الجمهوريـة الجزائـريـة الديمقراطيـة الشعبيـة
République Algérienne Démocratique et Populaire
وزارة التعليـم العالـي والبحـث العلمـي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Nº Réf :……

Centre Universitaire de Mila

Institut des sciences et des technologie **Département des Mathématiques et Informatiques**

Utilisation des colonies de fourmis pour la résolution d'un problème d'optimisation

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

Spécialité : ***Mathématiques Fondamentales et Appliqués***

Préparé par :
BOUAYAD Aya ***BARA Aida***

Soutenue le 4 Juin 2013 devant le jury composé de :

M. BOULAROUK Yakoub	**C.U. Mila**	**président**
M. AZI Mourad	**C.U. Mila**	**examinateur**
M. ZAIDI Ali	**C.U. Mila**	**rapporteur**

Année universitaire : 2012/2013

✲✲✲Remerciement✲✲✲

Au terme de ce travail, nous commençons par remercier DIEU pour nous avoir donné la volonté et le courage pour terminer ce modeste travail.

Nous tenons à exprimer notre gratitude et notre sincère remerciement à tous ceux qui ont contribués, de prés ou de loin à l'élaboration de ce mémoire.

Nous remerciements très chaleureux sont adressé à Monsieur ***ZAIDI Ali*** *notre encadreur.*

Nous adressons également nos vifs remerciements aux membres de jury ***BOULAROUK Yakoub*** *et* ***AZI Mourad****, qui ont bien voulu et accepter d'examiner ce modeste travail.*

Nous adressons également mes remerciements chaleureux aux membres de l'institut des sciences et de la technologie.

Nous remerciements sons également adressés à tous les enseignants qui ont contribué à notre formation.

AYA & AIDA

Table des matières

Introduction

L'optimisation est un sujet central en recherche opérationnelle, un grand nombre de problèmes d'aide à la décision pouvant en effet être décrits sous la forme de problèmes d'optimisation. Les problèmes d'identification, l'apprentissage supervisé de réseaux de neurones ou encore la recherche du plus court chemin sont, par exemple, des problèmes d'optimisation.

Les algorithmes de colonies de fourmis forment une classe des métaheuristiques récemment proposée pour les problèmes d'optimisation difficile. En recherche opérationnelle, et plus précisément dans le domaine de l'optimisation difficile, la majorité des méthodes sont inspirées par de telles études, et notamment par la biologie. Le fait que la biologie étudie souvent des systèmes présentant des comportements dits "intelligents" n'est pas étranger au fait qu'ils soient modélisés, puis transposés dans le cadre de problèmes "réels".

Parmi les domaines de la biologie fertiles en inspiration, l'éthologie (étude du comportement des animaux) a récemment donné lieu à plusieurs avancées significatives, dont la conception de systèmes de fourmis artificielles. Ces systèmes sont notamment étudiés en robotique, en classification ou encore en optimisation. On rencontre souvent le terme d'intelligence en essaim pour désigner ces systèmes, oú l'intelligence du système entier est plus grande que celle de la simple somme de ses parties.

Dans le cadre de l'optimisation, cette approche a donné lieu à la création de nouvelles métaheuristiques. Les métaheuristiques forment une famille d'algorithmes d'optimisation visant à résoudre des problèmes d'optimisation difficile, pour lesquels on ne connaît pas de méthode classique plus efficace. Elles sont généralement utilisées comme des méthodes génériques pouvant optimiser une large gamme de problèmes différents, sans nécessiter de changements profonds dans l'algorithme employé. Les algorithmes de colonies de fourmis forment ainsi une classe de métaheuristique récemment proposée pour les problèmes d'optimisation difficile discrets.[7]

Nous avons défini dans le premier chapitre, les problèmes d'optimisation combinatoire mono-objectif, et quelque exemples de ces problèmes, et par la suite différentes approches pour la résolution des problèmes d'optimisation combinatoire.

Nous avons commencé par présenter brièvement quelques méthodes exactes. Par la suite, nous avons décrit des approches heuristiques en présentant des méthodes par voisinage et d'autres par construction, ainsi les approches métaheuristiques.

Nous verrons, au cours du deuxième chapitre, la méthode des colonies de fourmis et quelques expériences sur le comportement des fourmis réelles ; et nous avons présenté des algorithmes de colonies de fourmis.
le dernier chapitre est une application d'algorithme "ant system" sur le problème de voyageur de commerce.

Chapitre 1

Optimisation combinatoire

Nous définissons dans ce chapitre les problèmes d'optimisation combinatoire et les éléments de base relatifs ainsi quelques exemples de ces problèmes. Par la suite Nous présentons les méthodes principales de résolution complètes et approchées.

1.1 Problème d'optimisation

Un problème d'optimisation au sens général est défini par un ensemble de solutions possibles S, dont la qualité peut être décrite par une fonction objectif f. On cherche alors à trouver la solution s^* possédant la meilleure qualité $f(s^*)$ (par la suite, on peut chercher à minimiser ou à maximiser $f(s)$). Un problème d'optimisation peut présenter des contraintes d'égalité (ou d'inégalité) sur S, être dynamique si $f(s)$ change avec le temps.

Il existe des méthodes déterministes (dites exactes) permettant de résoudre certains problèmes en un temps fini. Ces méthodes nécessitent généralement un certain nombre de caractéristiques de la fonction objectif, comme la stricte convexité, la continuité ou encore la dérivabilité. On peut citer comme exemple de méthode la programmation linéaire, quadratique ou dynamique, la méthode du gradient, la méthode de Newton, etc.[7].

1.2 Optimisation combinatoire

On qualifie généralement de *combinatoires* les problèmes dont la résolution se ramène à l'examen d'un nombre fini de combinaisons. Bien souvent

cette résolution se heurte à une explosion du nombre de combinaisons à explorer.

Un problème d'optimisation combinatoire peut être défini comme suit :

Etant donnée un ensemble S de combinaisons, et une fonction $f : S \rightarrow \mathbb{R}$, il s'agit de trouver la combinaison de S minimisant f, i.e, $s^* \in S, f(s^*) \leq f(s_i), \forall s_i \in S$.

Il est à noter que pour les problèmes de maximisation , il suffit de multiplier la fonction coût par (-1). Solen le contexte, f est appelée la fonction de coût, la fonction économique ou la fonction objectif,...

L'optimisation combinatoire trouve des applications dans des domaines aussi variés que la gestion, l'ingénierie, la production, les télécommunications, les transports, l'énergie, les sciences sociales et l'informatique elle-même.

1.3 Complexité théorique d'un problème

On entend ici par «complexité d'un problème» une estimation du nombre d'instructions à exécuter pour résoudre les instances de ce problème, cette estimation étant un ordre de grandeur par rapport à la taille de l'instance. Il s'agit là d'une estimation dans le pire des cas dans le sens où la complexité d'un problème est définie en considérant son instance la plus difficile. les travaux théoriques dans ce domaine ont permis d'identifier différentes classes de problèmes en fonction de la complexité de leur résolution [22]. En effet, il existe un très grand nombre de classes différents, et on se limitera ici à une présentation succincte nous permettant de caractériser formellement la notion de problème combinatoire.

Problèmes de décision

Les classes de complexité ont été introduites pour les problèmes de décision, c'est-à-dire les problèmes posant une question dont la réponse est«oui» ou «non». Pour ces problèmes, on définit notamment les deux classes P et NP :

- La classe P contient l'ensemble des problèmes polynomiaux, i.e., pouvant être résolus par un algorithme de complexité polynomiale. Cette classe caractérise l'ensemble des problèmes que l'on peut résoudre «efficacement».

– la classe NP contient l'ensemble des problèmes polynomiaux non déterministes, i.e., pouvant être résolus par un algorithme de complexité polynomiale pour une machine non déterministe (que l'on peut voir comme une machine capable d'exécuter en parallèle un nombre fini d'alternatives). Intuitivement, cela signifie que la résolution des problèmes de NP peut nécessiter l'examen d'un grand nombre (éventuellement exponentiel) de cas, mais que l'examen de chaque cas doit pouvoir être fait en un temps polynomial.

Les relations d'inclusion entre les classes P et NP sont à l'origine d'une très célèbre conjecture que l'on peut résumer par «$P \neq NP$». En effet, si la classe P est manifestement inclue dans la classe NP , la relation inverse n'a jamais été ni montrée ni infirmée.

Cependant, certains problèmes de NP apparaissent plus difficiles à résoudre dans le sens où l'on ne trouve pas d'algorithme polynomial pour les résoudre avec une machine déterministe. Les problèmes les plus difficiles de NP définissent la classe des problèmes NP-complets : un problème de NP est NP-complet s'il est au moins aussi difficile à résoudre que n'importe quel autre problème de NP, i.e., si n'importe quel autre problème de NP peut être transformé en ce problème par une procédure polynomiale.

Ainsi, les problèmes NP-complets sont des problèmes combinatoires dans le sens où leur résolution implique l'examen d'un nombre exponentiel de cas. Notons que cette classe des problèmes NP-complets contient un très grand nombre de problèmes. Pour autant, tous les problèmes combinatoires n'appartiennent pas à cette classe. En effet, pour qu'un problème soit NP-complet, il faut qu'il soit dans la classe NP, i.e., que l'examen de chaque cas puisse être réalisé efficacement, par une procédure polynomiale. Si on enlève cette contrainte d'appartenance à la classe NP, on obtient la classe plus générale des problèmes NP-difficiles, contenant l'ensemble des problèmes qui sont (au moins aussi difficiles) que n'importe quel problème de NP, sans nécessairement appartenir à NP.

A chaque problème d'optimisation on peut associer un problème de décision dont le but est de déterminer s'il existe une solution pour laquelle la fonction objectif soit supérieure (reps.inférieure) ou égale à une valeur donnée. la complexité d'un problème d'optimisation est liée à celle du problème de décision qui lui est associe. En particulier, si le problème de décision est NP-complet, alors le problème d'optimisation est dit NP-difficile. [23].

1.4 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 trois problèmes classiques d'optimisation combinatoire : le problème du sac-à-dos, le problème d'affectation, et le problème du voyageur de commerce.

1.4.1 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 sac à- 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 $j = 1$ à 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 [21].

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

donc sous la forme :

$$\begin{cases} Max \sum_{j=1}^{n} p_j \cdot x_j \\ \sum_{j=1}^{n} a_j \cdot x_j \leq b \\ x_j \in \{0,1\}, \forall j = 1..n \end{cases}$$

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 $j$eu 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 [19]. 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 [18], la programmation dynamique est combinée avec l'identification d'une variable critique et l'utilisation de techniques de renforcement des bornes.

1.4.2 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 $n \geq m$. 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 [21].

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

$$\begin{cases} Min \sum_{i=1}^{m}\sum_{j=1}^{n} c_{i,j} \cdot x_{i,j} \\ \sum_{j=1}^{n} x_{i,j} = 1, \forall i = 1..m \\ \sum_{i=1}^{m} x_{i,j} \leq 1, \forall j = 1..n \\ x_{i,j} \in \{0, 1\}, \forall i = 1..m, \forall j = 1..n \end{cases}$$

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".

1.4.3 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 TSP et certains problèmes, comme les problèmes de transport, sont plus complexes que le TSP mais 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 [21].

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 ∞ 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 :

$$\begin{cases} Min \sum_{i=1}^{n} \sum_{j=1}^{n} \delta_{i,j} \cdot x_{i,j} \\ \sum_{j=1}^{n} x_{i,j} = 1, \ \forall i = 1..n \\ \sum_{i=1}^{n} x_{i,j} = 1, \ \forall j = 1..n \\ \sum_{i \in S, j \notin S}^{n} x_{i,j} \geq 2, \ \forall S \subset X, \ S \neq \emptyset \\ x_{i,j} \in \{0,1\}, \ \forall i = 1..n, \forall j = 1..n \end{cases}$$

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

1.5 Méthodes de résolution de problèmes d'optimisation combinatoire

La majorité des problèmes d'optimisation combinatoire, sont des problèmes *NP*-difficiles et donc ne possèdent pas à ce jour un algorithme efficace, i.e ; de complexité polynomiale, valable pour toutes les données. Ceci a motivé les chercheurs à développer de nombreuses méthodes de résolution en recherche opérationnelle et en intelligence artificielle. Ces méthodes ap-

partiennent à deux grandes familles : les méthodes exactes et les méthodes approchées.

1.5.1 Les méthodes exactes

Le principe essentiel d'une méthode exacte consiste généralement à énumérer, souvent de manière implicite, l'ensemble des combinaisons de l'espace de recherche.

Branch & Bound

Le *Branch & Bound* est une technique qui effectue un parcours en profondeur de l'arbre de recherche afin de fournir une ou plusieurs solutions optimales à partir d'un ensemble de solutions potentielles. A chaque étape de la recherche, correspondant à un noeud de l'arbre de recherche, l'algorithme utilise une fonction Bound pour calculer une borne de l'ensemble des solutions du sous-arbre prenant sa racine à ce noeud. En début de résolution, cette borne est initialisée à une valeur maximale (en cas de minimisation). Si cette évaluation est moins bonne que la meilleure solution trouvée jusqu'à ce niveau de recherche, tout le sous-arbre peut être coupé.

Il importe de souligner que l'efficacité de l'algorithme Branch & Bound dépend étroitement du calcul de la borne utilisée.

Programmation dynamique

La programmation dynamique est une méthode ascendante : On commence d'habitude par les sous problèmes les plus petits et on remonte vers les sous problèmes de plus en plus difficiles. Elle est utilisée pour les problèmes qui satisfont au principe d'optimalité de Bellman : "Dans une séquence optimale (de décisions ou de choix), chaque sous-séquence doit aussi être optimale". Un exemple de ce type de problème est le plus court chemin entre deux sommets d'un graphe. L'idée de base est d'éviter de calculer deux fois la même chose, généralement en utilisant une table de résultats déjà calculés, remplie au fur et à mesure qu'on résout les sous problèmes.

Il est à noter que la programmation dynamique est utilisée pour résoudre des problèmes polynomiaux (et non *NP*-difficiles).

1.5.2 Les méthodes approchées

1. Les approches heuristiques

Une heuristique d'optimisation est une méthode approchée se voulant simple, rapide et adaptée à un problème donne. 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 trouvée.

Nous distinguons parmi les approches heuristiques les approches par voisinage et les approches par construction. Les approches par voisinage partent d'une ou plusieurs solutions initiales qu'elles cherchent à améliorer à partir d'un voisinage défini, alors que les approches constructives partent d'une solution vide et génèrent de nouvelles solutions de façon incrémentale en ajoutant itérativement des composants aux solutions en cours de construction. [7].

Approches par voisinage

L'idée de ces approches est de structurer l'ensemble des solutions en terme de voisinage, et d'explorer cet ensemble en partant d'une ou plusieurs solutions initiales et en choisissant à chaque itération une ou plusieurs solutions voisines d'une ou plusieurs solutions courantes. Le voisinage d'une solution S est l'ensemble des solutions qui peuvent être atteintes à partir de S en un seul *mouvement*, un mouvement étant par exemple, le changement de valeur d'une variable.

Un algorithme de recherche locale glouton cherche dans le voisinage une solution qui améliore la qualité de la solution courante. Si une telle solution est trouvée, elle remplace la solution courante et la recherche locale continue. Ces étapes sont répétées jusqu'à ce qu'aucune solution meilleure n'est trouvée dans le voisinage de la solution courante et l'algorithme termine avec un optimum local. L'optimum local est la meilleure solution parmi les solutions du voisinage, contrairement à l'optimum global qui est la meilleure solution parmi toutes les solutions possibles.

Plusieurs améliorations à cet algorithme glouton ont été proposées dans la littérature pour essayer d'échapper aux optima locaux. Nous présentons, dans ce qui suit, quelques unes de ces améliorations comme la recherche tabou, le recuit simulé, les algorithmes génétiques et l'optimisation par essaim particulaire.

Lors de l'utilisation de ces approches heuristiques, un point critique est

de trouver un équilibre entre l'exploitation de l'expérience de recherche (la notion d'intensification) et l'exploration de nouvelles zones de l'espace de recherche non encore visitées (la diversification). Une bonne recherche doit être capable de trouver un bon compromis entre ces deux notions duales.

Algorithme de Dijkstra

Il s'agit de construire progressivement, à partir des données initiales, un sous-graphe dans lequel sont classés les différents sommets par ordre croissant de leur distance minimale au sommet de départ. La distance correspond à la somme des poids des arêtes empruntées.

Au départ, on considère que les distances de chaque sommet au sommet de départ sont infinies. Au cours de chaque itération, on va mettre à jour les distances des sommets reliés par un arc au dernier du sous-graphe (en ajoutant le poids de l'arc à la distance séparant ce dernier sommet du sommet de départ ; si la distance obtenue ainsi est supérieure à celle qui précédait, la distance n'est cependant pas modifiée). Après cette mise à jour, on examine l'ensemble des sommets qui ne font pas partie du sous-graphe, et on choisit celui dont la distance est minimale pour l'ajouter au sous-graphe.

La première étape consiste à mettre de côté le sommet de départ et à lui attribuer une distance de 0. Les sommets qui lui sont adjacents sont mis à jour avec une valeur égale au poids de l'arc qui les relie au sommet de départ (ou à celui de poids le plus faible si plusieurs arcs les relient) et les autres sommets conservent leur distance infinie. Le plus proche des sommets adjacents est alors ajouté au sous-graphe.

La seconde étape consiste à mettre à jour les distances des sommets adjacents à ce dernier. Encore une fois, on recherche alors le sommet doté de la distance la plus faible. Comme tous les sommets n'avaient plus une valeur infinie, il est donc possible que le sommet choisi ne soit pas un des derniers mis à jour.

On l'ajoute au sous-graphe, puis on continue ainsi à partir du dernier sommet ajouté, jusqu'à épuisement des sommets ou jusqu'à sélection du sommet d'arrivée.

2. Approche constructives

Les approches constructives commencent à partir d'une solution vide qu'elles construisent par incréments au fur et à mesure de la recherche. Nous

commençons par définir les algorithmes gloutons qui présentent la méthode de base à partir de laquelle sont dérivés les algorithmes des colonies de fourmis, par la suite nous présentons les algorithmes à estimation de distribution.

Algorithmes gloutons

Le principe d'un algorithme glouton (appelé en anglais greedy algorithm) est de partir d'une solution initiale vide et d'ajouter, d'une manière incrémentale, des composants de solutions sans remettre en cause les choix intérieurs jusqu'à obtenir une solution complète.

Dans le cas le plus simple, les composants de solutions sont ajoutés d'une manière aléatoire. De meilleurs résultats sont généralement obtenus en utilisant une heuristique du bénéfice qu'apporte le composant à ajouter, c'est ce qu'on appelle le critère gradient. Un inconvénient majeur de ces méthodes que l'utilisation du critère gradient dans les premières étapes peut mener à des déplacements très faibles dans les dernières phases de construction de solutions et engendrer des solutions de qualité médiocre.

La performance des algorithmes gloutons dépend étroitement de la pertinence de l'heuristique utilisée,i.e. leur capacité à exploiter les connaissances du problème.

Algorithmes à estimation de distribution

Les algorithmes à estimation de distribution (ou en anglais estimation of distribution Algorithme, EDA) forment une famille de métaheuristique inspirée aussi des algorithme génétiques.cependant, ils n'emploient pas d'opérateurs de croisement ni de mutation, mais plutôt ils sélectionnent les meilleures solutions de la population courante et extraient des informations statistiques globales des solutions extraites. Un modèle de probabilité de distribution des solutions prometteuses est construit en se basant sur les informations extraites. Par la suite, les solutions construites remplacent en partie ou totalement les solutions de la population courante Plus précisément, l'algorithme générale de EDA peut être décrit comme suit :

Etape 0 : Prendre aléatoirement un ensemble de solutions pour construire la population initiale.

Etape 1 : sélectionner quelques solutions de la population courante relativement à une méthode de sélection. Construire le modèle de proba-

bilité des solutions sélectionnées. Utiliser le modèle probabiliste pour construire de nouvelles solutions selon un principe glouton.

Etape 2 : Remplacer quelques ou tous les membres de la population courante par les nouvelles solutions construites à partir du modèle de probabilité.

Etape 3 : Si la condition d'arrêt n'est pas satisfaite,aller à Etape 1.

3. les approches métaheuristique

Parmi les heuristiques, certaines sont adaptables à un grand nombre de problèmes différents sans changements majeurs dans l'algorithme, on parle alors de méta-heuristiques. La plupart des heuristiques et des métaheuristiques utilisent des processus aléatoires comme moyens de récolter de l'information et de faire face à des problèmes comme l'explosion combinatoire. les métaheuristiques sont généralement itératives, c'est-à-dire qu'un même schéma de recherche est appliqué plusieurs fois au cours de l'optimisation, et directes, c'est-à-dire qu'elles n'utilisent pas l'information du gradient de la fonction objectif.
les métaheuristiques souvent inspirées d'analogies avec la réalité (physique, biologie, éthologie, . . .). [7].

les algorithmes génétiques

Les algorithmes génétiques sont des méthodes évolutionnistes qui s'inspirent fortement des mécanismes biologiques liés aux principes de sélection et d'évolution naturelle. Développés initialement par Holland et al. [16] pour répondre à des besoins spécifiques en biologie, les algorithmes génétiques ont rapidement été adaptés à des contextes très variés. Dans un algorithme génétique simple, un individu (une solution) est caractérisé par une structure de données qui représente son code génétique. La force de ce dernier, appelée finesse, peut être mesurée par la valeur de la fonction objectif correspondante. La recherche est réglée par trois opérateurs qui sont appliqués successivement. La phase de coopération est gouvernée par un opérateur de sélection et un opérateur de croisement (ou crossover) alors que la phase d'adaptation individuelle fait appel à un opérateur de mutation.

La création d'une nouvelle génération est obtenue par itération de l'algorithme génétique qui va créer de nouveaux individus et en détruire d'autres (mécanisme de sélection naturelle) ce qui permet le renouvellement de la po-

pulation (l'ensemble des solutions courantes).

La sélection : La phase de sélection spécifie les individus de la population P qui sont amenés à se reproduire par croisement La méthode de base, appelée roue de loterie attribue à chaque individu une probabilité de reproduction proportionnelle à son adaptation dans la population.

Le croisement : Étant donné deux individus sélectionnés, l'opérateur de croisement crée deux nouveaux individus. Cet opérateur coupe les deux chaines juxtaposées en un ou plusieurs points et produit deux nouvelles chaines après avoir échangé les parties coupées.

La mutation : Une mutation est un changement aléatoire d'un ou plusieurs bits de la chaine codant l'individu. L'opérateur de croisement devient moins efficace avec le temps, car les individus deviennent similaires. C'est à ce moment que le phénomène de mutation prend toute son importance : ces mutations ne créent généralement pas de meilleurs solutions au problème, mais elles évitent l'établissement de populations uniformes incapables d'évoluer.

Il est important de souligner que les concepts qui sont à la base des algorithmes génétiques sont extrêmement simples. En effet, ils font uniquement intervenir des nombres générés aléatoirement et un ensemble de règles probabilistes très générales qui ne tiennent pas forcément compte de toutes les particularités du problème traité.

Recherche Tabou

La Recherche Tabou est une méthode heuristique d'optimisation combinatoire développée par Glover [14]. La méthode Tabou effectue des mouvements d'une solution S à une meilleure solution S appartenant au voisinage de S noté $N(S)$. Si ce voisinage est trop grand, seulement une partie $V(S) \subset N(S)$ sera examinée. Pour éviter de rester bloqué dans le premier minimum local rencontré, la méthode Tabou accepte des solutions voisines dont la qualité est moins bonne que la solution courante. Elle se dirige alors vers la solution voisine qui dégrade le moins possible la fonction objectif. Cette dégradation de f, que l'on espère être temporaire, devrait permettre à l'algorithme d'atteindre des régions de l'espace contenant des solutions meilleures que celles rencontrées jusque-là. Cependant, l'algorithme Tabou risque de boucler en revisitant des solutions antérieures pendant l'itération suivante (ou après quelques itérations intermédiaires).

Pour éviter ce phénomène, l'algorithme utilise une structure de mémoire

dans laquelle il stocke les mouvements interdits à chaque étape afin de ne pas obtenir une solution déjà rencontrée récemment. Ces mouvements interdits à une itération donnée sont dits tabous, d'où le nom attribué à la méthode par Glover [14]. Le caractère tabou d'un mouvement doit être temporaire afin de donner une plus grande flexibilité à l'algorithme en lui permettant de remettre en question les choix effectués, une fois que les risques de bouclage ont été écartés. La façon la plus simple de mettre en oeuvre ce système d'interdictions consiste à garder en mémoire les derniers mouvements effectués et à empêcher l'algorithme de faire les mouvements inverses à ces derniers mouvements. Ces mouvements sont mémorisés dans une liste T, appelée liste taboue. La gestion de cette liste se fait de manière circulaire en remplaçant à chaque itération le mouvement le plus ancien de la liste par le mouvement courant.

Cependant, il peut se trouver des cas où il est intéressant de révoquer le statut tabou d'un mouvement lorsque son application à la solution courante permet d'atteindre une solution meilleure. La mise en œuvre d'un tel mécanisme est réalisée à l'aide d'une fonction auxiliaire A qui est appelée fonction d'aspiration.

Dans cette méthode, le compromis entre les mécanismes «d'intensification-diversification» peut être géré par la taille de la liste taboue. Pour favoriser l'intensification de la recherche il suffit de diminuer la taille de la liste tabou, et contrairement, pour favoriser la diversification il s'agit d'augmenter la taille de cette liste.

Recuit simulé

La méthode du "Recuit Simulé" repose sur une analogie avec la thermodynamique. En effet, la recherche par un système physique des états d'énergie les plus bas est l'analogue formel d'un processus d'optimisation combinatoire.

Le recuit est un processus physique de chauffage. Un système physique porté à une température assez élevée devient liquide dans ce cas le degré de liberté des atomes qui le composent augmente. Inversement lorsque l'on baisse la température le degré de liberté diminue jusqu'à obtenir un solide. Suivant la façon dont on diminue la température on obtient des configurations d'énergie différentes :

– Baisse brutale de la température, la configuration atteinte est le plus souvent un état métastable, dont l'énergie est supérieure à celle du minimum absolu. Le système est en quelque sorte piégé dans ce minimum

local.
- Baisse progressive de la température de façon à éviter de piéger le système dans des vallées d'énergie élevée, pour l'envoyer vers les bassins les plus importants et les plus profonds, là où les baisses ultérieures de température le précipiteront vers les fonds.

La méthode d'optimisation du recuit simulé a été proposée en 1982 par S. Kirkpatrick et al. [17] à partir de la méthode de Metropolis et al. [20] qui était utilisée pour modéliser les processus physiques. Kirkpatrick s'est inspiré de la mécanique statistique en utilisant en particulier la distribution de Bolzmann. Le recuit simulé permet de sortir d'un minimum local en acceptant avec une certaine probabilité une dégradation de la fonction. Ainsi, la probabilité p qu'un système physique passe d'un niveau d'énergie E_1 à un niveau E_2 est donnée par :

$$p = e^{-\frac{E_2 - E_1}{k_b \cdot T}} \tag{1.1}$$

k_b est la constante de Bolzmann.
T est la température du système.

Comme le montre cette formule la probabilité d'observer une augmentation de l'énergie est d'autant plus grande que la température est élevée, donc au niveau du recuit simulé :

- Une diminution de la fonction sera toujours acceptée.
- Une augmentation de la fonction sera acceptée avec une probabilité définie selon une formule du type (1.1).

Dans la méthode du recuit simulé, les mécanismes d'intensification et de diversification sont contrôlés par la température. En effet,la température décroît au fil du temps de sorte que la recherche tend à s'intensifer vers la fin de l'algorithme.

Algorithmes de colonies de fourmis

Les algorithmes de colonies de fourmis forment une classe des métaheuristiques récemment proposée pour des problèmes d'optimisation difficiles. Ces algorithmes s'inspirent des comportements collectifs de dépôt et de suivi de piste observés dans les colonies de fourmis. Une colonie d'agents simples (les fourmis) communiquent indirectement via des modifications dynamiques de leur environnement (les pistes de phéromones) et construisent ainsi une

solution à un problème en s'appuyant sur leur expérience collective. Et nous parlerons en détail dans le chapitre suivant.

Chapitre 2

Colonie de fourmis et TSP

2.1 La méthode de colonie de fourmis

Dans ce chapitre, nous présentons la métaheuristique ACO (Ant Colony Optimization), Nous expliquons l'analogie biologique, et quelques expériences à partir de la quelle ACO a été inspirée par l'étude sur le comportement des fourmis réelles effectuées par Deneubourg et al. [5].

Nous présentons par la suite l'algorithme "Ant System", ainsi que ses extensions.

2.1.1 Historique

La métaheuristique d'optimisation par colonies de fourmis a été initialement introduite par Dorigo, Maniezzo et al dans les années 1990.

A l'origine, l'optimisation par colonie de fourmis a été conçue pour résoudre le problème du voyageur de commerce en proposant le premier algorithme ACO : "Ant System" (AS) [10]. Par la suite, un nombre considérable d'applications de ACO a été proposé telles que l'affectation quadratique [13], le routage des véhicules [2], le problème de satisfaction de contraintes [23],...

2.1.2 Analogie biologique

les insectes sociaux en général, et les fourmis en particulier, résolvent naturellement des problèmes relativement complexes. Les biologistes ont étudié comment les fourmis arrivent à résoudre collectivement des problèmes trop complexes pour un seul individu, notamment les problèmes de choix lors de

l'exploitation de sources de nourriture.

Les fourmis ont la particularité d'employer pour communiquer des substances volatiles appelées phéromones. Elles sont attirées par ces substances, qu'elles perçoivent grâce à des récepteurs situés dans leurs antennes. Ces substances sont nombreuses et varient selon les espèces.

Les fourmis peuvent déposer des phéromones au sol, grâce à une glande située dans leur abdomen, et former ainsi des pistes odorantes, qui pourront être suivies par leurs congénères (figure 2.1). [3].

Il est difficile de connaître avec précision les propriétés physico-chimiques des pistes de phéromone, qui varient en fonction des espèces et d'un grand nombre de paramètres. Cependant, les métaheuristiques d'optimisation de colonies de fourmis s'appuient en grande partie sur le phénomone d'évaporation des pistes de phéromone. Or, on constate dans la nature que les pistes s'évaporent plus lentement que ne le prévoient les modèles. Les fourmis réelles disposent en effet d'heuristiques leur apportant un peu plus d'informations sur le problème (par exemple une information sur la direction).

Il faut garder à l'esprit que l'intérêt immédiat de la colonie (trouver le plus court chemin vers une source de nourriture) peut être en concurrence avec l'intérêt adaptatif de tels comportements. Si l'on prend en compte l'ensemble des contraintes que subit une colonie de fourmis (prédation, compétition avec d'autres colonies, etc.), un choix rapide et stable peut être meilleur, et un changement de site exploité peut entraîner des coûts trop forts pour permettre la sélection naturelle d'une telle option. [7].

Fig. 2.1 Des fourmis suivant une piste de phéromone.

2.2 Compertement des fourmis

2.2.1 Pont binaire de Deneubourg

L'expérience montre un nid d'une colonie de fourmis, qui est séparé d'une source de nourriture par un pont à deux voies de même longueur. On laisse évoluer les fourmis sur le pont, on trace ainsi en fonction du temps, le graphe du nombre de fourmis empruntant chaque branche. Le résultat de l'expérience est exposé à la figure 2.2.

L'illustration (a) représente la configuration physique de l'expérience. Le graphique (b) indique l'évolution de ce système en fonction du temps : on constate que les fourmis ont tendance à emprunter le même chemin (par exemple celui du haut) après une dizaine de minutes.

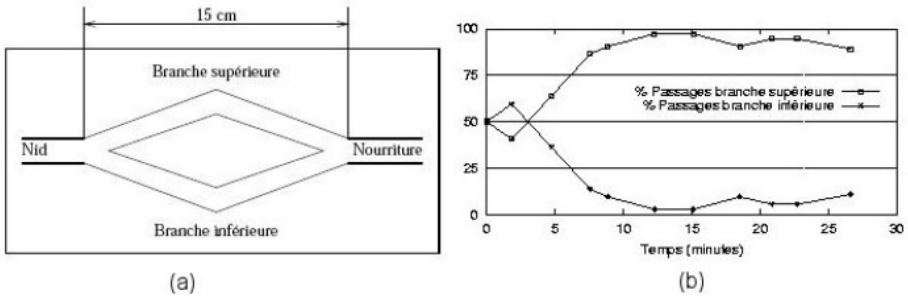

Fig. 2.2– Pont binaire de Deneubourg.

Explication

Au départ, il n'y a pas de phéromone sur le pont. Donc, chaque branche peut être choisie par une fourmi avec la même probabilité. Néanmoins, dans notre exemple, après une certaine période, des variations aléatoires ont fait qu'un peu plus de fourmis ont choisi le chemin du haut plutôt que celui du bas.

Puisque les fourmis déposent des phéromones en avançant et puisqu'elles sont plus nombreuses en haut qu'en bas, le chemin du haut comportera plus de phéromones. Cette quantité supérieure de phéromone incite plus de fourmis à choisir la branche du haut, donc la quantité de phéromone déposée augmentera encore plus. On en déduit que plus les fourmis suivent un chemin, plus ce chemin devient intéressant à suivre. Ainsi la probabilité avec laquelle une fourmi choisit un chemin, augmente avec le nombre de fourmis qui ont pris ce chemin précédemment.

2.2.2 Expérience de sélection des branches les plus courtes

Les fourmis utilisent les pistes de phéromone pour marquer leur trajet, par exemple entre le nid et une source de nourriture. Une colonie est ainsi capable de choisir (sous certaines conditions) le plus court chemin vers une source à exploiter [15, 1], sans que les individus aient une vision globale du trajet. En effet, comme l'illustre la figure 2.3, les fourmis le plus rapidement arrivées au nid, après avoir visité la source de nourriture, sont celles qui empruntent les deux branches les plus courtes. Ainsi, la quantité de phéromone présente sur le plus court trajet est légèrement plus importante que celle présente sur le chemin le plus long. Or, une piste présentant une plus grande concentration en phéromone est plus attirante pour les fourmis, elle a une probabilité plus grande d'être empruntée. La piste courte va alors être plus renforcée que la longue, et, à terme, sera choisie par la grande majorité des fourmis. On constate qu'ici le choix s'opère par un mécanisme d'amplification d'une fluctuation initiale.

Cependant, il est possible qu'en cas d'une plus grande quantité de phéromone déposée sur les grandes branches, au début de l'expérience, la colonie choisisse le plus long parcours. D'autres expériences [1], avec une autre espèce de fourmis, ont montré que si les fourmis sont capables d'effectuer des demi-tours sur la base d'un trop grand écart par rapport à la direction de la source de nourriture, alors la colonie est plus flexible et le risque d'être piégé sur le chemin long est plus faible.

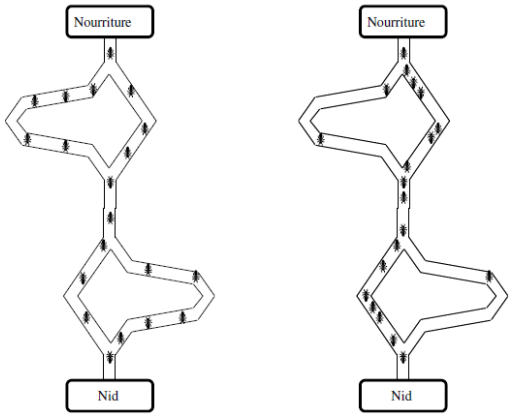

Fig. 2.3 Expérience de sélection des branches les plus courtes par une colonie de fourmis : (a) au début de l'expérience, (b) à la fin de l'expérience.

2.2.3 Effet de la coupure d'une piste de phéromone

Cette fois, les fourmis sont en train de suivre une piste de phéromones, comme présenté à la figure 2.4 (a). À un moment donné, on a un obstacle qui barre la route des fourmis. Les fourmis qui arrivent à côté de l'obstacle doivent choisir soit d'aller à gauche soit d'aller à droite (b). Puisqu'aucune phéromone n'est déposée le long de l'obstacle, il y a autant de fourmis qui partent à gauche qu'à droite. Néanmoins, puisque le chemin de droite est plus court que celui de gauche, les fourmis qui l'empruntent, vont retrouver plus vite la piste de phéromone de départ. Pour chaque fourmi allant du nid à la nourriture, on associe également une fourmi qui va de la nourriture au nid (en fait elles ont été séparées par l'apparition brutale de l'obstacle). Les phéromones de ces fourmis vont se superposer à droite. Donc quand elles vont rejoindre le chemin initial, le chemin de droite sera deux fois plus imprégnée de phéromone que la piste de gauche, où les deux fourmis n'ont pas encore pu rejoindre la piste initiale (ce chemin étant plus long). Les fourmis qui arrivent à l'obstacle à partir de ce moment, préféreront suivre la piste de droite. Le nombre de fourmis qui passent par la droite va augmenter, ce qui augmentera encore la concentration de phéromones.

De plus, l'évaporation des phéromones sera plus forte sur la piste de gauche du fait que sa longueur est supérieure. La piste de gauche sera dont rapidement abandonnée, parce qu'elle en est beaucoup moins imprégnée : les fourmis passeront toutes très rapidement par la piste la plus courte. [7].

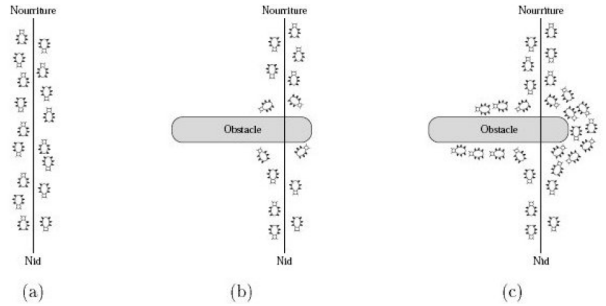

Fig 2.4 : Effet de la coupure d'une piste de phéromone

Conclusion

Il est intéressant de remarquer que bien qu'une seule fourmi soit capable de construire une solution(ie, de trouver une chemin du nid à la nourriture), c'est seulement le comportement de l'ensemble de la colonie qui crée le chemin le plus court.

2.3 Les différences entre les fourmis réelles et virtuelles

2.3.1 Points communs

Colonie d'individus coopérants

Comme pour les fourmis réelles, une colonie virtuelle est un ensemble d'entités non-synchronisés, qui se rassemblent ensemble pour trouver une "bonne" solution au problème considéré. Chaque groupe d'individus doit pouvoir trouver une solution même si elle est mauvaise.

Pistes de phéromones

Ces entités communiquent par le mécanisme des pistes de phéromone. Cette forme de communication joue un grand rôle dans le comportement des fourmis : son rôle principal est de changer la manière dont l'environnement est perçu par les fourmis, en fonction de l'historique laissé par ces phéromones.

Évaporation des phéromones

La méta-heuristique ACO comprend aussi la possibilité d'évaporation des phéromones. Ce mécanisme permet d'oublier lentement ce qui s'est passé avant. C'est ainsi qu'elle peut diriger sa recherche vers de nouvelles directions, sans être trop contrainte par ses anciennes décisions.

Recherche du plus petit chemin

Les fourmis réelles et virtuelles partagent un but commun, recherche du plus court chemin reliant un point de départ (le nid) à des sites de destination (la nourriture).

Déplacement locaux

Les vraies fourmis ne sautent pas des cases, tout comme les fourmis virtuelles. Elles se contentent de se déplacer entre sites adjacents du terrain.

Choix aléatoire lors des transitions

Lorsqu'elles sont sur un site, les fourmis réelles et virtuelles doivent décider sur quel site adjacent se déplacer. Cette prise de décision se fait au hasard et dépend de l'information locale déposée sur le site courant. Elle doit tenir compte des pistes de phéromones, mais aussi du contexte de départ (ce qui revient à prendre en considération les données du problème d'optimisation combinatoire pour une fourmi virtuelle).

2.3.2 Différences

Les fourmis virtuelles possèdent certaines caractéristiques que ne possèdent pas les fourmis réelles :

Elle vivent dans un monde non-continu

Leurs déplacements consistent en des transitions d'état.

Mémoire (état interne) de la fourmi

Les fourmis réelles ont une mémoire très limitée. Tandis que nos fourmis virtuelles mémorisent l'historique de leurs actions. Elles peuvent aussi retenir des données supplémentaires sur leurs performances.

Nature des phéromones déposées

Les fourmis réelles déposent une information physique sur la piste qu'elles parcourent, là où les fourmis virtuelles modifient des informations dans les variables d'états associées au problème. Ainsi, l'évaporation des phéromones est une simple décrémentation de la valeur des variables d'états à chaque itération.

Qualité de la solution

Les fourmis virtuelles déposent une quantité de phéromone proportionnelle à la qualité de la solution qu'elles ont découvert.

Retard dans le dépôt de phéromone

Les fourmis virtuelles peuvent mettre à jour les pistes de phéromones de façon non immédiate : souvent elles attendent d'avoir terminé la construction de leur solution. Ce choix dépend du problème considéré bien évidemment.

Capacités supplémentaires

Les fourmis virtuelles peuvent être pourvues de capacités artificielles afin d'améliorer les performances du système. Ces possibilités sont liées au problème et peuvent être :

1. L'anticipation : la fourmi étudie les états suivants pour faire son choix et non seulement l'état local.
2. Le retour en arrière : une fourmi peut revenir à un état déjà parcouru car la décision qu'elle avait prise à cet état a été mauvaise. [7]

2.4 Les algorithmes de colonie de fourmis pour le problème du voyageur de commerce

définition 1 *(Problème de voyageur de commerce TSP)*
Un voyageur de commerce doit visiter un ensemble $v_1, \cdots, v_n$ *de n villes dont on connaît les distances respectives* $d(v_i, v_j), \forall (i,j) \in \{1, \cdots, n\}^2$. *Le problème consiste à trouver la permutation* σ *telle que la séquence* $s = (v_{\sigma(1)}, \cdots, v_{\sigma(n)})$ *minimise la distance totale* $D(\sigma)$ *parcourue par le voyageur :*

$$D(\sigma) = \sum_{i=1}^{n-1} d(v_{\sigma(i)}, v_{\sigma(i+1)}) + d(v_{\sigma(n)}, v_{\sigma(1)})$$

L'espace de recherche est l'ensemble des combinaisons possibles des n villes, soit au total $n!$ *combinaisons.*

2.4.1 Ant System

Le problème de voyageur de commerce (TSP) a fait l'objet de la première implémentation d'un algorithme de colonie de fourmis : le "Ant System".[8].

Du côté des fourmis artificielles, quelques modifications sont apportées aux capacités des fourmis décrites précédemment :
– Elles possèdent une mémoire.
– Elles ne sont pas totalement aveugles.
– Le temps est discret.

Dans [4] sont introduits trois algorithmes qui mettent à profit ce comportement collectif. Ils sont appliqués au TSP. De ces trois algorithmes, on retiendra celui qui à donné naissance à l'algorithme AS. [9].

Voici la modélisation du comportement des fourmis qui est proposée. Les fourmis sont placées sur les sommets du graphe (i.e., sur chaque ville). Elles se déplacent d'un sommet à l'autre en empruntant les arêtes du graphe. On note par $b_i(t)$ le nombre de fourmis dans la ville i à l'instant t et soit $m = \sum_{i=1}^{n} b_i(t)$ le nombre total de fourmis. Chaque agent-fourmi possède les caractéristiques suivantes :

- La fourmi dépose une trace de phéromones sur l'arête (i, j) quand elle se déplace de la ville i à la ville j ;

- Elle choisit la ville de destination suivant une probabilité qui dépend de la distance entre cette ville et sa position et de la quantité de phéromones présente sur l'arête (règle de transition) ;

- Afin de ne passer qu'une seule fois par chaque ville, la fourmi ne peut se rendre sur une ville qu'elle a déjà traversée, c'est pour cela que la fourmi doit être dotée d'une mémoire.

Pour éviter qu'une fourmi ne revienne sur ses pas, elle conserve la liste des villes qu'elle a déjà traversées. Cette liste, nommée liste-tabou est remise à zéro chaque fois que la fourmi a terminé un tour. La liste-tabou constitue la mémoire de la fourmi.

Les traces de phéromones sont modélisées par les variables $\tau_{ij}(t)$ qui donnent l'intensité de la trace sur le chemin (i, j) à l'instant t. La probabilité de transition du sommet i vers le sommet j par la fourmi k est donnée

par :

$$P_{ij}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha} \cdot [\nu_{ij}]^{\beta}}{\sum\limits_{l \notin L_k(i)} ([\tau_{il}(t)]^{\alpha} \cdot [\nu_{il}]^{\beta})} & \text{si} \quad j \notin L_k(i) \\ 0 & \text{sinon} \end{cases} \tag{2.1}$$

où $L_k(i)$ représente la liste-tabou de la fourmi k située sur le sommet i et ν_{ij} représente une mesure de visibilité qui correspond à l'inverse de la distance entre les villes i et j.
α, β sont deux paramètres permettant de moduler l'importance relative des phéromones et de la visibilité.

La mise à jour des phéromones est effectuée une fois que toutes les fourmis sont passées par toutes les villes :

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \sum_{k=1}^{m} \triangle\tau_{ij}^{k} \tag{2.2}$$

où ρ est un coefficient représentant l'évaporation des traces de phéromones.
$\triangle\tau_{ij}^{k}$ représente le renforcement de l'arc (i,j) pour la fourmi k :

$$\triangle\tau_{ij}^{k} = \begin{cases} \frac{Q}{L^k} & \text{si la fourmis } k \text{ est passé par l'arc } (i,j) \\ 0 & \text{sinon} \end{cases} \tag{2.3}$$

avec Q une constante et L^k la longueur du chemin parcouru par la fourmi k.

L'algorithme suivant donne la structure générale de AS pour le TSP (noté AS-TSP).

Algorithme AS-TSP

1. Initialisation : $\tau_{ij} \rightarrow \tau_0 \forall (i,j) \in \{1,...,n\}^2$, placer aléatoirement chaque fourmi sur une ville
2. pour $t = 1$ à $t = t_{max}$ faire
3. pour chaque fourmi k faire
4. Construire un chemin $T^k(t)$ avec la règle de transition 2.1
5. Calculer la longueur $L^k(t)$ de ce chemin

6. fin pour
7. Soient T^+ le meilleur chemin trouvé et L^+ la longueur correspondante
8. Mettre à jour les traces de phéromones suivant la règle 2.2
9. fin pour
10. retourner T^+ et L^+

La valeur initiale des τ_{ij} est τ_0. Concernant le nombre de fourmis, il est raisonnablement proposé d'utiliser autant de fourmis que de villes ($m = n$).

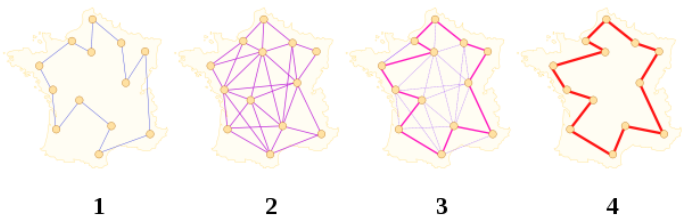

Fig 2.5 : L'algorithme « Ant System » optimisant le problème du voyageur de commerce :

1) une fourmi choisit un trajet, et trace une piste de phéromone.
2) l'ensemble des fourmis parcourt un certain nombre de trajets, chaque fourmi déposant une quantité de phéromone proportionnelle à la qualité du parcours.
3) chaque arête du meilleur chemin est plus renforcée que les autres.
4) l'évaporation fait disparaître les mauvaises solutions.

2.4.2 *MAX-MIN* Ant System (MMAS)

Stützle et Hoos ont introduit l'algorithme MMAS [24] qui, pour améliorer les performances de ACO, combine une exploitation améliorée des meilleures solutions trouvées durant la recherche avec un mécanisme efficace pour éviter une stagnation prématurée de la recherche. MMAS diffère en trois aspects clé de AS : Pour exploiter les meilleures solutions trouvées durant une itération ou durant l'exécution de l'algorithme, après chaque itération, une seule fourmi ajoute de la phéromone. Cette fourmi peut être celle qui a trouvé la meilleure solution depuis le début de l'exécution, ou bien celle durant le dernier cycle. Pour éviter une stagnation de la recherche, les valeurs possibles

de la trace de phéromone sur chaque composant de solution sont limitées à l'intervalle $[\tau_{min}, \tau_{max}]$. De plus, les traces de phéromone sont initialisées à τ_{max} pour assurer une exploration élevée de l'espace des solutions au début de l'algorithme.

2.4.3 Ant Colony System

L'algorithme Ant Colony System (ACS) a été introduit pour améliorer les performances du premier algorithme sur des problèmes de grandes tailles [11,12]. ACS est fondé sur des modifications du AS :
- la fourmi placée en i choisit la ville j par la règle de transition :

$$j = \begin{cases} argmax_{l \in J_k(i)}[\tau_{il}(t)] \cdot [\nu_{il}]^\beta & si \quad q \leq q_0 \\ J & sinon \end{cases} \tag{2.4}$$

où q est un réel aléatoirement tiré dans $[0, 1]$, q_0 est un paramètre ($q_0 \in [0, 1]$) et J est une ville choisie aléatoirement suivant la probabilité :

$$P_{ij}^k(t) = \frac{[\tau_{ij}(t)] \cdot [\nu_{ij}]^\beta}{\sum_{y \in J_i^k} [\tau_{iy}(t)] \cdot [\nu_{iy}]^\beta} \tag{2.5}$$

ou β est un paramètre servant à moduler la prise en compte des phéromones par rapport à la visibilité ;

- des listes de villes candidates sont utilisées pour accélérer le processus de construction d'un chemin ;
- Une heuristique locale est utilisée pour améliorer les solutions générées par les fourmis (2-opt ou 3-opt) ;
- La mise à jour des phéromones n'est faite qu'à partir du meilleur chemin généré ;
- une règle de mise à jour locale des phéromones est utilisée à chaque transition d'une fourmi.

Les résultats obtenus par ACS sur le TSP sont les meilleurs obtenus par les heuristiques à base de fourmis sans toutefois dépasser les meilleures heuristiques dédiées à ce problème.

Chapitre 3

Application et interprétation des résultats

Dans ce chapitre, nous donnerons une application de l'algorithme de colonie de fourmis «AS-TSP», et puis comparons les résultats obtenus avec autre algorithmes (algorithme de Dijkstra et algorithme de recherche tabou).

3.1 Exemples de l'application

NB : les v_i représentent les villes.

Exemple 1

Avec n villes, il y a $(n-1)!$ possibilités.
Dans l'exemple 1, il y a 5 sommets, soit 24 possibilités.

	v_1	v_2	v_3	v_4	v_5
v_1	0	8	4	2	3
v_2	8	0	7	1	6
v_3	4	7	0	6	6
v_4	2	1	6	0	4
v_5	3	6	6	4	0

Tab 3.1 : la matrice des distances de 5 villes.

-Algorithme de Dijkstra :

Parcours : $v_1 \quad v_4 \quad v_2 \quad v_5 \quad v_3 \quad v_1$.

Distance = 19.

-Algorithme de Recherche Tabou :

Parcours : $v_2 \quad v_1 \quad v_3 \quad v_5 \quad v_4 \quad v_2$.

Distance =19.

-Algorithme AS-TSP :

Parcours : $v_3 \quad v_1 \quad v_4 \quad v_2 \quad v_5 \quad v_3$.

Distance =19.

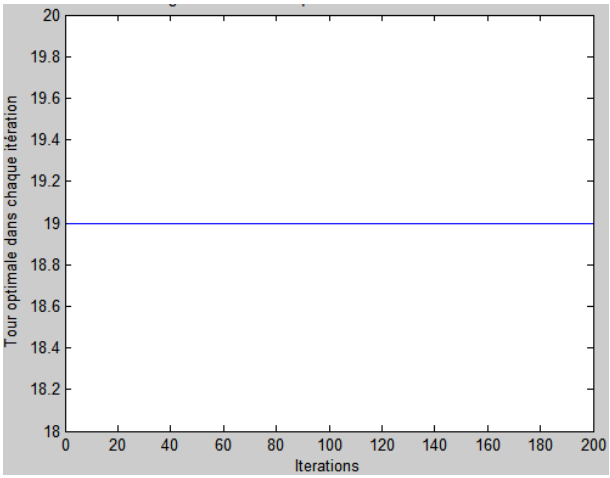

Fig 3.1 : Diagramme des résultats d'algorithme AS-TSP pour 5 villes.

Exemple 2 :

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

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0	5	8	4	3	2
v_2	5	0	4	2	1	3
v_3	8	4	0	9	5	4
v_4	4	2	9	0	9	8
v_5	3	1	5	9	0	4
v_6	2	3	4	8	4	0

Tab 3.2 : la matrice des distances de 6 villes.

-Algorithme de Dijkstra :

Parcours : $v_1 \quad v_6 \quad v_2 \quad v_5 \quad v_3 \quad v_4 \quad v_1$.

Distance = 18.

-Algorithme de Recherche Tabou :

Parcours : $v_1 \quad v_5 \quad v_4 \quad v_6 \quad v_2 \quad v_3 \quad v_1$.

Distance =18.

-Algorithme AS-TSP :

Parcours : $v_5 \quad v_2 \quad v_4 \quad v_1 \quad v_6 \quad v_3 \quad v_5$.

Distance =18.

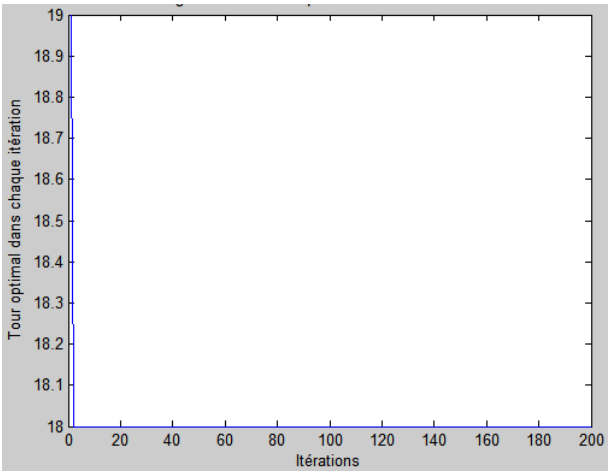

Fig 3.2 : Diagramme des résultats d'algorithme AS-TSP pour 6 villes.

Exemple 3 :

Nombre de villes : $n = 7$. Il y a 720 possibilités.

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v_1	0	5	9	6	3	5	9
v_2	5	0	8	8	5	9	2
v_3	9	8	0	1	6	7	3
v_4	6	8	1	0	4	2	9
v_5	3	5	6	4	0	2	8
v_6	5	9	7	2	2	0	3
v_7	9	2	3	9	8	3	0

Fig 3.3 : la matrice des distances de 7 villes.

-Algorithme de Dijkstra :

Parcours : $v_1 \quad v_5 \quad v_6 \quad v_4 \quad v_3 \quad v_7 \quad v_2 \quad v_1$.

Distance = 18.

-Algorithme de Recherche Tabou :

Parcours : $v_1 \quad v_5 \quad v_6 \quad v_4 \quad v_3 \quad v_7 \quad v_2 \quad v_1$.

Distance = 18.

-Algorithme de colonie de fourmis :

parcours :$v_7 \quad v_4 \quad v_3 \quad v_2 \quad v_1 \quad v_5 \quad v_6 \quad v_7$.

Distance = 18.

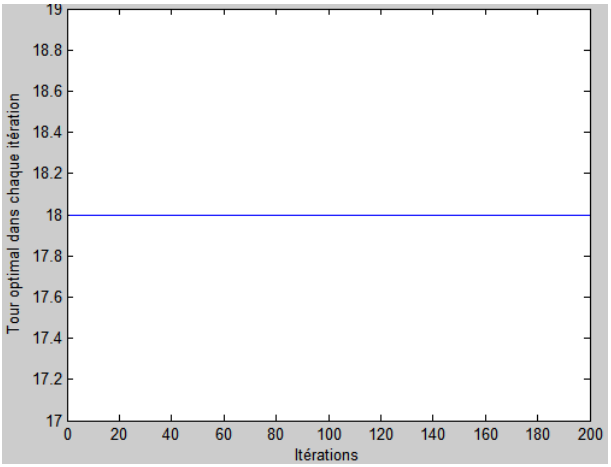

Fig 3.3 : Diagramme des résultats d'algorithme AS-TSP pour 7 villes.

Exemple 4 :

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

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
v_1	0	7	6	1	5	6	5	4
v_2	7	0	8	4	6	8	9	7
v_3	6	8	0	4	7	9	8	5
v_4	1	4	4	0	8	4	5	3
v_5	5	6	7	8	0	2	9	5
v_6	6	8	9	4	2	0	6	2
v_7	5	9	8	5	9	6	0	5
v_8	4	7	5	3	5	2	5	0

Tab 3.4 : la matrice des distances de 8 villes.

-Algorithme de Dijkstra :

Parcours : $v_1 \quad v_4 \quad v_8 \quad v_6 \quad v_5 \quad v_2 \quad v_3 \quad v_7 \quad v_1$.

Distance =35.

-Algorithme de Recherche Tabou :

Parcours : $v_6 \quad v_3 \quad v_7 \quad v_5 \quad v_2 \quad v_8 \quad v_1 \quad v_7 \quad v_6$.

Distance =33.

-Algorithme AS-TSP :

Parcours : $v_6 \quad v_5 \quad v_2 \quad v_4 \quad v_1 \quad v_7 \quad v_3 \quad v_8 \quad v_6$.

Distance = 33.

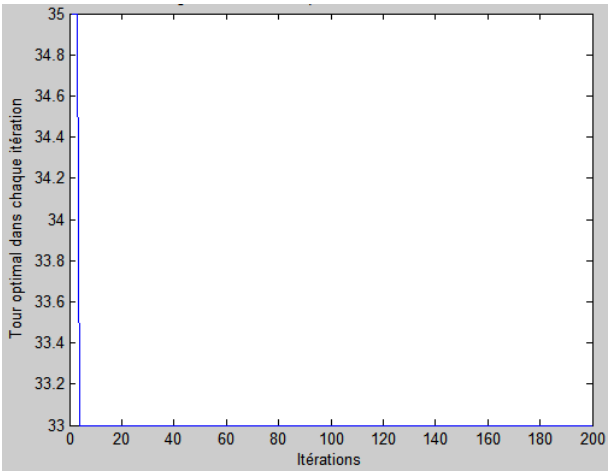

Fig 3.4 : Diagramme des résultats d'algorithme AS-TSP pour 8 villes.

Exemple 5 :

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

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
v_1	0	20	27	19	24	17	20	24	21
v_2	20	0	4	27	21	19	17	5	4
v_3	27	4	0	13	10	28	29	16	15
v_4	19	27	13	0	11	14	18	3	8
v_5	24	21	10	11	0	16	11	15	19
v_6	17	19	28	14	16	0	2	12	24
v_7	20	17	29	18	11	2	0	18	26
v_8	24	5	16	3	15	12	18	0	1
v_9	21	4	15	8	19	24	26	1	0

Tab 3.5 : la matrice des distances de 9 villes.

-Algorithme de Dijkstra :

Parcours : $v_1 \quad v_6 \quad v_7 \quad v_3 \quad v_5 \quad v_3 \quad v_2 \quad v_9 \quad v_8 \quad v_4 \quad v_1$.
Distance =73.

-Algorithme de Recherche Tabou :

Parcours :$v_1 \quad v_6 \quad v_7 \quad v_3 \quad v_5 \quad v_3 \quad v_2 \quad v_9 \quad v_8 \quad v_4 \quad v_1$.
Distance =71.

-Algorithme AS-TSP :

Parcours :$v_4 \quad v_8 \quad v_9 \quad v_2 \quad v_3 \quad v_5 \quad v_7 \quad v_6 \quad v_1 \quad v_4$.
Distance = 71.

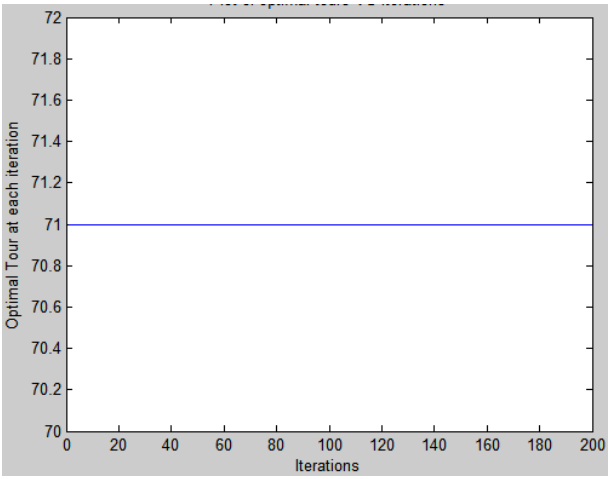

Fig 3.5 : Diagramme des résultats d'algorithme AS-TSP pour 9 villes.

Exemple 6 :

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

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
v_1	0	12	10	9	2	5	8	6	1	3
v_2	12	0	11	1	9	7	3	7	2	9
v_3	10	11	0	2	1	6	10	8	7	11
v_4	9	1	2	0	2	7	1	10	11	12
v_5	2	9	1	2	0	8	10	1	8	7
v_6	5	7	6	7	8	0	6	11	7	7
v_7	8	3	10	1	10	6	0	6	4	5
v_8	6	7	8	10	1	11	6	0	4	4
v_9	1	2	7	11	8	7	4	4	0	3
v_{10}	3	9	11	12	7	7	5	4	3	0

Tab 3.6 : la matrice des distances de 10 villes.

-Algorithme de Dijkstra :

Parcours : $v_1 \quad v_9 \quad v_2 \quad v_4 \quad v_7 \quad v_{10} \quad v_8 \quad v_5 \quad v_3 \quad v_6 \quad v_1$.

Distance = 27.

-Algorithme de Recherche Tabou :

Parcours :$v_5 \quad v_1 \quad v_2 \quad v_4 \quad v_3 \quad v_9 \quad v_{10} \quad v_6 \quad v_7 \quad v_8 \quad v_5$.

Distance =27.

-Algorithme AS-TSP :

Parcours : $v_3 \quad v_5 \quad v_8 \quad v_{10} \quad v_1 \quad v_9 \quad v_2 \quad v_4 \quad v7 \quad v_6 \quad v_3$.

Distance =26.

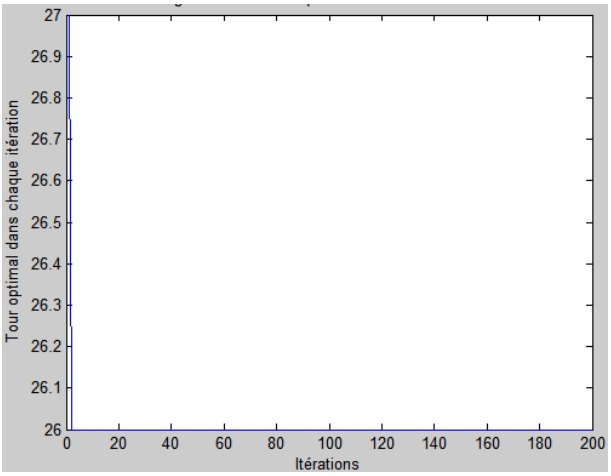

Fig 3.6 : Diagramme des résultats d'algorithme AS-TSP pour 10 villes.

Exemple 7 :

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

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}
v_1	0	4	1	6	5	10	9	5	9	2	4	10
v_2	4	0	7	4	10	2	8	7	9	4	8	9
v_3	1	7	0	6	4	7	4	8	5	5	7	10
v_4	6	4	6	0	1	4	7	3	3	8	7	6
v_5	5	10	4	1	0	8	9	3	8	2	9	2
v_6	10	2	7	4	8	0	4	5	10	1	10	2
v_7	9	8	4	7	9	4	0	1	10	7	3	5
v_8	5	7	8	3	3	5	1	0	1	10	4	1
v_9	9	9	5	3	8	10	10	1	0	2	8	8
v_{10}	2	4	5	8	2	1	7	10	2	0	5	5
v_{11}	4	8	7	7	9	10	3	4	8	5	0	7
v_{12}	10	9	10	6	2	2	5	1	8	5	7	0

Tab 3.7 : la matrice des distances de 12 villes.

- Algorithme de Dijkstra :
  Parcours : $v_1 \quad v_3 \quad v_5 \quad v_4 \quad v_8 \quad v_7 \quad v_{11} \quad v_{10} \quad v_6 \quad v_2 \quad v_{12} \quad v_1$.
  Distance = 40.
- Algorithme de Recherche Tabou :

Parcours : $v_1 \quad v_{10} \quad v_6 \quad v_4 \quad v_3 \quad v_7 \quad v_{11} \quad v_8 \quad v_2 \quad v_12 \quad v_9 \quad v_5 \quad v_1$.

Distance = 35.

-Algorithme AS-TSP :

Parcours :$v_{10} \quad v_6 \quad v_2 \quad v_4 \quad v_3 \quad v_12 \quad v_8 \quad v_3 \quad v_{11}v_1 \quad v_3 \quad v_9 \quad v_{10}$.

Distance =27.

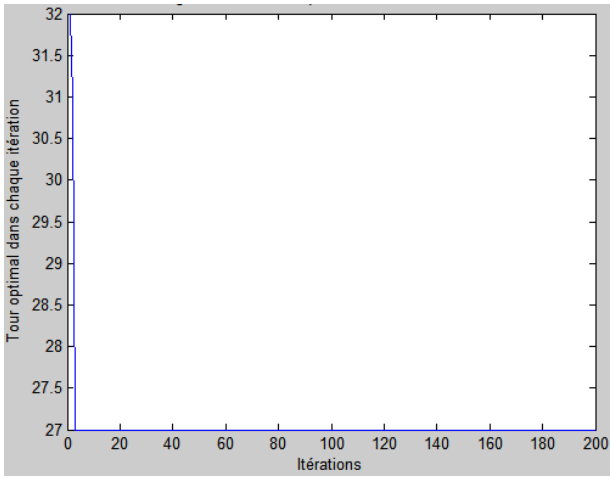

Fig 3.7 : Diagramme des résultats d'algorithme AS-TSP pour 12 villes.

Exemple 8 :

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

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}
v_1	0	34	7	10	22	13	7	16	5	37	11	5	15	31
v_2	34	0	30	25	16	1	32	17	12	24	15	33	30	17
v_3	7	30	0	33	38	11	21	7	35	22	30	24	2	23
v_4	10	25	33	0	34	1	26	22	31	37	22	20	4	10
v_5	22	16	38	34	0	13	29	8	39	23	1	9	6	1
v_6	13	1	11	1	13	0	36	35	15	4	30	36	35	40
v_7	7	32	21	26	29	36	0	30	40	14	23	38	11	18
v_8	16	17	7	22	8	35	30	0	35	15	36	17	40	31
v_9	5	12	35	31	39	15	40	35	0	34	15	33	32	21
v_{10}	37	24	22	37	23	4	14	15	34	0	5	38	24	39
v_{11}	11	15	30	22	1	30	23	36	15	5	0	19	39	33
v_{12}	5	33	24	20	9	36	38	17	33	38	19	0	38	30
v_{13}	15	30	2	4	6	35	11	40	32	24	39	38	0	19
v_{14}	31	17	23	10	1	40	18	31	21	39	33	30	19	0

Tab 3.8 : la matrice des distances de 14 villes.

- Algorithme de Dijkstra :

  Parcours : $v_1 \quad v_9 \quad v_2 \quad v_6 \quad v_4 \quad v_{13} \quad v_3 \quad v_8 \quad v_5 \quad v_{11} \quad v_{10} \quad v_7 \quad v_{14} \quad v_{12} \quad v_1$.

  Distance = 113.

-Algorithme de Recherche Tabou :

  Parcours : $v1 \quad v9 \quad v2 \quad v6 \quad v4 \quad v13 \quad v3 \quad v8 \quad v5 \quad v11 \quad v10v7 \quad v14 \quad v12 \quad v1$.

  Distance = 113.

- Algorithme AS-TSP :

  Parcours : $v7 \quad v_{10} \quad v_{11} \quad v5 \quad v_{14} \quad v_4 \quad v_6 \quad v_2 \quad v_9 \quad v_1 \quad v_{12} \quad v_8 \quad v_3 \quad v_{13} \quad v_7$.

  Distance =92.

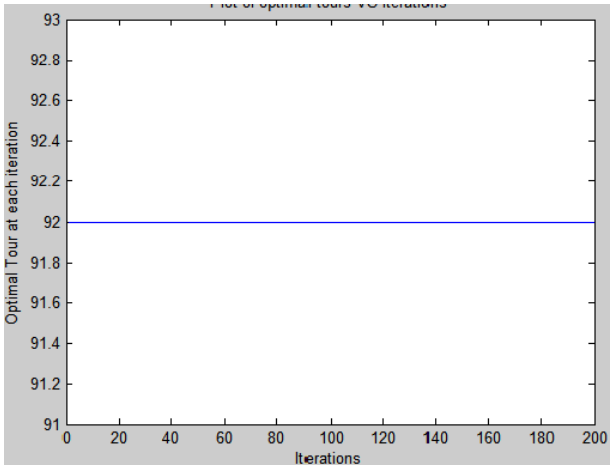

Fig 3.8 : Diagramme des résultats d'algorithme AS-TSP pour 14 villes.

Exemple 9 :

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

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
v_1	0	30	32	1	34	19	42	45	40	29	10	8	29	1	9
v_2	30	0	44	17	44	31	10	12	35	18	40	18	10	30	17
v_3	32	44	0	37	20	10	29	22	21	29	13	17	39	35	36
v_4	1	17	37	0	16	21	5	28	4	12	25	37	22	18	2
v_5	34	44	20	16	0	43	2	5	39	21	24	7	7	5	40
v_6	19	31	10	21	43	0	43	14	23	24	11	13	12	18	24
v_7	42	10	29	5	2	43	0	40	14	1	37	21	23	4	12
v_8	45	12	22	28	5	14	40	0	33	41	36	20	42	21	44
v_9	40	35	21	4	39	23	14	33	0	24	33	21	36	4	10
v_{10}	29	18	29	12	21	24	1	41	24	0	6	23	28	28	36
v_{11}	10	40	13	25	24	11	37	36	33	6	0	28	5	9	6
v_{12}	8	18	1	37	7	13	21	20	21	23	28	0	10	42	4
v_{13}	29	10	39	22	7	12	23	42	36	28	5	10	0	9	25
v_{14}	1	30	35	18	5	18	4	21	4	28	9	42	9	0	21
v_{15}	9	17	36	2	40	24	12	44	10	36	6	4	25	21	0

Tab 3.9 : la matrice des distances de 15 villes.

-Algorithme de Dijkstra :

Parcours : $v_1 \quad v_4 \quad v_{15} \quad v_{12} \quad v_5 \quad v_7 \quad v_{10} \quad v_{11} \quad v_{13} \quad v_{14} \quad v_9 \quad v_3 \quad v_6 \quad v_8 \quad v_2 \quad v_1$.

Distance =128.

-Algorithme Recherche Tabou :

Parcours : $v_4 \quad v_1 \quad v_{15} \quad v_{12} \quad v_5 \quad v_7 \quad v_{10} \quad v_{11} \quad v_{13} \quad v_{14} \quad v_9 \quad v_3 \quad v_6 \quad v_8 \quad v_2 \quad v_4$.

Distance =122.

-Algorithme AS-TSP :

Parcours : $v_9 \quad v_{14} \quad v_1 \quad v_4 \quad v_{15} \quad v_{12} \quad v_5 \quad v_7 \quad v_{10} \quad v_{11} \quad v_{13} \quad v_2 \quad v_8 \quad v_6 \quad v_3 \quad v_9$.

Distance =100.

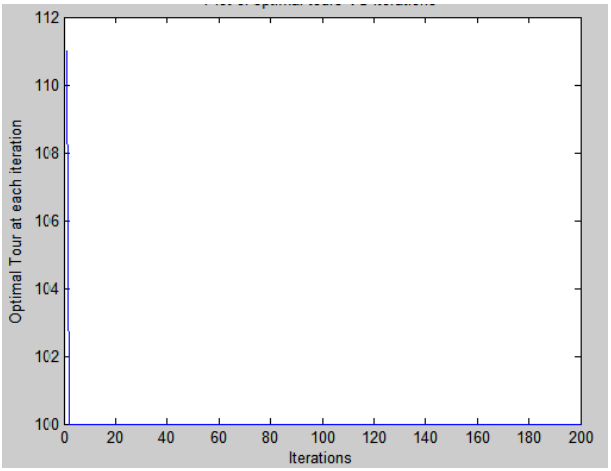

Fig 3.9 : Diagramme des résultats d'algorithme AS-TSP pour 15 villes.

Exemple 10 :

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

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}	v_{16}
v_1	0	12	2	21	42	9	40	34	40	34	23	18	33	26	24	7
v_2	12	0	39	14	1	30	29	3	27	10	13	40	23	14	31	1
v_3	2	39	0	36	13	19	18	39	48	2	15	15	3	27	47	24
v_4	21	14	36	0	42	24	15	50	16	15	4	2	49	32	19	1
v_5	42	1	13	42	0	2	13	25	46	39	10	47	33	19	26	33
v_6	9	30	19	24	2	0	33	42	24	4	2	25	26	36	22	49
v_7	40	29	18	15	13	33	0	18	26	35	6	1	12	25	9	36
v_8	34	3	39	50	25	42	18	0	36	3	43	11	31	43	18	30
v_9	40	27	48	16	46	24	26	36	0	27	20	26	16	14	30	30
v_{10}	34	10	2	15	39	4	35	3	27	0	45	41	14	32	36	38
v_{11}	23	13	15	4	10	2	6	43	20	45	0	18	41	1	25	41
v_{12}	18	40	15	2	47	25	1	11	26	41	18	0	6	20	39	25
v_{13}	33	23	3	49	33	26	12	31	16	14	41	6	0	26	29	38
v_{14}	26	14	27	32	19	36	25	43	14	32	1	20	26	0	4	13
v_{15}	24	31	47	19	26	22	9	18	30	36	25	39	29	4	0	40
v_{16}	7	1	24	1	33	49	36	30	30	38	41	25	38	13	40	0

Tab 3.10 : la matrice des distances de 16 villes.

-Algorithme de Dijkstra :
Parcours : v_1 v_3 v_{10} v_8 v_2 v_5 v_6 v_{11} v_{14} v_{15} v_7 v_{12} v_4 v_{16} v_9 v_{13} v_1.
Distance =112 .
-Algorithme de Recherche Tabou :
Parcours : v_1 v_3 v_{10} v_8 v_2 v_5 v_6 v_{11} v_{14} v_{15} v_7 v_{12} v_4 v_{16} v_9 v_{13} v_1.
Distance =112.
-Algorithme AS-TSP :
Parcours : v_{10} v_8 v_2 v_5 v_6 v_{11} v_{14} v_{15} v_7 v_{12} v_4 v_{16} v_1 v_3 v_{13} v_9 v_{10}.
Distance =84.

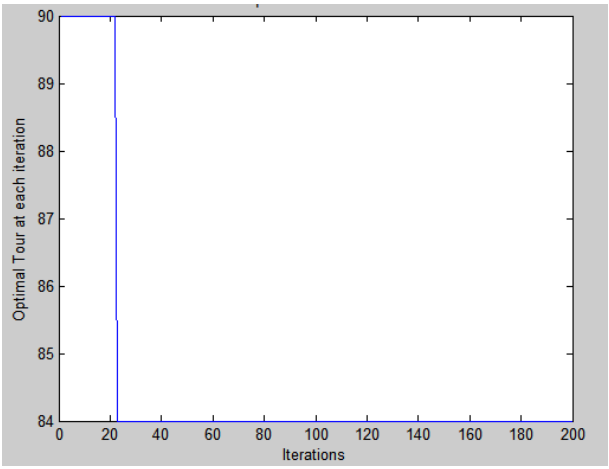

Fig 3.10 : Diagramme des résultats d'algorithme AS-TSP pour 16 villes.

3.2 Comparaison entre les résultats d'algorithmes

Nous comparons les résultats de distance de parcours qui obtenue dans chaque algorithme.

Nombre de villes	Dijkstra	Recherche Tabou	AS-TSP
5	19	19	19
6	24	18	18
7	18	18	18
8	35	33	33
9	73	71	71
10	27	27	26
12	40	35	27
14	113	113	92
15	128	122	100
16	112	112	84

Tab 3.11 :Résultats de comparaison.

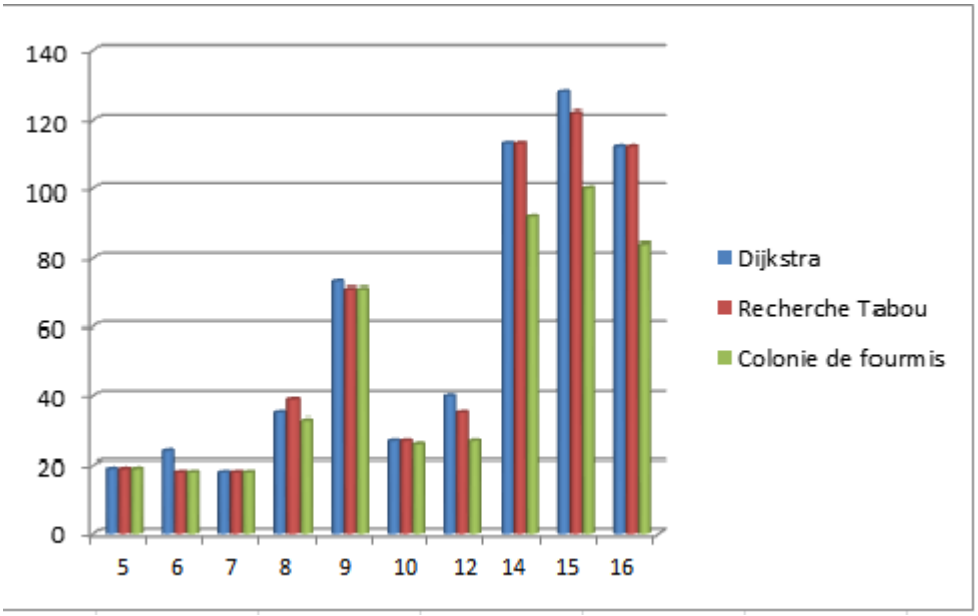

Fig 3.11 : Représentation des résultats de comparaison

3.2.1 Analyse de l'histogramme :

Au cours de notre étude de l'algorithme de colonie de fourmis appliqué au problème de voyageur de commerce (TSP) ; nous découvrons que cette méthode est meilleure que la méthode de Dijkstra et la recherche tabou, nous comparons les solutions obtenues à partir de l'application de l'algorithme de colonie de fourmis et l'algorithme de recherche tabou et l'algorithme de Dijkstra.

Après avoir tracé de l'histogramme, qui représente la variation de la distance en ce qui concerne le nombre des villes pour chacun de l'algorithme de colonie de fourmis (colonne vert) et l'algorithme de Dijkstra (colonne bleue), et l'algorithme de recherche tabou (colonne rouge). Nous avons observé une différence dans les colonnes, pour un petit nombre de villes, la colonne Rouge presque égal à la colonne bleue, et la colonne vert aussi, car les résultats obtenus par les trois méthodes sont presque égaux .

Lorsque vous avez un grand nombre de villes augmente la différence entre la colonne bleue, la colonne rouge, et la colonne vert, où l'algorithme de colonie de fourmis et l'algorithme de recherche tabou donnent une distance plus petite par apport à l'algorithme de dijkstra.

Et par conséquent, nous pouvons dire que l'algorithme de colonie de fourmis et l'algorithme de Recherche Tabou est efficace par apport à l'algorithme de dijkstra , telle que l'algorithme de colonie de fourmis est plus efficace par rapport à l'algorithme de recherche tabou.

3.3 Inconvénients et Avantages d'algorithme de colonie de fourmis

1. Avantages :
   - Très grande adaptabilité.
   - Parfait pour les problèmes basés sur des graphes.
2. Inconvénients :
   - Un état bloquant peut arriver.
   - Temps d'exécution parfois long.
   - Ne s'applique pas à tous type de problèmes.
   - Les Résultats sont pseudo-optimale.

Conclusion

L'optimisation combinatoire est l'une des branches les plus difficiles en optimisation, et pour cela l'application des méthodes exactes est presque impossible.
Dans ce mémoire nous avons opté pour une méthode méthaheuristique pour parcourir des tailles plus grande dans un temps réduit.

Bibliographie

[1] Beckers, R., Deneubourg, J. L., Goss, S. (1992). Trails, and UTurns in the Selection of a Path by the Ant Lasius Niger. J. Theor. Biol., 159 :397-415.

[2] Bullnheimer, B., Hartl, R.F.,& Strauss, C. (1999). An Improved Ant system Algorithm for the Vehicule Routing Problem. Annals of Operations Research, 89, 319-328.

[3] Brossut, R. (1996). Phéromones, la communication chimique chez les animaux . CNRS éditions, Berlin.

[4] Colorni,et al. (1991). Distributed Optimization by Ant Colonies . In (Varela and Bourgine, 1991), pages 134–142.

[5] Deneubourg, J.L., Goss, S., & Verhaeghe, J.C. (1983). Probabilistic behaviour in ants. Journal of Theorical Biology, 105, 259-271.

[6] Dréo, Pétrowski, Siarry, and Taillard. (2003). Métaheuristiques pour l'optimisation difficile. Eyrolles.

[7] Dréo, J. (2004). Adaptation de la méthode des colonies de fourmis pour l'optimisation en variables continues. Application en gniée biomédical.

[8] Dorigo, M. (1992). Optimization, learning, and natural algorithms, Ph.D. dissertation (in Italian), Dipartimento di Elettronica, Politecnico dimilano, Italy.

[9] Dorigo, M., & Gambardella, L. (1996). The Ant System : Optimization by a Colony of Cooperating Agents . IEEE Transactions on Systems, Man, and Cybernetics-Part B, 26(1) :29–41.

[10] Dorigo, M., & Gambardella, L.M. (1997). Ant Colony System : A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation, 1(1), 53-66.

[11] Dorigo, M. & Gambardella, L. (1997). Ant Colonies for the Traveling Salesman Problem . BioSystems, 43 :73–81.

[12] Dorigo, M., & Gambardella, L. (1997). Ant Colony Sytem : A Cooperative Learning Approach to the Travelling Salesman Problem . IEEE Transactions on Evolutionary Computation, 1(1) :53–66.

[13] Gambardella, L., Taillard, E., & Dorigo, M. (1999). Ant Colonies for the Quadratic Assignment Problem. Journal of the Operational Research Society, 50, 167-176.

[14] Glover, F. (1989). Future Paths for Integer Programming and Links to Artificial Intelligence. Computers and Operations Research, 13(5), 533-549.

[15] Goss, S., Aron, S., Deneubourg, J. L., and Pasteels, J. M. (1989). Self-Organized Shortcuts in the Argentine Ant. Naturwissenchaften, 76 :579-581.

[16] Holland, John H. (1992). Adaptation in Natural and Artificial Systems : An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. The MIT Press.

[17] Kirkpatrick, S., Gelatt, C.D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. Science, 220(4598), 671-680.

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

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

[20] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., & Teller, E. (1953). Equation of State Calculations by Fast Computing Machines. J. Chem. Phys., 21, 1087-1092.

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

[22] Papadimitriou, C. (1994). Computational Complexity. AddisonWesley.

[23] Solnon, C. (2002). Ants can Solve Constraint Satisfaction Problems. IEEE Transactions on Evolutionary Computation, 6(4), 347-357.

[24] Stützle, & Hoos. (2000). $MAX-MIN$ Ant System. Future Generation Computer Systems, 16(8) :889–914.

Résumé :

Nous avons dans ce travail présenté l'optimisation combinatoire ainsi que la classification de ses problèmes, certains problèmes classiques, et les méthodes de résolution.
Nous avons étudié l'un des problèmes (problème de voyageur de commerce), où nous avons pris la métaheuristique "algorithme de colonie de fourmis" pour résoudre ce problème, et nous avons utilisé un programme de cette algorithme et nous avons appliqué sur différents tailles de villes, ainsi que nous avons comparé les résultats avec autres méthodes classiques.

mots clés : Optimisation combinatoire, problème de voyageur de commerce, métaheuristique, l'algorithme de colonie de fourmis

Abstract :

We present in this work, combinatorial optimisation and its classification problems as well as some classical problems addition to some methods to solve this problems. we have been studied one of them the traveling salesman problem, where we used the metaheuristic "ant colony algorithm" to solve this problem, we used a program of this algorithm and applied to different lenthg of cities, as well as compare with other methods.

Keywords : Combinatorial optimization, traveling salesman problem, metaheuristic, ant colony algorithm.