une tâche donnée visant à résoudre une large gamme de problèmes d'optimisation difficile, pour lesquels on ne connaît pas de méthode classique plus efficace .la figure 1 résume le principe des Métaheuristiques.

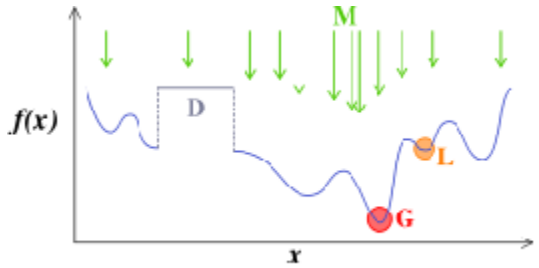


figure 1 : Principe des Métaheuristiques

La Métaheuristiques :(M) tente de trouver l'optimum global (G) d'un problème d'optimisation difficile (D), sans être piégé par les optimums locaux (L).

Le but d'une Métaheuristiques est de résoudre un problème d'optimisation donné, si l'on est capable d'attribuer une « qualité » à une solution du problème, alors la métaheuristique va rechercher la meilleure » solution en fonction de ce critère (on parle « *d'optimum global*»).

D'une manière générale, les Métaheuristiques s'articulent autour de trois notions : « Diversification / Exploration », « Intensification / Exploitation » et « Mémoire / Apprentissage ».*La diversification*-ou exploration- (synonyme utilisé indifféremment) désigne les processus visant à récolter de l'information sur le problème optimisé. *L'intensification* -ou exploitation- vise à utiliser l'information déjà récoltée pour définir et parcourir les solutions les plus prometteuses. La mémoire est le support de l'apprentissage, qui permet à l'algorithme de ne tenir compte que des zones où l'optimum global est susceptible de se trouver, évitant ainsi les optima locaux (de bonnes solutions, mais qui ne sont pas les meilleures des solutions possibles).

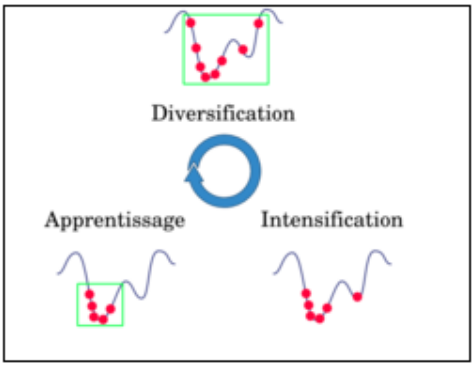


figure 2 : Les trois phases d'une métaheuristique

Les Métaheuristiques progressent de façon itérative, en alternant des phases d'intensification, de diversification et d'apprentissage. L'état de départ est souvent choisi aléatoirement, l'algorithme se déroulant ensuite jusqu'à ce qu'un critère d'arrêt soit atteint.
Ces méthodes sont souvent inspirées par des systèmes naturels, qu'ils soient pris en physique (cas du *recuit simulé* qui s'inspire de l'organisation des molécules dans un métal que l'on refroidit par étapes), en biologie de l'évolution (cas des *méthodes évolutionniste* fondés sur la théorie de l'évolution des êtres vivants) ou encore en éthologie (cas des algorithmes de *colonies de fourmis* ou de *l'optimisation par essaims particulaires*)[25].
Parmi les Métaheuristiques les plus utilisées, nous distinguons *le Recuit simulé* , la *Recherche Tabou* , les *méthodes évolutionnistes* et les *algorithmes de colonies de fourmis* .
ces dernières regroupent différents algorithmes basés sur le même principe d'exploration de l'espace de recherche en utilisant un ensemble de solutions et pas seulement une solution unique.
Comme représentantes des méthodes évolutionnistes, nous avons les Algorithmes Génétiques les stratégies d'Évolution, la programmation évolutionniste et la Programmation génétique.

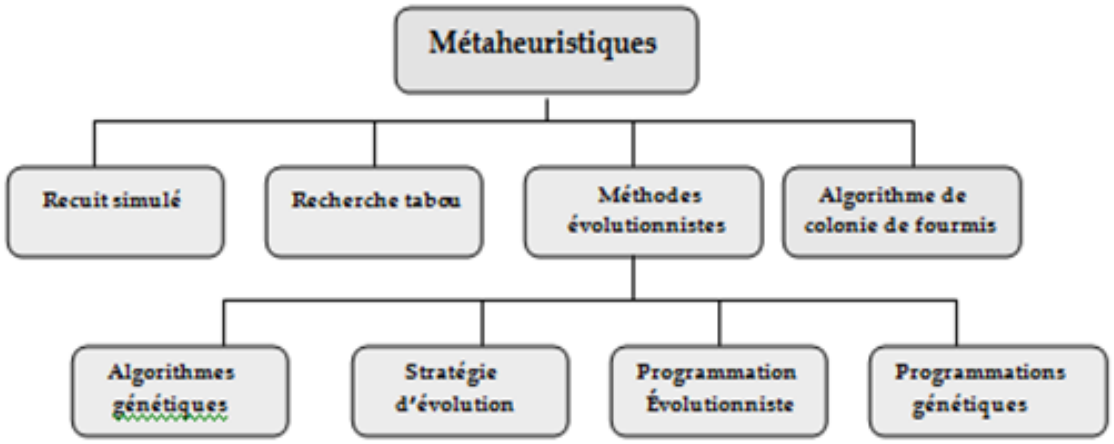


figure 3 :les principales Métaheuristiques-

2.2 Présentation de la méthode Tabou

La recherche taboue est une métaheuristique d'optimisation présentée par Fred Glover en 1986 [7] . On trouve souvent l'appellation recherche avec tabous en français. Cette méthode est une méta-heuristique itérative qualifiée de recherche locale au sens large.

2.3 Principe de la méthode Tabou

La procédure de la Recherche Tabou est destinée à trouver un optimum global d'une fonction f définie sur un ensemble de solutions réalisables S. Pour chaque solution i de S on définit un voisinage $N(i)$ constitué de toutes les solutions réalisables qui peuvent être obtenues par l'application d'une simple modification m sur i.

La procédure commence par une solution réalisable initiale et tente d'atteindre un optimum global du problème par un déplacement à chaque étape, dés qu'une solution réalisable est obtenue, on génère un sous -ensemble V de $N(i)$ et on se déplace vers la meilleure solution $i^{'}$ dans V. Si l'ensemble $N(i)$ n'est pas très large, il est possible de prendre $V = N(i)$ [5].

Liste taboue

Un élément fondamental de la recherche tabou est l'utilisation d'une mémoire flexible, à court terme, qui garde une certaine trace des dernières opérations passées. On peut y stocker des informations pertinentes à certaines étapes de la recherche pour en profiter ultérieurement. Cette liste permet d'empêcher les blocages dans les optima locaux en interdisant de passer à nouveau sur des solutions de l'espace de recherche précédemment visitées.

La durée de cette interdiction, notons la L, appelée longueur ou teneur tabou, est l'un des paramètres les plus importants et aussi l'un des plus difficiles à déterminer. Si sa valeur est trop élevée alors des solutions non visitées seront injustement inacces sibles et la capacité de la méthode à exploiter le voisinage sera réduit . Inversement, si sa valeur est trop faible alors la méthode risque fortement d'être bloquée dans un optimum local. La longueur taboue permet donc d'éviter tous les cycles de longueur inférieure ou égale à L . La valeur de L est fixée généralement à 7 [25] .

Critère d'Aspiration

liste Tabou pourrait jouer le rôle qui consiste à interdire de se déplacer vers des nouvelles régions susceptibles de contenir de meilleures solutions. Pour éviter cela et dans le but d'avoir une plus grande liberté dans la génération d'un sous- ensemble de solutions réalisables V , il devrait être possible de perdre le statut Tabou d'une modifi cation quand il semble raisonnable de le faire. C'est pourquoi on introduit

pour chaque valeur possible Z de la fonction objectif une valeur d'aspiration $A(Z)$. une solution i dans $N(i)$ qui veut devenir Tabou à cause de la liste Tabou peut quand même être prise en compte si $f(i) \geq A(f(i))$ (pour un problème de maximisation), la fonction A est appelée fonction d'aspiration.

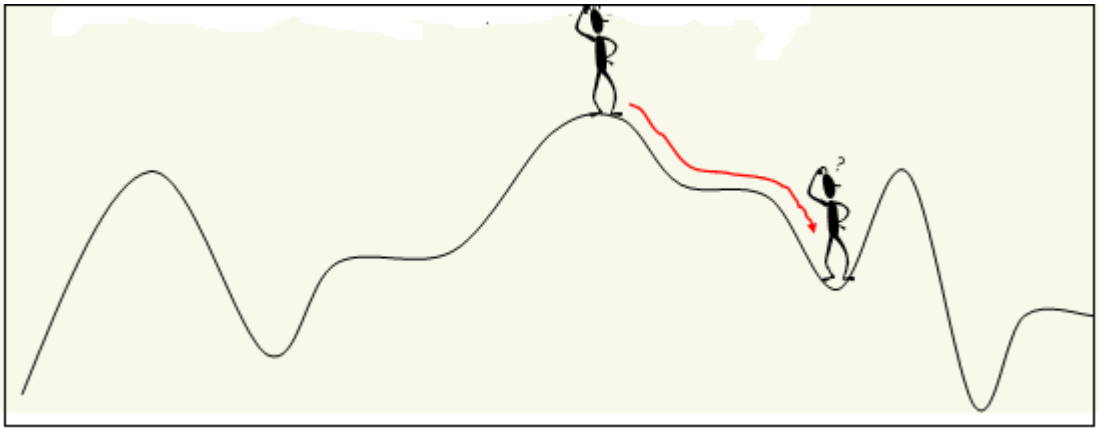
Critère d'arrêt

Comme critère d'arrêt on peut par exemple fixer un nombre maximum d'itérations, ou on peut fixer un temps limite après lequel la recherche doit s'arrêter.

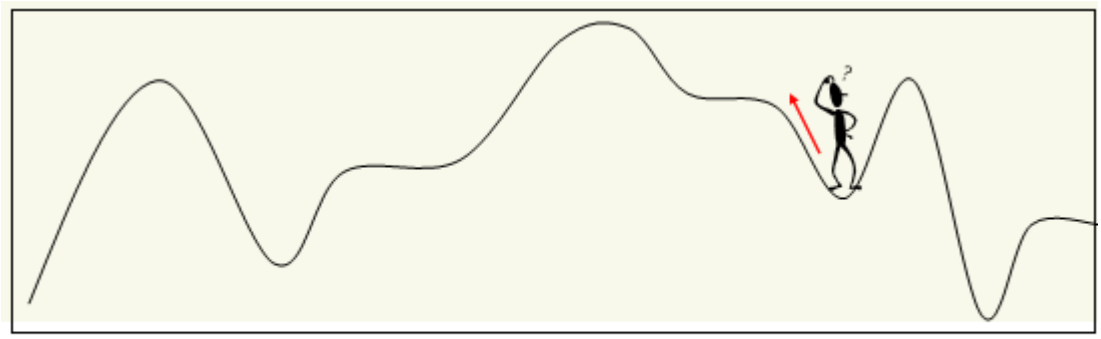
2.4 Mise en contexte

« Fable des randonneurs »

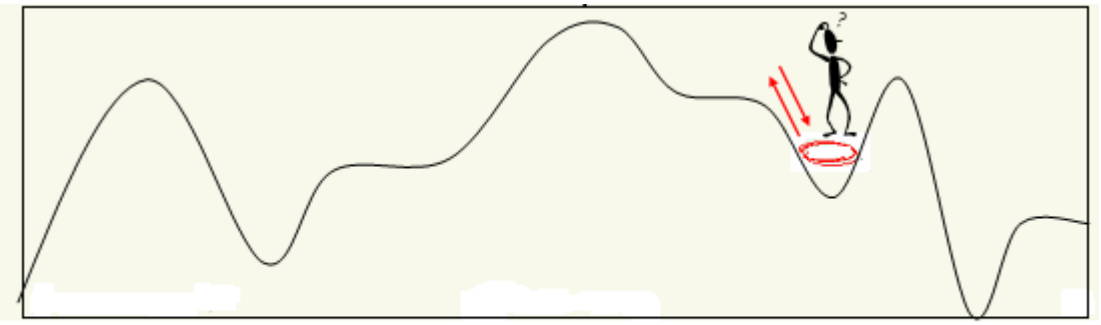
Un randonneur malchanceux , T. A. Bhoulx, est perdu dans une région montagneuse. Toutefois, il sait qu'une équipe de secours passe régulièrement par le point situé à la plus basse altitude dans la région. Ainsi, il doit se rendre à ce point pour attendre les secours. Comment s'y prendra-t-il ? Il ne connaît pas l'altitude de ce point et, à cause du brouillard, il ne voit pas autour de lui. Donc, arrivé à un croisement, il doit s'engager dans une direction pour voir si le chemin monte ou descend. Tout d'abord, il commence par descendre tant qu'il peut, en choisissant le chemin de plus grande pente à chaque croisement.[25]



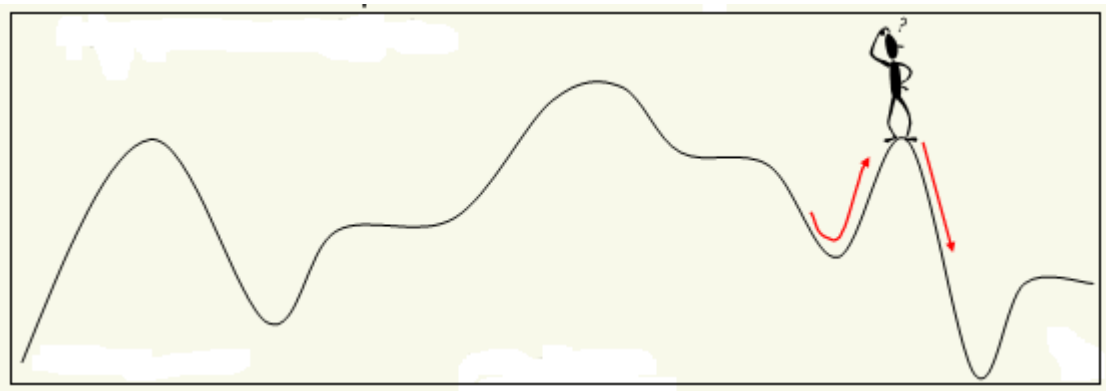
Puis, lorsqu'il n'y a plus de sentier menant vers le bas, il décide de suivre le chemin qui remonte avec la plus faible pente car il est conscient qu'il peut se trouver à un minimum local.



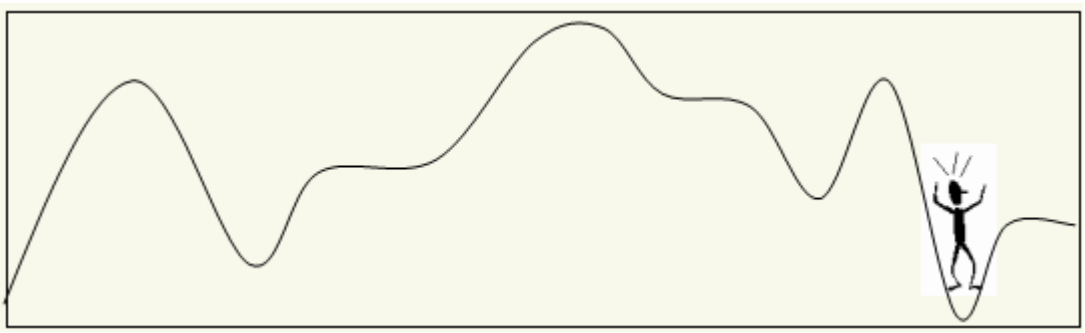
Toutefois, dès qu' il remonte, il redesce nd vers le point où il était. Cette straté gie ne fonctionne pas. Par conséquent, il décide de s'interdire de faire marche arrière en mémorisant la direct ion d'où il vient. Il est à noter que sa mémoire ne lui permet de mémoriser que les deux dernières directions prohibées.



Cette nouvelle stratégie lui permet d'explorer des minimum locaux et d'en ressortir. À un moment donné, il arrive à un point où il décèle une forte pente descendante vers le sud. Toutefois,les directions mémorisées lui interdisent d'aller vers le sud car cette direction est prohibée. Il décide d'ignorer cette interdiction et emprunte ce chemin [26].

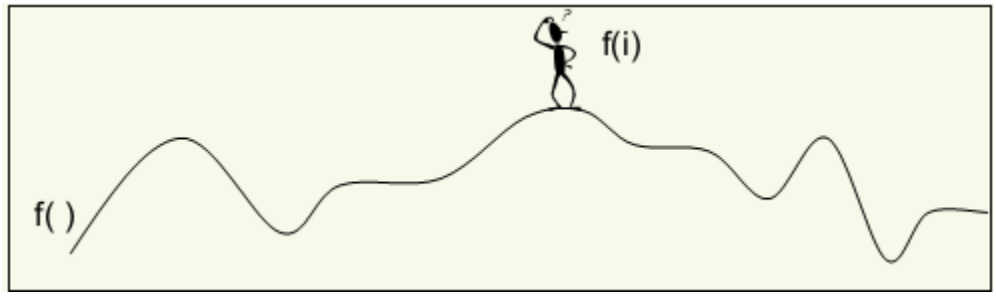


Cette décision fut bénéfique : il arriva au point de plus basse altitude et attendit les secours qui ne tardèrent à arriver.

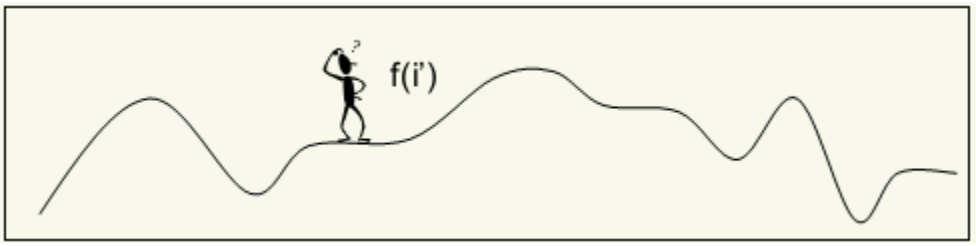


2.5 Définition des variables

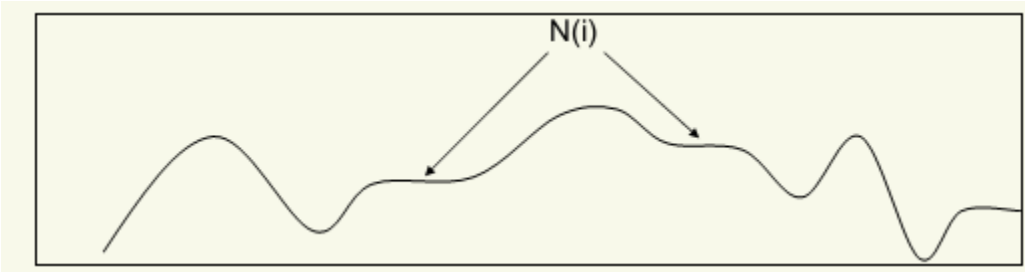
- i : la solution actuelle



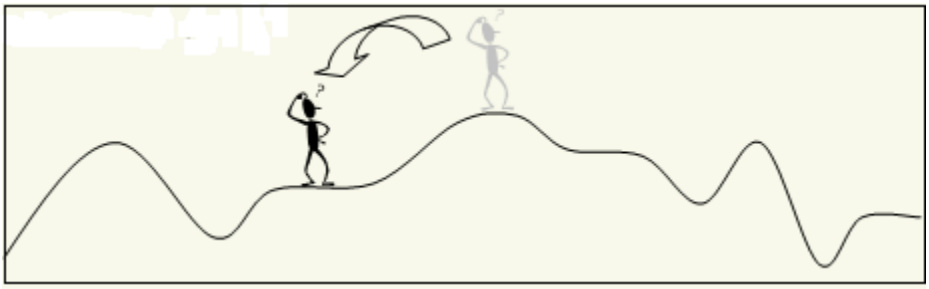
- $i^{'}$: la prochaine solution atteinte (solution voisine)



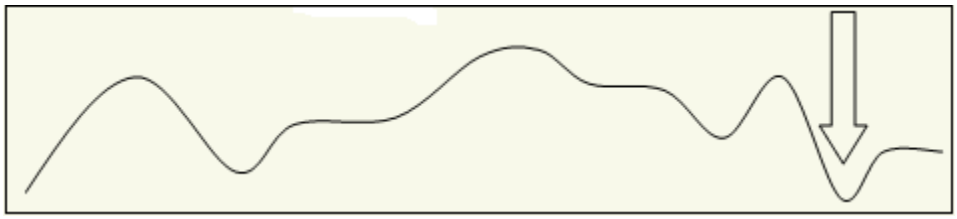
- $N(i)$: l'espace de solutions voisines à i (l'ensemble des $i^{'}$)



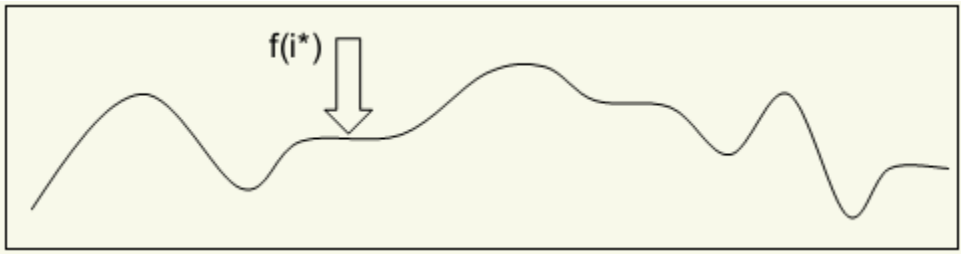
- m : mouvement de i à $i^{'}$



•$i_{globale}$: la solution optimale globalequi minimise la fonction objectif $f()$.

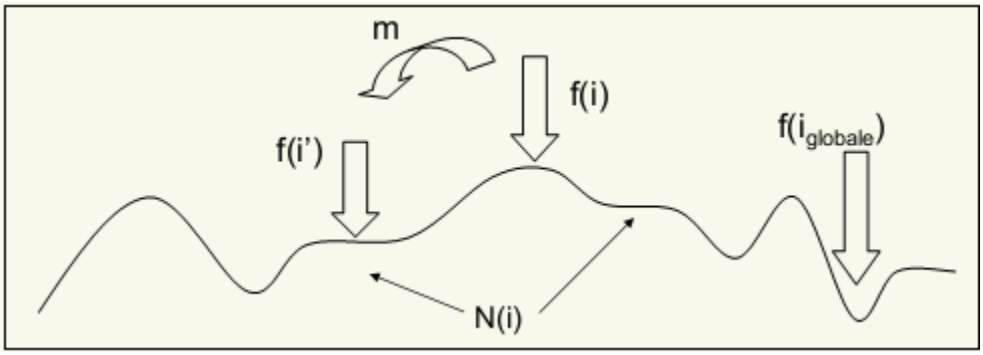


•i^* : la solution optimale actuelle



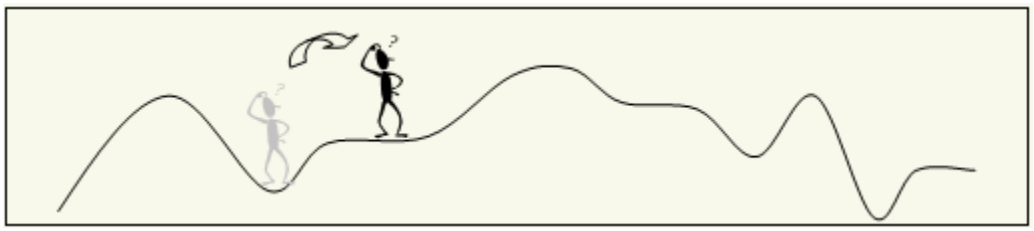
Résumé des variables

Et donc, jusqu'à présent, nous avons :



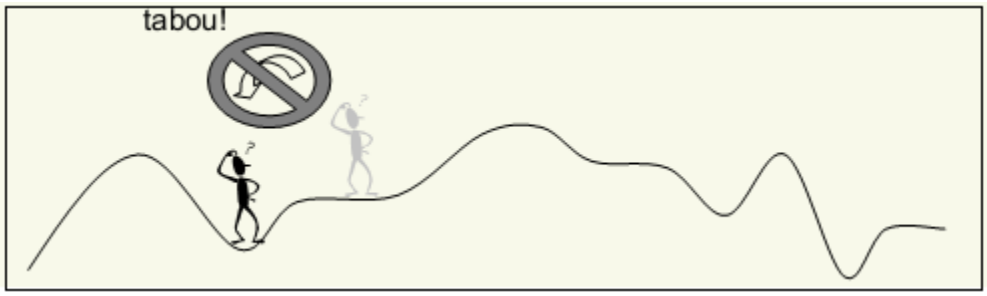
2.6 Définition des termes

• Mouvement non améliorateur : un mouvement qui nous sortirait d'un minimum local i^* en nous amenant à une solution voisine $i^{'}$ pire que l'actuelle [26].



La méthode tabou permet un mouvement non améliorateur, comme le permet le recuit simulé. La différence entre les 2 méthodes est que la RT choisira le meilleur $i^{'}$ dans $N(i)$, l'ensemble des solutions voisines.

• Mouvement tabou : un mouvement non souhaitable, comme si on redescendait à un minimum local d'où on vient juste de s'échapper.



Le mouvement est considéré tabou pour un nombre prédéterminé d'itérations.
k représente l'index des itérations (l'itération actuelle).

• T : liste des mouvements tabous. Il peut exister plusieurs listes simultanément. Les éléments de la liste sont $t(i, m)$. Une liste T avec trop d'éléments peut devenir très restrictive. Il a été observé que trop de contraintes (tabous) forcent le programme à visiter des solutions voisines peu alléchantes à la prochaine itération. Une liste T contenant trop peu d'éléments peu s'avérer inutile et mener à des mouvements cycliques.

•A : critères d'aspiration . Déterminent quand il est avantageux d'entreprendre m, malgré son statut tabou.

2.7 Algorithme général de la méthode Tabou

Étape 1 : choisir une solution initiale i dans S (l'ensemble des solutions)

Appliquer $i^* = i$ et $k = 0$

Étape 2 : appliquer $k = k+1$ et générer un sous-ensemble de solutions en $N(i, k)$

pour que :

– les mouvements tabous ne soient pas choisis

– un des critères d'aspiration A soit applicable

Étape 3 : choisir la meilleure solution $i^{'}$ parmi l'ensemble de solutions voisines

$N(i, k)$

Appliquer i = meilleur $i^{'}$

Étape 4 : si $f(i) \leq f(i^*)$, alors nous avons trouvé une meilleure solution

Appliquer $i^* = i$

Étape 5 : mettre à jour la liste T et les critères d'aspiration

Étape 6 : si une condition d'arrêt est atteinte, stop.

Sinon, retour à *Étape 2*

Condition d'arrêt : condition qui régira l'arrêt de l'algorithme.

ex : arrêt après 22 itérations ($k = 22$). [26]

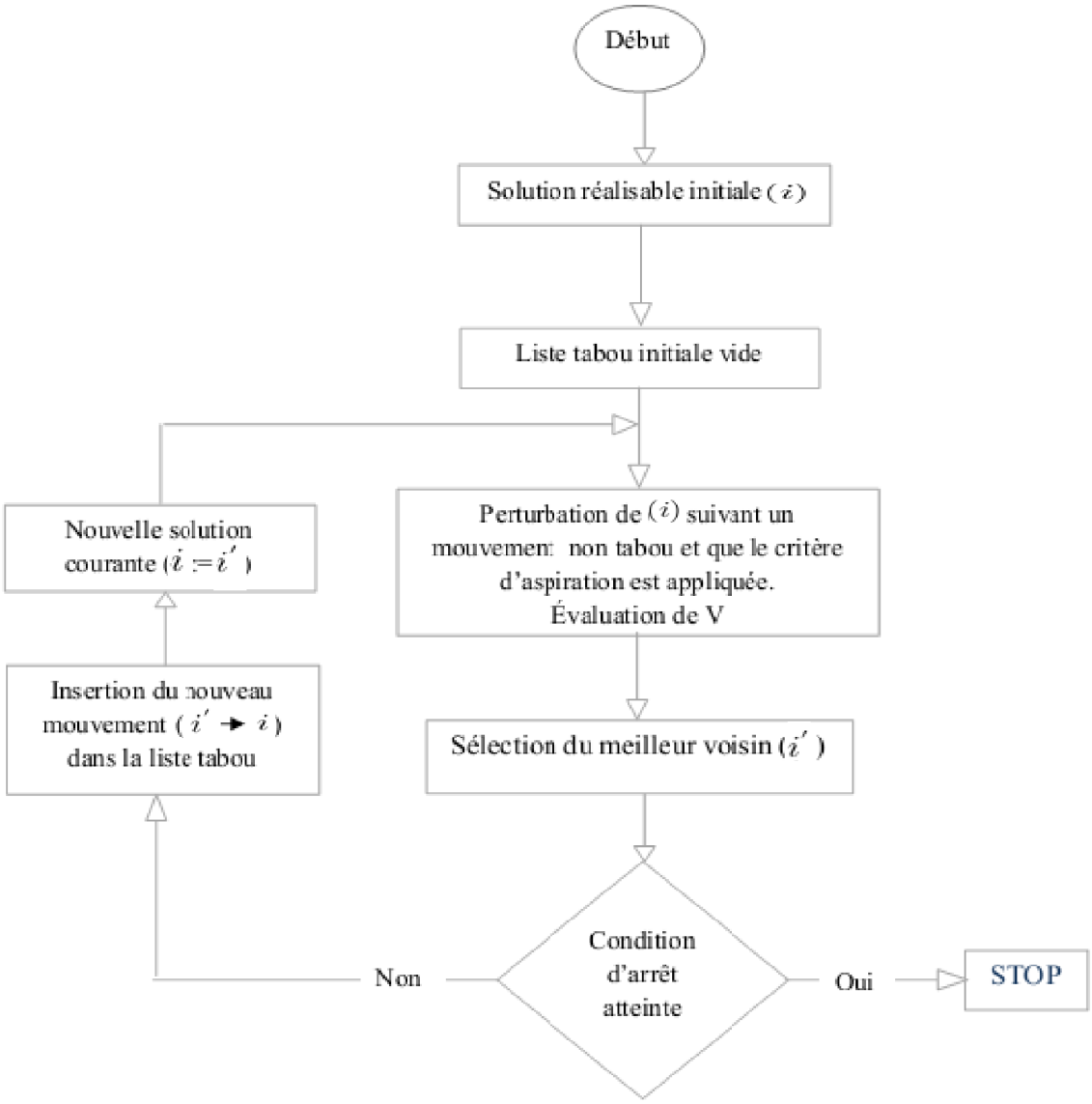


Figure - Organi gramme de la méthode Tabou .[28]

2.8 Améliorations

• La recherche de la solution optimale peut être améliorée. Voici quelques options :

– choix stratégique de la solution initiale i. Ceci donnera une « bonne » valeur $def(i^*)$

– Intensifier la recherche dans les voisinages de solutions qui semblent propices à mener à des solutions proches ou égales à l'optimum.

– Diversifier la recherche en éloignant celle-ci de voisinages
peu propices à produire de bonne solutions.

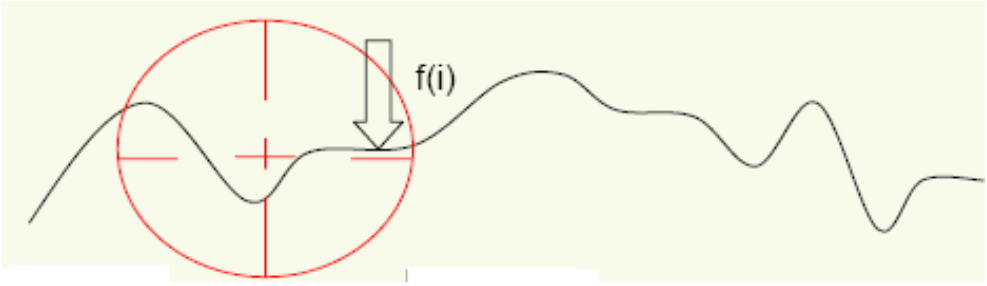
– concepts d'intensificati on et de diversification.

2.9 Intensification

La recherche est menée dans un voisinage$N(i)$ de S, l'ensemble des solutions

Une haute priorité est donnée aux solutions $f(i^{'})$ qui ressemblent à la solution actuelle $f(i)$

Le résultat est donc une intensification de la recherche dans un certain secteur, dans un voisinage choisi :

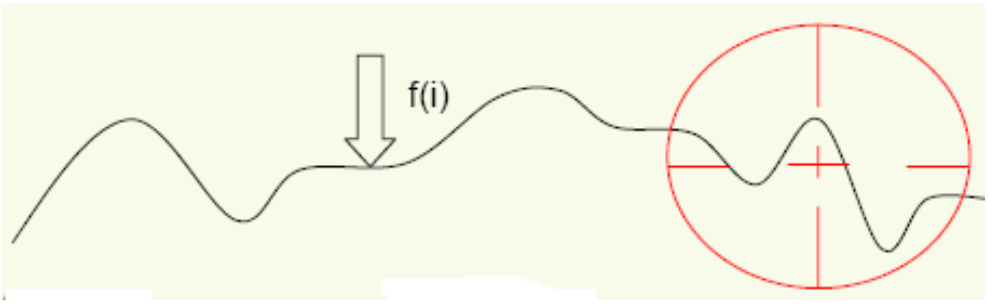


2.10 Diversification

La recherche est éloignée du voisinage $N(i)$ actuelle de l'ensemble des solutions

Une haute priorité est donnée aux solutions $f(i^{'})$ d'une autre région que celle actuellement sous exploration

Le résultat : chercher ailleurs



2.11 Modifications à $f()$

L'intensification et la diversification sont présentées comme des modifications à la fonction objectif.

Pour l'intensification , une « pénalité » est attribuée à des solutions éloignées de l'actuelle. Ceci cause un gonflement de la fonction objectif : les solutions semblables seront donc privilégiées. Pour la diversification, l'effet est le contraire. Les solutions proches de l'actuelle sont pénalisées.

Donc :$f = f + intensification + diversification$

Il est à noter que l'intensification et diversification se manifestent pendant quelques itérations k seulement, ensuite il y a alternance [26].

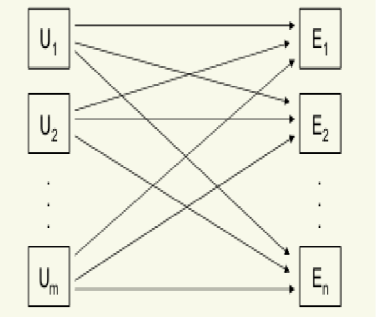
2.12 Exemple – Transport

• La recherche tabou peut être utilisée pour le problème de transport que nous connaissons si bien :

• m usines

• n entrepôts

• Un coût fixe et un coût variable associés à l'utilisation d'une route



• Une recherche menée par Sun et al. en 1995 cherchait à démontrer les avantages de la RT dans le problème de transport.

• L'utilisation des attributs de mémoire et de visites récentes.

• Algorithme de descente : simplex.

• 15 problèmes de différentes tailles étudiés.

• Résultats comparés avec un algorithme de solution exacte.

• Résultats :

- solution optimale trouvée: 12 en 15
- 3 autres, la solution < 0.06% de l'optimale
- vitesse avec RT: 1.63s de processeur (moyenne)
- vitesse sans RT: 5888s de processeur (moyenne)
- RT beaucoup plus efficace pour des problèmes de plus grande envergure

Chapitre 3

la programation de la méthode recherche tabou

3.1 Algorithme Générale De La Méthode Recherche tabou

TL : la liste tabou
It_Max : nombre Max d'itération permise sans amélioration
$S\_courante$, $S\_optimale$: Liste de tournées
$S\_courante \longleftarrow$ Générer solution initiale
$S\_optimale \longleftarrow S\_courante$
Générer le voisinage de la solution optimale
$It \longleftarrow 0$
Tant que $It \leq It\_Max$ faire

$S_{courante} \longleftarrow$ Meilleur voisin non tabou

Si $Cot(S_{courante}) < Cot(S_{optimale})$ alors
$S\_optimale \longleftarrow S\_courante$
Si Taille(TL) = TailleMax(TL) alors
Retirer la tête de la liste TL
Fin si
Stocker $S_{optimale}$dans TL

$It \longleftarrow 0$

Sinon

$It \longleftarrow It + 1$

Fin si

Fin tant que

NB :on choisir le meilleur voisin selon le problème. Par exemple dans le problème de TSP le meilleur voisin c'est le permutation des villes

3.2 le code source de l'algorithme recherche tabou qui résoudre le pvc sous Matlab

iter=1000 ;%number of cycles.
x=[6 3 10 2.5 -5 7 9 11 13 12 20 16 14 13 1 8 15 10 8 0 -1 2 4 7 8 9 10 0 2 9 4] ;
y=[2 4 6 -1 -2 0.5 0 3.7 1.8 1 0 4 3 2 14 2 3 6 9 4 5 23 5 0 9 1 -4 0 6 -1 4] ;
$n = 31$;%take care not to enter iterative points.
$n = length(x)$;%number of nodes.
$maxtabou = 100$;
$listabou = zeros(maxtabou, n)$;
$it = 1$;
for $i = 1 : n$ %generating link length matrix.

for $j = 1 : n$

$d(i, j) = sqrt((x(i) - x(j))^2 + (y(i) - y(j))^2)$;

end

end
%**********sol initial et optimal
$soptimal(1) = 1$;
$itt = 1; i = 1$;
while $itt < n$

for $j = 1 : n$

$ad(j, i) = 999999999999999999999999999999999999999999999$;

```
        end
        x = min(ad(i,:));
        y =find(ad(i,:) == x);
        i = y(1);
        itt = itt + 1;
        soptimal(itt) = i;
        y = [];
end
soptimal coutinitial = 0;
for k = 1 : n−1

            villcourant = soptimal(k);
            villsuivant = soptimal(k + 1);
            coutinitial = coutinitial + d(villcourant, villsuivant);
   end
coutinitial = coutinitial + d(soptimal(n), soptimal(1));
coutsoptimal = coutinitial;
coutoptimal1(1) = coutinitial;
coutinitial

    %**********fin de sol initial et coutinitial

    while it < iter

       vill1 = soptimal;
       for i = 1 : n − 1

            p = vill1(i);
            vill1(i) = vill1(i + 1);
            vill1(i + 1) = p;
                for j = 1 : n
```

$permutation(i,j) = vill1(j);$

end

$vill1 = soptimal;$

end

$p = vill1(n);$

$vill1(n) = vill1(1);$

$vill1(1) = p;$

for $j = 1 : n$

$permutation(n,j) = vill1(j);$

end

for $i = 1 : n * 10$

$x = randperm(n);$

$tab(i,:) = x;$

end

$Q = [permutation; tab];$
$permutation = Q;$
$[ligperm, colperm] = size(permutation)\,;$

%***************couts des permutations*******************

for $i = 1 : n + n * 10 - 1$
$coutpermutation = 0;$

for $j = 1 : n - 1$
coutpermutation=coutpermutation+d(permutation(i,j),permutation(i,j+1)) ;
end

$coutpermutation = coutpermutation + d(permutation(i,n), permutation(i,1));$

```
    cout_voisinage(i) = coutpermutation;
end
%*****************************************************************
minvoisinage = min(cout_voisinage)

    if (minvoisinage < coutsoptimal)
        [ligtab, coltab] = size(listabou);
        if ligtab == maxtabou
            listabou(1,:) = [];
        end
        listabou = [listabou; soptimal];

        jj = 1;

        trouve = false;

        while ((jj <= n)&(trouve == false))
            if cout_voisinage(jj) == minvoisinage
                trouve = true;
            else
                jj = jj + 1;
            end
         soptimal = permutation(jj,:);
         end
        coutsoptimal = minvoisinage
    end
    it = it + 1;
    coutoptimal1(it) = coutsoptimal;

end
k = 1 : iter;
plot(k, coutoptimal1(k))
```

• Définition des variables

$iter$:le nombre des itérations.

Les vecteurs x et y sont les coordonners des villes.

n :le nombre des villes.

$listabou$:la liste tabou

$maxtabou$:la taille maximale de la liste tabou.

$[ligtab, coltab]$:le nombre des lignes et des colonnes de la liste tabou.

$[ligperm, colperm]$:le nombre des lignes et des colonnes de la matrice de permutation.

- l'algorithme :
for $i = 1 : n$

for $j = 1 : n$

$d(i, j) = sqrt((x(i) - x(j))^2 + (y(i) - y(j))^2)$;

end

end
qui calculer les ditance entre les villes.

- l'algorithme de Dijkstra :
$soptimal(1) = 1$;
$itt = 1; i = 1$;
while $itt < n$

for$j = 1 : n$

$ad(j, i) = 9999999999999999999999999999999999999999999999$;

end

$x = min(ad(i, :))$;

$y =$find$(ad(i, :) == x)$;

$i = y(1)$;

$itt = itt + 1$;

$soptimal(itt) = i;$

$y = [];$

end
soptimal
qui calculer la solution initiale

- l'algorithme :
for $k = 1 : n-1$

$villcourant = soptimal(k);$

$villsuivant = soptimal(k+1);$

$coutinitial = coutinitial + d(villcourant, villsuivant);$

end
qui calculer le coût initiale (la somme des distance entre les villes dans la solution initiale).

$coutinitial = coutinitial + d(soptimal(n), soptimal(1));$
elle est calculer la distance entre la dernière ville et la première.

while $it < iter$:qui contrôler le condition d'arrêt.

- l'algorithme :

vill1 = soptimal ;
for $i = 1 : n - 1$

$p = vill1(i);$

$vill1(i) = vill1(i+1);$

$vill1(i+1) = p;$

for $j = 1 : n$

$permutation(i, j) = vill1(j);$

end

$vill1 = soptimal;$

end

$p = vill1(n);$

$vill1(n) = vill1(1);$

$vill1(1) = p;$

for $j = 1 : n$

$permutation(n, j) = vill1(j);$

end

for $i = 1 : n * 10$

$x = randperm(n);$

$tab(i, :) = x;$

end

$Q = [permutation; tab];$
$permutation = Q;$
qui permuter les villes dans la solution optimale deux à deux et les met ensuite dans une matrice qui doit être le voisinage de la solution optimale.

- l'algorithme :
for $i = 1 : n - 1$
$coutpermutation = 0;$

for $j = 1 : n - 1$

coutpermutation=coutpermutation+d(permutation(i,j),permutation(i,j+1)) ;

end

$coutpermutation = coutpermutation + d(permutation(i, n), permutation(i, 1));$

$cout\_voisinage(i) = coutpermutation;$

end

$minvoisinage = min(cout\_voisinage)$

qui calculer le coût des permutation (la somme des distance entre les ville dans les permutations).En suite choisir le minimum dans le voisinage

- l'algorithme :

if $(minvoisinage < coutsoptimal)$

$[ligtab, coltab] = size(listabou);$

if $ligtab == maxtabou$

$listabou(1,:) = [];$

end

$listabou = [listabou; soptimal];$

$jj = 1;$

$trouve = false;$

while $((jj <= n)\&(trouve == false))$

if $cout\_voisinage(jj) == minvoisinage$

$trouve = true;$

else

$jj = jj + 1;$

end

$soptimal = permutation(jj,:);$

end

$coutsoptimal = minvoisinage$

end
$it = it + 1$;
$coutoptimal1(it) = coutsoptimal$;

end
qui comparer le coût minimum voisinage par le coût de la solution optimale précédent ,et en suite elle est tester la liste tabou, si elle est complet ou non, et il est chercher la solution qui corresponde le minimum voisinage dans la matrice permutation(voisinage de la solution).

3.3 Les Exemple

Avec n villes, il y a $(n-1)!$ possibilités.

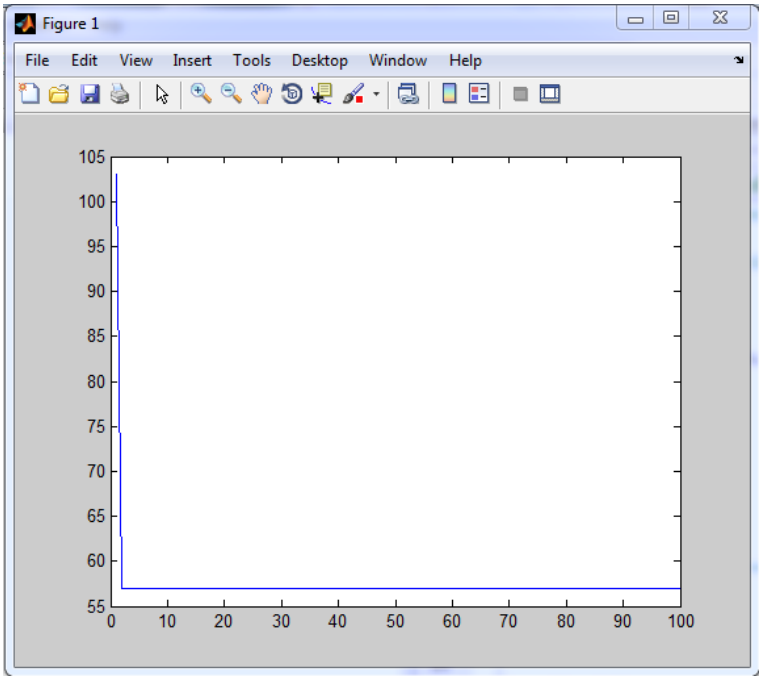
Exemple 1 :

Nombre de villes : $n = 5$. Il y a 24 possibilités.

ville	1	2	3	4	5
1	0	17	70	8	15
2	17	0	19	25	2
3	70	19	0	20	10
4	8	25	20	0	4
5	15	2	10	4	0

-Table de distance entre les villes de l'exemple 1.

-Algorithme de Dijkstra :
Parcours : 1 4 5 2 3 1.
Distance = 103.
Algorithme de colonies de fourmis pour TSP :
Parcours : 3 5 2 1 4 3.
Distance =57.



-Le graphe qui représenter la solution de l'exemple 1 par l'algorithme RT.

-L'algorithme RT pour TSP :
Parcours : 4 3 5 2 1 4.
Distance =57.

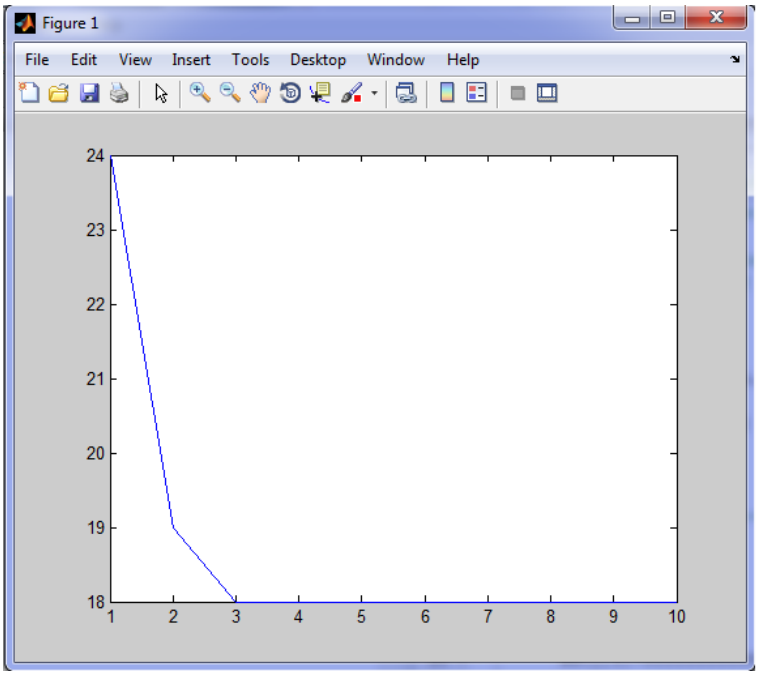
Exemple 2 :

Nombre de villes : $n = 6$. Il y a 120 possibilités.

ville	1	2	3	4	5	6
1	0	5	8	4	3	2
2	5	0	4	2	1	3
3	8	4	0	9	5	4
4	4	2	7	0	9	8
5	3	1	5	9	0	4
6	2	3	4	8	4	0

-Table de distance entre les villes de l'exemple 2.

-Algorithme de Dijkstra :
Parcours : 1 6 2 5 3 4 1.
Distance = 24.
Algorithme de colonies de fourmis pour TSP :
Parcours : 5 2 4 1 6 3 5.
Distance =18.



-Le graphe qui représenter la solution de l'exemple 2 par l'algorithme RT.

-L'algorithme de RT pour TSP :
Parcours : 5 1 4 6 3 2 5.
Distance =18.

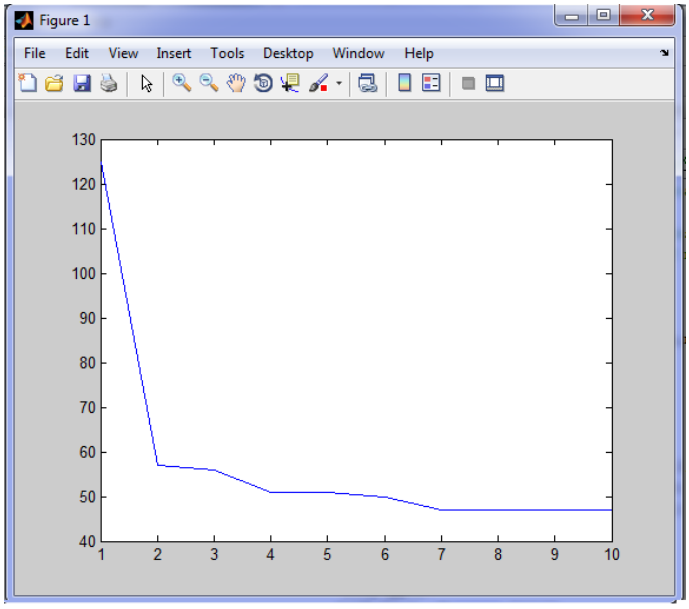
Exemple 3 :

la matrice des distances suivante (matrice carrée symétrique) :

ville	A	B	C	D	E	F	G
A	0	39	48	1	10	30	5
B	39	0	17	44	3	12	9
C	48	17	0	7	4	53	21
D	1	44	7	0	11	10	6
E	10	3	4	11	0	12	2
F	30	12	53	10	12	0	15
G	5	9	21	6	2	15	0

-Table de distance entre les villes de l'exemple 3.

Algorithme de Dijkstra :
Parcours A D G E B F C A.
Distance = 125.
Algorithme de colonies de fourmis pour TSP :
Parcours : C E G A D F B C.
Distance =51.



-Le graphe qui représenter la solution de l'exemple 3 par l'algorithme RT.

Algorithme de RT pour TSP :
parcours E F D A C B G E.
Distance = 47.

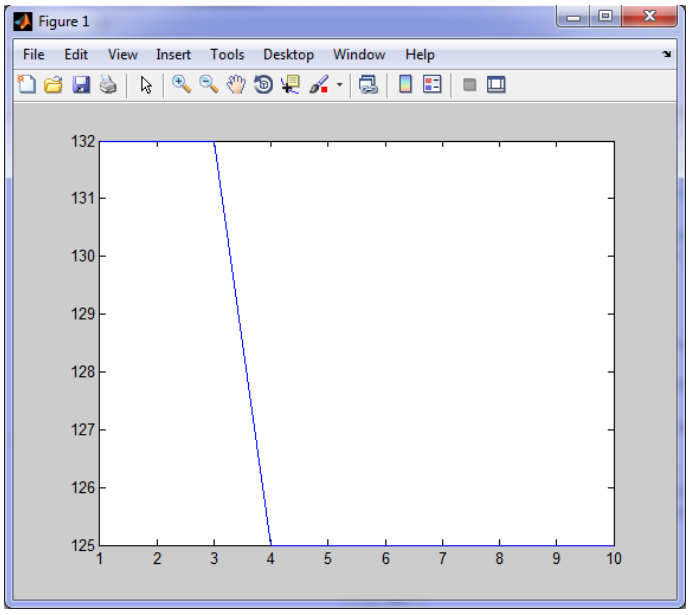
Exemple 4 :

Nombre de villes : $n = 8$. Il y a 5040 possibilités.

ville	1	2	3	4	5	6	7	8
1	0	9	17	40	32	41	49	8
2	9	0	47	26	34	23	15	58
3	17	47	0	27	35	22	14	59
4	40	26	27	0	29	44	52	5
5	32	34	35	29	0	45	53	4
6	41	23	22	44	45	0	11	62
7	49	15	14	52	53	11	0	63
8	8	58	59	5	4	62	63	0

-Table de distance entre les villes de l'exemple 4.

-Algorithme de Dijkstra :
Parcours : 1 8 5 4 2 7 6 3 1.
Distance =132.
Algorithme de colonies de fourmis pour TSP :
Parcours :2 1 8 5 4 3 7 6 2 .
Distance =125.



-Le graphe qui représenter la solution de l'exemple 4 par l'algorithme RT.

-L'algorithme RT pour TSP :
Parcours : 8 4 6 3 7 1 5 2 8.
Distance = 125.

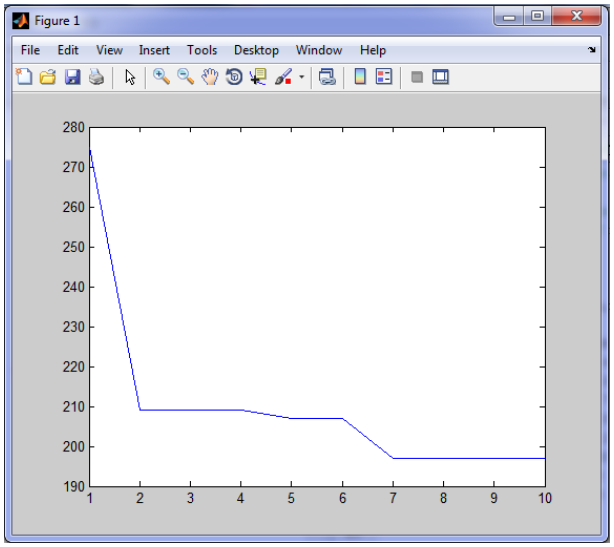
Exemple 5 :

Nombre de villes : $n = 9$. Il y a 40320 possibilités.

ville	1	2	3	4	5	6	7	8	9
1	0	57	18	77	3	16	80	36	120
2	57	0	90	7	17	27	28	38	48
3	18	90	0	18	19	29	39	49	59
4	77	7	18	0	30	40	50	60	70
5	3	17	19	30	0	51	61	103	81
6	16	27	29	40	51	0	72	73	2
7	80	28	39	50	61	72	0	150	13
8	36	38	49	60	103	73	150	0	24
9	120	48	59	70	81	2	13	24	0

-Table de distance entre les villes de l'exemple 5.

-Algorithme de Dijkstra :
Parcours : 1 5 2 4 3 6 9 7 8 1.
Distance =275.
Algorithme de colonies de fourmis pour TSP :
Parcours :4 2 5 1 6 9 7 3 8 4 .
Distance =206.



-Le graphe qui représenter la solution de l'exemple 5 par l'algorithme RT.

-L'algorithme RT pour TSP :
Parcours : 5 4 7 2 3 8 1 9 6 5.
Distance = 197.

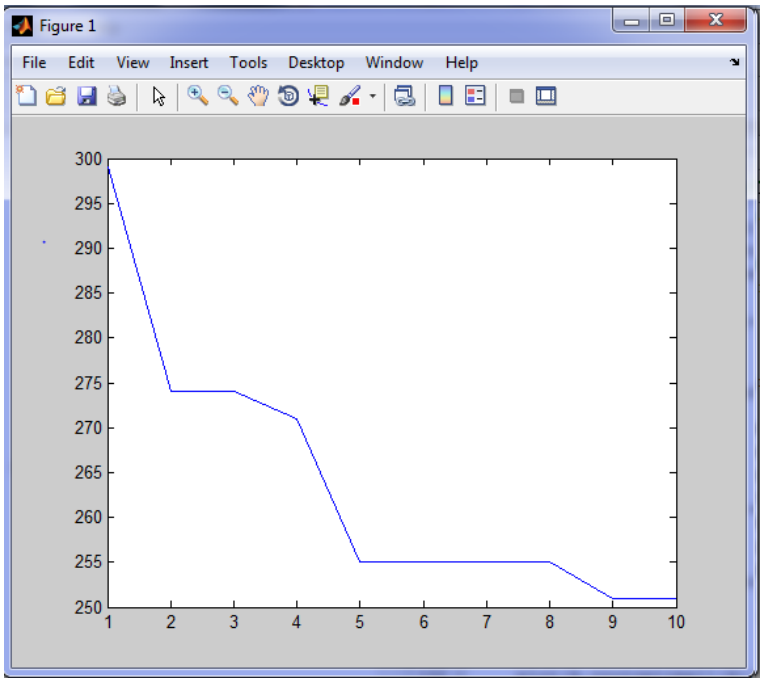
Exemple 6 :

nombre de villes $n = 10$. Il y a 262880 possibilités.

ville	1	2	3	4	5	6	7	8	9	10
1	0	98	4	85	86	17	23	79	10	11
2	98	0	81	87	93	24	5	6	12	18
3	4	81	0	19	25	76	82	13	94	100
4	85	87	19	0	2	83	89	95	96	77
5	86	93	25	2	0	90	91	97	78	84
6	17	24	76	83	90	0	48	29	35	36
7	23	5	82	89	91	48	0	31	37	43
8	79	6	13	95	97	29	31	0	44	50
9	10	12	94	96	78	35	37	44	0	27
10	11	18	100	77	84	36	43	50	27	0

-Table de distance entre les villes de l'exemple 6.

-Algorithme de Dijkstra :
Parcours : 1 3 8 2 7 9 10 6 4 5 1 .
Distance = 299.
Algorithme de colonies de fourmis pour TSP :
Parcours :8 2 7 1 3 4 5 9 10 6 8 .
Distance =229.



-Le graphe qui représenter la solution de l'exemple 6 par l'algorithme RT.

-L'algorithme de RT pour TSP :
Parcours : 8 4 1 5 6 2 10 7 3 9 8.
Distance =251.

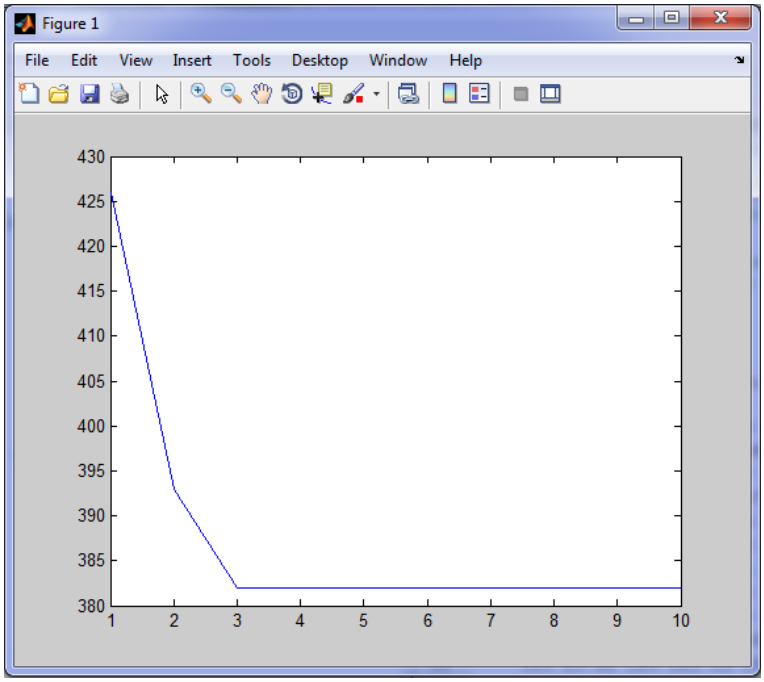
Exemple 7 :

Nombre de villes : $n = 12$. Il y a 39916800 possibilités.

ville	1	2	3	4	5	6	7	8	9	10	11	12
1	0	13	25	108	96	61	73	60	48	109	121	12
2	13	0	119	38	50	83	71	86	98	35	23	134
3	25	119	0	39	51	82	70	87	99	34	22	135
4	108	38	39	0	93	64	76	57	45	112	124	9
5	96	50	51	93	0	65	77	56	44	113	125	8
6	61	83	82	64	65	0	67	90	102	31	19	138
7	73	71	70	76	77	67	0	91	103	30	18	139
8	60	86	87	57	56	90	91	0	41	116	128	5
9	48	98	99	45	44	102	103	41	0	117	129	4
10	109	35	34	112	113	31	30	116	117	0	15	142
11	121	23	22	124	125	19	18	128	129	15	0	143
12	12	134	135	9	8	138	139	5	4	142	143	0

-Table de distance entre les villes de l'exemple 7.

– Algorithme de Dijkstra :
  Parcours : 1 12 9 8 5 2 11 10 7 6 4 3 1.
  Distance = 426.
  Algorithme de colonies de fourmis pour TSP :
  Parcours :10 11 3 1 2 4 12 9 8 5 6 7 10 .
  Distance =385.



-Le graphe qui représenter la solution de l'exemple 7 par l'algorithme RT.

– L'algorithme RT pour TSP :
Parcours :1 12 8 9 5 2 11 7 10 6 4 3 1.
Distance =382.

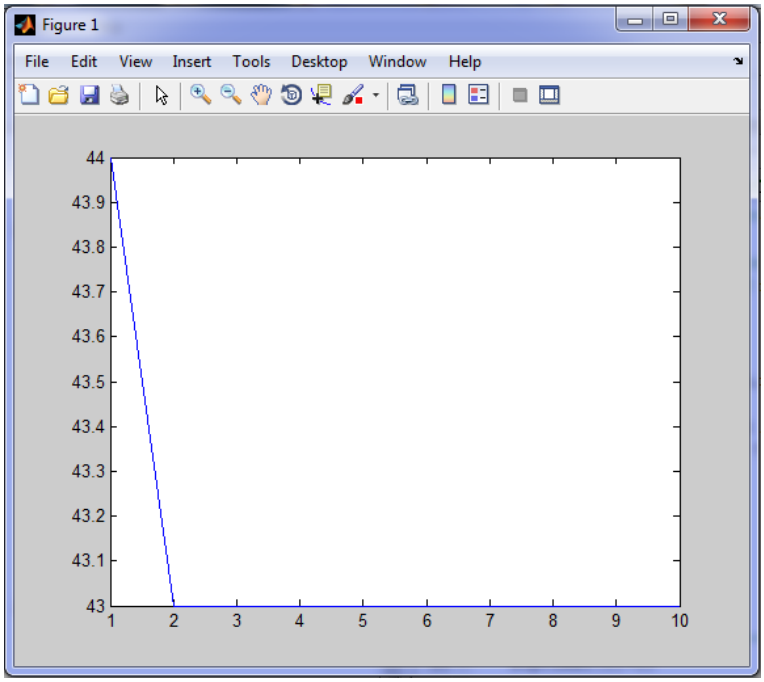
Exemple 8 :

Nombre de villes : $n = 14$. Il y a 6227020800 possibilités.

ville	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	8	9	6	5	2	5	2	5	13	2	2	3	5
2	8	0	2	5	7	6	5	10	3	5	8	9	11	13
3	9	2	0	7	9	8	7	11	5	6	10	10	12	14
4	6	5	7	0	2	4	1	9	2	7	5	8	9	11
5	5	7	9	2	0	3	2	8	4	9	3	7	8	9
6	2	6	8	4	3	0	3	5	13	11	1	4	5	7
7	5	5	7	1	2	3	0	7	1	8	4	7	8	10
8	2	11	11	9	8	5	7	0	7	15	4	1	1	3
9	5	3	5	2	4	3	1	7	0	7	4	6	8	10
10	13	5	6	7	9	11	8	15	7	0	12	14	16	18
11	2	8	10	5	3	1	4	4	4	12	0	4	5	6
12	2	9	10	8	7	4	7	1	6	14	4	0	2	4
13	3	11	12	9	8	5	8	1	8	16	5	2	0	2
14	5	13	14	11	9	7	10	3	10	18	6	4	2	0

-Table de distance entre les villes de l'exemple 8.

– Algorithme de Dijkstra :
  Parcours : 1 6 11 5 4 7 9 2 3 10 12 8 13 14 1.
  Distance = 44.
  Algorithme de colonies de fourmis pour TSP :
  Parcours :12 8 13 14 1 11 6 5 4 7 9 2 3 10 8 .
  Distance =44.



-Le graphe qui représenter la solution de l'exemple 8 par l'algorithme RT.

– RT pour TSP :
Parcours : 1 6 11 5 4 7 9 2 10 3 12 8 13 14 1.
Distance =43.

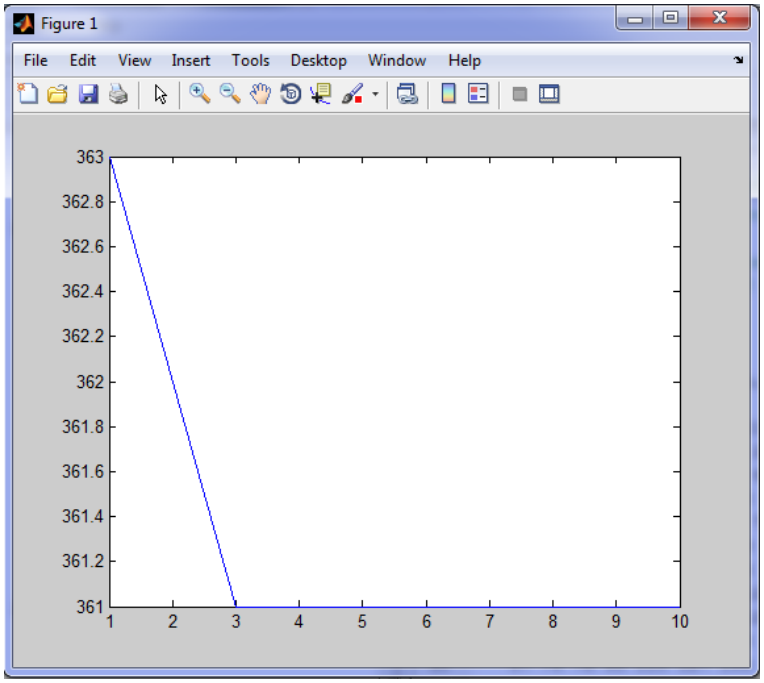
Exemple 9 :

Nombre de villes $n = 15$. Il y a 78178291200 possibilités.

ville	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	10	300	170	1	202	218	9	25	41	57	73	89	105	106
2	10	0	20	187	203	219	10	26	42	58	74	90	91	107	123
3	300	171	0	204	22	11	27	43	59	75	76	92	108	124	140
4	170	20	204	0	12	5	44	60	61	77	93	109	125	141	157
5	1	203	22	12	0	45	46	62	78	301	110	126	142	158	174
6	202	219	11	5	45	0	63	79	95	111	127	143	159	175	191
7	218	10	27	44	46	63	0	96	112	128	144	160	176	192	208
8	9	26	43	60	62	79	96	0	129	145	161	177	193	209	225
9	25	42	59	61	78	95	112	129	0	162	178	194	210	211	2
10	41	58	75	77	301	111	128	145	162	0	195	196	212	3	19
11	57	74	76	93	110	127	144	161	178	195	0	213	4	20	36
12	73	90	92	109	126	143	160	177	194	196	213	0	21	200	53
13	89	91	108	125	142	159	176	193	210	212	4	21	0	54	70
14	105	107	124	141	158	175	192	209	211	3	20	200	54	0	87
15	106	123	140	157	174	191	208	225	2	19	36	53	70	87	0

-Table de distance entre les villes de l'exemple 9.

-Algorithme de Dijkstra :
Parcours : 1 5 4 6 3 7 2 8 9 15 10 14 11 13 12 1.
Distance = 363.
Algorithme de colonies de fourmis pour TSP :
Parcours :4 6 3 7 2 8 1 5 12 13 11 14 10 15 9 4 .
Distance =345.



-Le graphe qui représenter la solution de l'exemple 9 par l'algorithme RT.

-L'algorithme de RT pour TSP :
Parcours :1 5 4 6 7 3 8 2 9 15 10 14 11 13 12 1.
Distance =361.

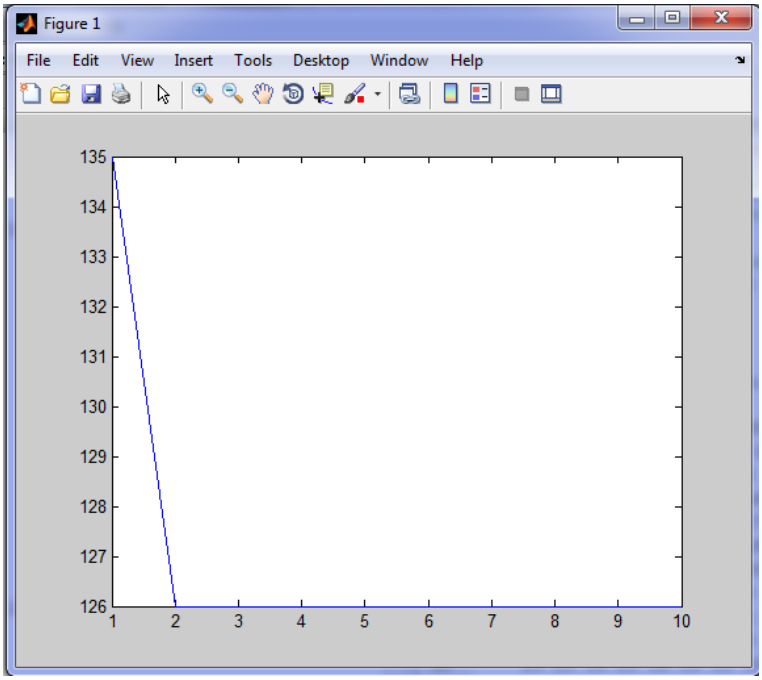
Exemple 10 :

Nombre de villes : $n = 16$. Il y a 1307674386000 possibilités.

ville	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	50	12	6	15	16	20	63	14	16	23	40	29	25	80	27
2	50	0	10	8	17	18	18	15	16	18	25	25	27	27	28	29
3	12	10	0	14	15	14	10	25	26	26	19	23	17	75	28	27
4	6	8	14	0	17	18	43	15	16	18	25	26	23	27	28	29
5	15	17	15	17	0	1	5	16	17	19	16	20	14	26	25	24
6	16	18	14	18	1	0	4	17	18	20	15	19	13	25	24	23
7	20	18	10	49	5	4	0	21	20	18	11	15	9	21	20	19
8	63	15	25	15	16	17	21	0	1	3	10	12	16	14	15	16
9	14	16	26	16	17	18	20	1	0	2	9	13	15	15	16	17
10	16	18	26	18	19	20	18	3	2	0	7	15	13	17	18	19
11	23	25	19	25	16	15	11	10	9	7	0	12	6	18	17	16
12	40	25	23	26	20	19	5	12	13	12	12	0	6	18	17	16
13	29	27	17	23	14	13	9	16	15	6	6	6	0	2	11	10
14	25	27	75	27	26	25	21	14	15	18	18	18	12	0	1	2
15	26	28	28	28	25	24	20	25	16	17	17	17	11	1	0	1
16	27	29	27	29	24	23	19	16	17	16	16	16	10	2	1	0

-Table de distance entre les villes de l'exemple 10.

-Algorithme de Dijkstra :
Parcours : 1 4 2 3 7 6 5 13 14 15 16 8 9 10 11 12 1.
Distance = 135.
Algorithme de colonies de fourmis pour TSP :
Parcours :3 2 4 1 9 8 10 11 13 14 15 16 12 7 6 5 3 .
Distance =100.



-Le graphe qui représenter la solution de l'exemple 10 par l'algorithme RT.

-L'algorithme de RT pour TSP :
Parcours : 1 4 2 3 7 6 5 14 15 16 8 9 10 12 11 1.
Distance =126.

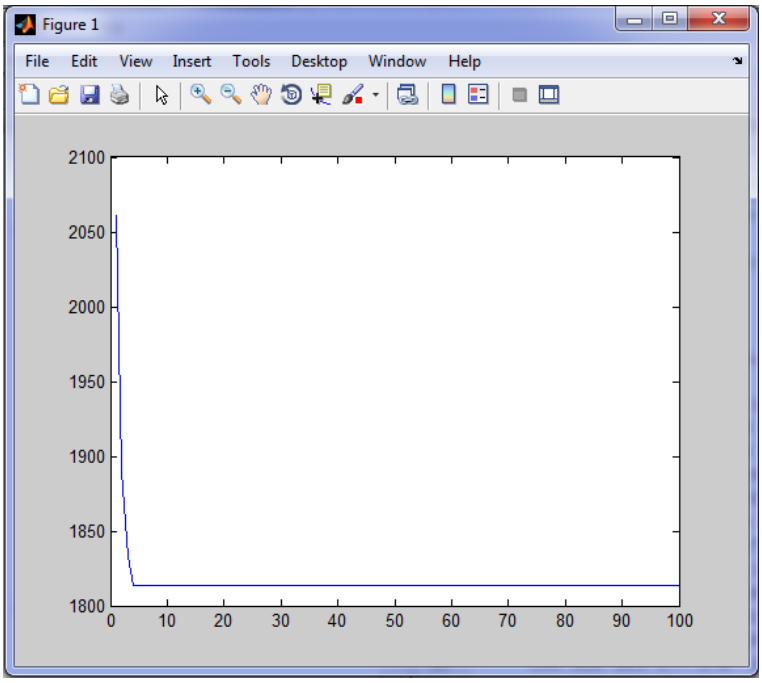
Exemple 11 :

Nombre de villes : $n = 20$. Il y a 121645100408832000 possibilités.

ville	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	21	41	340	320	101	121	260	240	181	201	180	160	261	281	100	80	341	361	20
2	21	0	359	62	82	299	279	142	162	219	199	222	242	139	119	302	322	59	39	382
3	41	359	0	63	83	298	278	143	163	218	198	223	243	138	118	303	323	58	38	383
4	340	62	63	0	317	104	124	257	237	184	204	177	157	264	284	97	77	344	364	17
5	320	82	83	317	0	105	125	256	236	185	205	176	156	265	285	96	76	345	365	16
6	101	299	298	104	105	0	275	146	166	215	195	226	246	135	115	306	326	55	35	386
7	121	279	278	124	125	275	0	147	167	214	194	227	247	134	114	307	327	54	34	387
8	260	142	143	257	256	146	147	0	233	188	208	173	153	268	288	93	73	348	368	13
9	240	162	163	237	236	166	167	233	0	189	209	172	152	269	289	92	72	349	369	12
10	181	219	218	184	185	215	214	188	189	0	191	230	250	131	111	310	330	51	31	390
11	201	199	198	204	205	195	194	208	209	191	0	231	251	130	110	311	331	50	30	391
12	180	222	223	177	176	226	227	173	172	230	231	0	149	272	292	89	69	352	372	9
13	160	242	243	157	156	246	247	153	152	250	251	149	0	273	293	88	68	353	373	8
14	261	139	138	264	265	135	134	268	269	131	130	272	273	0	107	314	334	47	27	394
15	281	119	118	284	285	115	114	288	289	111	110	292	293	107	0	315	335	46	26	395
16	100	302	303	97	96	306	307	93	92	310	311	89	88	314	315	0	65	356	376	5
17	80	322	323	77	76	326	327	73	72	330	331	69	68	334	335	65	0	357	377	4
18	341	59	58	344	345	55	54	348	349	51	50	352	353	47	46	356	357	0	23	398
19	361	39	38	364	365	35	34	368	369	31	30	372	373	27	26	376	377	23	0	399
20	20	382	383	17	16	386	387	13	12	390	391	9	8	394	395	5	4	398	399	0

-Table de distance entre les villes de l'exemple 11.

-Algorithme de Dijkstra :
Parcours : 1 20 17 16 13 12 9 2 19 18 15 7 4 3 5 6 14 11 10 8 1.
Distance = 2061.
Algorithme de colonies de fourmis pour TSP :
Parcours :13 20 17 16 8 11 19 18 15 7 4 2 1 3 5 6 14 10 9 12 13 .
Distance =1703.



-Le graphe qui représenter la solution de l'exemple 11 par l'algorithme RT.

-L'algorithme RT pour TSP :
Parcours : 20 1 17 13 12 16 9 2 19 18 15 7 4 3 5 6 14 11 10 8 20.
Distance =1814.

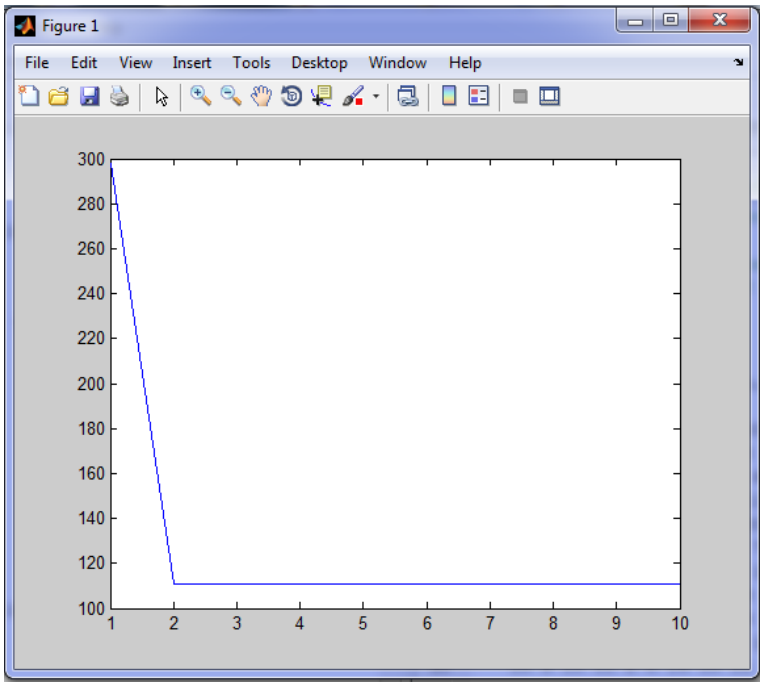
Exemple 12 :

Nombre de villes $n = 25$. il y a 620448401733239360000 possibilités.

ville	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	0	18	3	12	30	7	35	200	9	43	10	50	16	65	10	9	5	3	13	90	11	12	5	13	6
2	18	0	10	5	17	12	7	12	13	19	9	18	5	11	7	11	11	15	9	11	13	7	8	9	14
3	3	10	0	9	10	9	14	16	12	6	10	11	10	7	7	18	13	13	12	14	14	10	11	13	17
4	12	5	9	0	12	10	13	9	5	11	11	12	10	12	5	11	10	17	8	14	9	13	16	9	9
5	30	17	10	12	0	15	7	13	30	4	2	3	8	11	19	7	3	17	7	10	6	10	16	3	3
6	7	12	9	10	15	0	7	14	10	7	8	6	12	6	3	100	6	11	16	9	9	13	11	18	14
7	35	7	14	13	7	7	0	8	9	9	14	14	7	15	11	4	12	12	9	14	5	13	14	16	6
8	200	12	16	9	13	14	8	0	9	6	10	10	6	12	13	7	12	6	10	13	15	16	9	6	9
9	9	13	12	5	30	10	9	9	0	9	11	10	70	14	5	10	6	10	7	8	15	10	16	9	8
10	43	19	6	11	4	7	9	6	9	0	11	8	14	12	8	8	6	6	11	5	12	8	11	6	17
11	10	9	10	11	2	8	14	10	11	11	0	11	1	5	6	8	5	4	11	14	12	19	11	10	11
12	50	18	11	12	3	6	14	10	10	8	11	0	15	6	11	9	3	10	10	8	11	3	10	7	17
13	16	5	10	10	8	12	7	6	70	14	1	15	0	9	12	9	9	8	17	12	4	12	9	18	11
14	165	11	7	12	11	6	15	12	14	12	5	6	9	0	3	13	13	11	5	6	18	7	10	8	13
15	10	7	7	5	19	3	11	13	5	8	6	11	12	3	0	6	6	7	12	6	7	1	12	12	9
16	9	11	18	11	7	100	4	7	10	8	8	9	9	13	6	0	14	8	7	8	14	3	4	13	7
17	5	11	13	10	3	6	12	12	6	6	5	3	9	13	6	14	0	12	8	10	17	12	13	3	9
18	3	15	13	17	17	11	12	6	10	6	4	10	8	11	7	8	12	0	2	12	1	1	7	6	6
19	13	9	12	8	7	16	9	10	7	11	11	10	17	5	12	7	8	2	0	19	5	10	12	2	4
20	90	11	14	14	10	9	14	13	8	5	14	8	12	6	6	8	10	12	19	0	9	8	11	13	4
21	11	13	14	9	6	9	5	15	15	12	12	11	4	18	7	14	17	1	5	9	0	5	8	10	10
22	12	7	10	13	10	13	13	16	10	8	19	3	12	7	1	3	12	1	10	8	5	0	3	16	10
23	5	8	11	16	16	11	14	9	16	11	11	10	9	10	12	4	13	7	12	11	8	3	0	11	12
24	13	9	13	9	3	18	16	6	9	6	10	7	18	8	12	13	3	6	2	13	10	16	11	0	8
25	6	14	17	9	3	14	6	9	8	17	11	17	11	13	9	7	9	6	4	4	10	10	12	8	0

-Table de distance entre les villes de l'exemple 12.

-Algorithme de Dijkstra :
Parcours :1 3 10 5 11 13 21 18 22 15 6 12 17 24 19 25 20 14 23 16 7 2 4 9 8 1 .
Distance = 298.
Algorithme de colonies de fourmis pour TSP :
Parcours :3 1 18 21 13 11 5 12 17 24 19 25 20 10 8 9 4 2 7 16 23 22 15 6 14 3 .
Distance =98.



-Le graphe qui représenter la solution de l'exemple 12 par l'algorithme RT.

-L'algorithme RT pour TSP :
Parcours : 1 3 10 5 11 13 21 18 22 15 6 12 17 24 19 25 20 14 23 16 7 2 4 8 9 1.
Distance =111.

3.4 La Comparaison entre les algorithmes de Dijkstra ,Colonies de Fourmis et La RT

Table de comparison :

Nombre de villes	Algorithme de Dijkstra	L'algorithme RT-TSP	L'algorithme AS-TSP
5	103	57	57
6	24	18	18
7	125	47	51
8	132	125	125
9	275	197	206
10	299	251	229
12	426	382	385
14	44	43	44
15	363	361	345
16	135	126	100
20	2061	1814	1703
25	298	111	98

3.4.1 L'histogramme

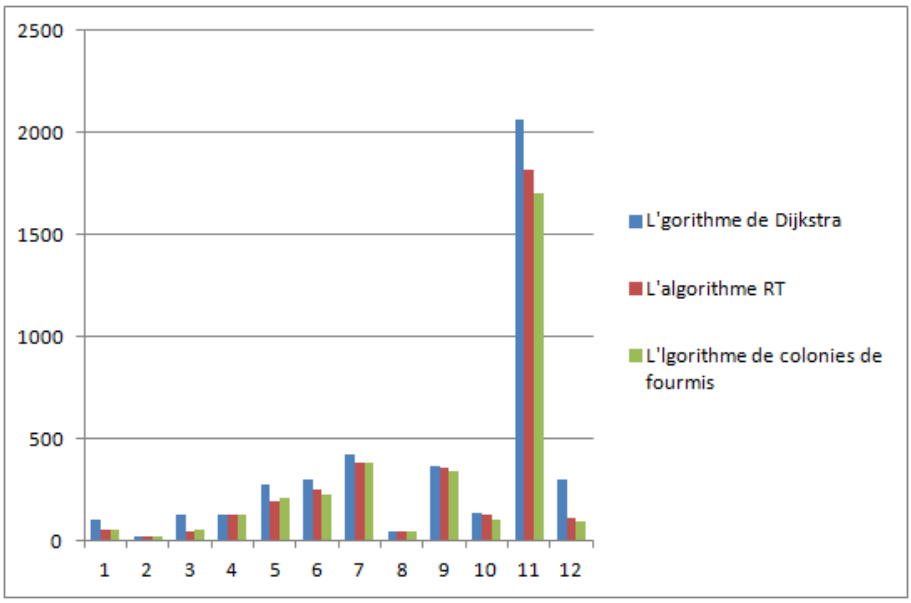


Fig 3.1 :Comparaison entre l'algorithme de Dijkstra,AS-TSP,et RT-TSP

3.4.2 Analyse de l'histogramme :

À travers les colonnes histogramme qui représentent la variation de la distance en termes de nombre de villes dans chacun de l'algorithme de Dijikstra, L'algorithme de Recherche Tabou et Colonie de Fourmis noter la différence dans les colonnes,pour un petit nombre de villes les colonnes représentants des résultats des trois méthodes sont presque égaux .
Lorsque, pour un grand nombre de villes, nous constatons une augmentation de la différence entre les trio colonnes , où l'algorithme de Recherche Tabou et l'algorithme Colonie de Fourmis atteints la distance la plus courte Par rapport à l'algorithme de Dijkstra .
Grâce à nos histogrammes de l'analyse concluent que l'algorithme de Recherche Tabou et le Colonie de Fourmis est meilleurs et plus efficace que l'algorithme de Dijkstra.
Mais engénérale pour un grande nombre des villes, l'algorithme de Colonie de Fourmis mieux que la Recherche Tabou.

Conclusion Générale

Tout un ensemble de méthodes, des plus particulières (la programmation linéaire et les méthodes générales) aux plus générales (les métaheuristiques) :

- Les méthodes particulières sont plus difficiles à programmer, mais plus efficaces.
- Les méthodes générales sont faciles à programmer et d'un champ d'application plus vaste.

Donc choisir toujours la méthode la plus spécialisée compatible avec la définition du problème.

La recherche Tabou peut être considérer comme une généralisation des méthodes d'améliorations locales traditionnelles.
L'application de recherche Tabou sur n'importe quel type de problèmes ne garantie en aucun cas un succès définitif, mais le plus important est de savoir comment adapter la recherche Tabou au problème posé, et ceci en ajustant de façon adéquate ses différents composants (restriction Tabou, critère d'aspiration,...).

Bibliographie

[1] Adam B. Levy. The Basics of Practical Optimization. SIAM, Philadelphia, USA, 2009.

[2] Ait hammoudi safa ,Bouguerra sihem :Conception et Réalisation d'une Bibliothèque de Modèle Mathématique pour le Fitting des Données Expérimentales. Thèse D'ingénieur d'Etat en Recherche Opérationnelle ,2006.

[3] A tabu search approach to the channel minimization problem. G.Dahl, K.Jornsten, and A.Lokketangen. In Liu, G., Phua, K.-H.,Ma, J., Xu, J., Gu, F., and He, C., editors, Optimization - Techniques and Applications, ICOTA'95, volume 1, pages 369–377, Chengdu, China. World Scientific.

[4] Bouchikhi Nouha :La recherche tabou . Thèse de Master, Université des Sciences Et de la Technologie d'ORAN Mohamed Boudiaf USTO 2011.

[5] Belkhir Amine,Sellal Cherif :Affectations pour optimiser le service du réseau ETUSA. Thèse de L'Ingénieur d'Etat en Recherche Opérationnelle,Université des Sciences et de la Technologie Houari Boumédiène,2009.

[6] Catherine Mancel :MODÉLISATION ET RÉSOLUTION DE PROBLÈMES D'OPTIMISATION COMBINATOIRE ISSUS D'APPLICATIONS SPATIALES. Thèse de doctorat, institut National des Sciences Appliquées de Toulouse,2004.

[7] Charles-Edmond Bichot. Élaboration d'une nouvelle métaheuristique pour le partitionnement de graphe : la méthode de fusion-fission. Application au découpage de l'espace aérien. Thèse de doctorat, institut national polytechnique de Toulouse, 2007. page 59 – 83.

[8] C. Oliva, P. Michelon & C. Artigues. Constraint and Linear Programming : Using Reduced Costs for solving the Zero/One Multiple Knapsack Problem. In International Conference on Constraint Programming, Workshop on Cooperative Solvers in Constraint Programming (CoSolv 01), pages 87–98, Paphos, Cyprus, 2001.

[9] D.Halhal, G.Walters, D.Ouazar, and D.Savic : Water network rehabilitation with a structured messy genetic algorithm. . Journal ofWater Resources Planning and Management,123(3).

[10] D.Todd, P.Sen : In Back, T., editor, Seventh Int : A multiple criteria genetic algorithm for container loading. Conf. on Genetic Algorithms ICGA'97, pages 674–681, San Mateo, California. Morgan Kaufmann.

[11] D.V.Veldhuizen, B.Sandlin, R.Marmelstein, G.Lamont, and A.Terzuoli. In Chawdhry, P., Roy, R., and Pant, P., editors, Soft Computing in Engineering Design and Manufacturing, London. Springer Verlag. Finding improved wire-antenna geometries with genetic algorithms, pages 231–240.

[12] Fabian Bastin :Modèles de Recherche Opérationnelle, pages 65-66 .Université de Montréal,2006.

[13] Frederick S. Hillier et Gerald J. Lieberman. Introduction to Operations Research. McGraw-Hill, New York, USA, seventh édition, 2001.

[14] F. Laburthe. Contraintes et algorithmes en optimisation combinatoire.Thèse de doctorat, Université Paris VII- Denis Diderot,Paris, 1998.

[15] G. Laporte. The Travelling Salesman Problem. European Journal of Operational Research, vol. 59, no. 2, pages 231–247, 1992.

[16] G. Gutin & A.P. Punnen. The traveling salesman problem and its variations. Kluwer Academic Publishers, Dordrecht, 2002.

[17] G.L. Nemhauser & L.A. Wolsey. Integer and combinatorial optimization.Wiley, New-York, 1988.

[18] Hervé Meunier : Algorithmes évolutionnaires parallèles pour l'optimisation multiobjectif de réseaux de télécommunications mobiles. Thèse de doctorat, 2002.

[19] Hervé Meunier : Algorithmes évolutionnaires parallèles pour l'optimisation multiobjectif de réseaux de télécommunications mobiles. Thèse de doctorat, 2002.

[20] http ://www.metaheuristics.org

[21] Jeffrey C. Lagarias, James A. Reeds, Margaret H. Wright et Paul E. Wright. Convergence properties of the Nelder-Mead simplex method in low dimensions. SIAM Journal on Optimization, 9(1) :112–147, 1998.

[22] Jin-Kao Hao*, Philippe Galinier**, Michel Habib*** : Revue d'Intelligence Artificielle Vol : No. 1999 Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes .

[23] Jorge Nocedal et Stephen J. Wright. Numerical Optimization. Springer, New York, NY, USA, 1999.

[24] John A. Nelder et Roger Mead. A simplex method for function minimization. Computer Journal, 7 :308–313, 1965.

[25] Johann Dréo, Alain Petrowski, Éric Taillard, Patrick Siarry, Métaheuristiques pour l'optimisation difficile, Éd.Eyrolles, Paris, septembre 2003.

[26] Joseph Ayas & Marc André Viau : La Recherche Tabou,16 novembre, 2004.

[27] Ken I. M. Mckinnon. Convergence of the Nelder-Mead simplex method to a nonstationary point. SIAM Journal on Optimization, 9(1) :148–158, 1998.

[28] Kévin Lapetoule. Les algorithmes métaheuristiques , 2006.

[29] K.Fujita, N.Hirokawa, S.Akagi, S.Kimatura, and H.Yokohata : Multi-objective optimal design of automotive engine using genetic algorithm., (1998). In Design Engineering Technical Conferences DETC'98, pages 1–11, Atlanta, Georgia.

[30] K.Fujimura : Path planning with multiple objectives. IEEE Robotics and Automation Society Magazine.

[31] K.Shaw, P.Fleming : Initial study of multi-objective genetic algorithms for scheduling the production of chilled ready meals.. In Mendel'96 2nd Int. Conf. on Genetic Algorithms, Brno, Czech Republic.

[32] Krid Radia Chenouki,Imane :Une approche d'optimisation coopérative à base d'agents optimiseurs appliqué au problème de transport à la demande (TAD) .Thèse de Master en Informatique, Université de Jijel,2012.

[33] Mahdi Samir :Optimisation Multiobjectif Par Un Nouveau Schéma De Coopération Méta/Exacte, Mémoire de Magister.

[34] M. Gondran & M. Minoux. Graphes et algorithmes. Eyrolles, Paris,1995.

[35] M. Sevaux1 and K. Ssorensen2, «MA/PM : une nouvelle. métaheuristique hybride », 1université de valenciennes-CNRS, UMR 8530, LAMIH/SP, 2université d'Anvers-Faculty of Applied Economics.

[36] Paul Feautrier, « Recherche opérationnelle », 16 novembre 2005.

[37] P. Esquirol & P. Lopez. L'ordonnancement. Economica, Paris, 1999.

[38] P. Lopez. Approche par contraintes des problèmes d'ordonnancement et d'affectation : structures temporelles et mécanismes de propagation. Habilitation à Diriger des Recherches, Institut National Polytechnique de Toulouse, dec 2003.

[39] Revue d'Intelligence Artificielle Vol : No. 1999 Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes Jin-Kao Hao*, Philippe Galinier**, Michel Habib***.

[40] R. Ostermann, D. de Werra, « Somme experiments with a timetabling system », OR Sepektum 3, 1982.

[41] ROBERT faure,CHRISTOPHE Picouleau,BERNAR Lemaire : précis de recherche opérationnelle .

[42] S. Martello, D. Pisinger & P. Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem. Management Science, vol. 45, pages 414–424, 1999.

[43] S. Martello, D. Pisinger & P. Toth. New trends in exact algorithms for the 0-1 knapsack problem. European Journal of Operational Research, vol. 123, no. 2, pages 325–332, 2000.

[44] S.Obayashi, S.Takahashi, and Y.Takeguchi : Niching and elitist models for multiobjective genetic algorithms, In Parallel ProblemSolving from Nature PPSN'5. (1998). pages 260–269,Amsterdam. Springer-Verkag.

[45] Techniques d'optimisation, les métaheurstiques, M.salomon université de Franche-comté 2004/2005.

[46] T.Friesz, G.Anandalingam, N.Mehta, K.Nam, S.Shah, and R.Tobin : The multiobjective equilibrium network design problem revisited : A simulated annealing approach.EuropeanJournal of Operational Research.

[47] The neighborhood contraint method : A genetic algorithmbased multiobjective optimization technique. D.Loughlin, S.Ranjithan. In Back, T., editor, Seventh Int. Conf. on Genetic

[48] Tripathy A., " A Lagrangian Relaxation Approach to Course Scheduling", Journal of the Operational Research Society 31, 1980.

[49] Troudi Fatiha :Résolution du problème de l'emploi du temps,Proposition d'un algorithme évolutionnaire multi objectif. Thèse de Magister,2006.

[50] V. Barichard and J. Hao, « Genetic Tabu Search for Multiobjective knapsach problem », Angres 2003.