Approche Heuristique Multi Colonie Des Fourmis Pour La Résolution du Problème de Voyageur de Commerce

Mathurin Soh, Baudouin Nguimeya Tsofack, Clémentin Tayou Djamegni

Research Unit in Fundamental Informatics, Engineering and Applications
University of Dschang
P.O. Box 67 Dschang, Cameroon
mathurinsoh@gmail.com ; mathurin.soh@univ-dschang.org
nguimeyabaudoin@yahoo.fr
dtayou@gmail.com

RÉSUMÉ. Dans ce papier, nous proposons une nouvelle approche de solution approchée au problème de voyageur de commerce (Travelling Salesman Problem in english), dont on ne connait pas d'algorithme exact permettant de trouver une solution en un temps polynomial. L'approche proposée est basée sur l'optimisation par des fourmis. Elle met plusieurs colonies en compétition pour la recherche des solutions améliorées (en temps d'exécution et en qualité de la solution) aux instances de TSP de grandes tailles, et permet d'explorer plus efficacement les plages de solutions possibles. Les résultats issus de nos expériences nous montrent, que l'approche permet de déterminer des résultats meilleurs par rapport aux autres heuristiques (ACS-LKH, et AG-LKH), issues de la littérature, notamment en qualité des solutions obtenues et en temps d'exécution.

ABSTRACT. In this paper, we propose a new approach to solving the Travelling Salesman Problem (TSP), for which no exact algorithm is known that allows to find a solution in polynomial time. The proposed approach is based on optimization by ants. It puts several colonies in competition for improved solutions (in execution time and solution quality) to large TSP instances, and allows to efficiently explore the range of possible solutions. The results of our experiments show that the approach leads to better results compared to other heuristics from the literature, especially in terms of the quality of solutions obtained and execution time.

MOTS-CLÉS : Colonie, Fourmis, Heuristique Multi-colonie, Problème du Voyageur de Commerce

KEYWORDS : Ants, Colony, Heuristic, Multi-colony, Travelling Salesman Problem

1. Introduction

Le problème du voyageur de commerce ou *Traveling Salesman Problem* (TSP) est l'un des problèmes d'optimisation combinatoire. Son énoncé est simple, et pourtant il reste l'un des problèmes les plus difficiles de la recherche opérationnelle car considéré comme NP-diificile[1]. Le TSP est un problème classique en optimisation combinatoire [1] qui, a fait l'objet d'études intensives tant dans le domaine de la recherche opérationnelle qu'en informatique depuis les années 1950. Ce qui a eu pour conséquence qu'un grand nombre de techniques ont été développées pour résoudre ce problème. Une grande partie des travaux sur le TSP n'est pas motivée par des applications pratiques, mais plutôt par le fait qu'il offre un cadre idéal pour les plate-formes d'étude des méthodes générales pouvant être appliquées à un large éventail de problèmes d'optimisation combinatoire. Dans le TSP, un certain nombre de villes sont indiquées, et sont liées à d'autres par des arcs. Un vendeur de commerce commence sa tournée depuis n'importe quelle ville et ne voyageant qu'une seule fois dans chaque ville, il retourne à la ville de départ encore une fois. L'idée du problème est de trouver le chemin le plus court du vendeur de commerce à partir d'une ville donnée, en visitant n villes une seule fois et en retournant finalement dans la ville d'origine.

Malgré la multitude des méthodes existantes pour résoudre le TSP fondamental ou appliqué, aucune des méthodes n'a pu donner à ce jour une issue favorable pour des instances de taille elevée[1, 2]. En effet on se trouve généralement confronté à une explosion combinatoire et une limitation des ressourcés mémoires des ordinateurs quand la taille des instances augmente. Pourtant les problèmes se rapportant au TSP sont de plus en plus fréquents dans notre société et doivent être résolus. Pour la plupart des instances de TSP de grande taille, la solution optimale est rarement atteinte même par les heuristiques les plus performantes de l'heure développées à ce sujet. Dans ce travail, nous sommes interessé par l'utilisation de l'optimsation par colonies de fourmis. Il est question pour nous de proposer une autre solution à ce problème. Nous developpons ainsi une nouvelle heuristique faisant intervenir plusieurs colonies de fourmis.

Dans la suite de ce papier, nous présentons dans la section 2, le TSP et ses applications. La section 3 est consacrée à la revue de littérature sur la résolution du TSP par les heuristiques. Dans la section 4 nous présentons notre contribution. La section 5 est consacrée aux résultats expérimentaux, élucidés à partir des tests et discussions qui sont effectués avec des instances de TSP tirées de la bibliothèque TSPLIB. La section 6 conclut notre travail.

2. Le Problème de Voyageur de Commerce

Le problème du voyageur de commerce (PVC ou TSP en anglais) est défini comme suit : étant donné n points (des villes) et les distances séparant chaque point, trouver un chemin de longueur totale minimale qui passe exactement une fois par chaque point et revienne au point de départ[1, 10]. La distance peut être vue également comme le coût en général. Ce problème d'optimisation combinatoire, consiste donc à rechercher la meilleure solution parmi plusieurs choix possibles. Cependant, il est facile à énoncer mais difficile à résoudre. Le problème consiste à déterminer un tour ou circuit hamiltonien, c'est-à-dire ne passant qu'une et une seule fois par les n villes, et qui soit de coût

minimum. Il est classé dans les problèmes NP-difficile [1, 2, 3], car on ne connaît pas de méthode de résolution permettant d'obtenir des solutions exactes en un temps raisonnable pour de grandes instances (grand nombre de villes) du problème. Pour ces grandes instances, on se contente très souvent des solutions approchées car après une énumération explicite, le nombre de chemins hamiltoniens est égal à (n-1)!/2 [10].

Mathématiquement, le TSP peut être formulé en ces termes: Soit n villes et C_{ij}, le coût ou la distance correspondant au trajet $i \rightarrow j$. Soit la variable X_{ij}, qui vaut 1 si le tour contient le trajet $i \rightarrow j$, et 0 autrement. Le problème s'écrit [6]:

$$\begin{cases} MinZ = \sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} X_{ij} \\ \sum_{J=1}^{n} X_{ij} = 1 \quad \forall i & (1) \\ \sum_{i=1}^{n} X_{ij} = 1 \ \ \forall j & (2) \quad [1] \\ \sum_{i \in Q}^{n} \sum_{j \in Q}^{n} X_{ij} \geq 1 \ \ \forall Q & (3) \\ X_{ij} \in \{0,1\} \forall i, \forall j \end{cases}$$

Où Q représente un sous-ensemble de 1,..,n et son complémentaire. Les contraintes (3) expriment que la permutation des n villes doit être un tour, c'est-à-dire qu'il ne peut pas exister de sous-tour.

Le TSP trouve des applications dans les transports, les réseaux et la logistique[6]. Par exemple, trouver le chemin le plus court chemin pour les bus de ramassage scolaire ou dans l'industrie, pour la collecte/distribution, pour trouver la plus courte distance que devra parcourir le bras mécanique d'une machine pour percer les trous d'un circuit imprimé[6, 7, 10].

3. Position du problème

Les algorithmes existants pour résoudre le TSP ne sont pas assez efficaces. Lorsque le nombre de villes augmente, le temps ou l'espace requis pour résoudre le problème augmente de façon exponentielle. Le système des colonies de fourmis résout le problème dans un délai raisonnable et produit des solutions optimales[2]. Mais lorsque la dimension du problème augmente, il est long de produire le résultat qui n'est pas souvent optimal. Ainsi, une modification est nécessaire pour résoudre les grandes instances du problème du voyageur de commerce.
Dans ce travail, nous proposons une approche de résolution du TSP sur la base dune heuristique doptimisation par colonies des fourmis (OCF) dont plusieurs colonies sont mis en compétition pour trouver des résultats compétitifs et capables de minimiser les temps d'exécutions connus ou de donner des solutions optimales ou presque aux instances dont une solution reste difficile à atteindre par les méthodes actuelles.
Cette nouvelle approche permettrait dexplorer plus efficacement les plages de solutions possibles, et déterminer les techniques ditération satisfaisantes pour construire le plus

rapidement possible le chemin menant à la solution optimale parmi des milliers de chemins possibles.
Nous nous intéressons particulièrement à l'approche multi-colonies pour plusieurs raisons: Premièrement, nous souhaitons offrir une plus grande exploration en terme de possibilité de déploiement des fourmis, ce qui n'était pas le cas avec une seule colonie. Deuxièmement, nous souhaitons obtenir une multiplicité des solutions potentielles au problème au cours du processus d'optimisation. Troisièmement, nous voulons obtenir plus rapidement de bonnes solutions, en raison de la compétition instaurée au niveau des colonies.

4. Etat de l'art

Plusieurs approches ont été utilisées pour résoudre le TSP. Des méthodes exactes en passant par les méthodes approchées. L'approche la plus simple est l'approche par les méthodes exactes qui consiste à énumérer toutes les solutions possibles et à prendre la meilleure solution. Ces approches exactes ne sont efficaces que pour des instances de petite taille dont l'énumération explicite des solutions est possible en temps raisonnable [2, 7, 8]. Pour les instances de grandes tailles, la plupart des travaux sont basés sur les méthodes heuristiques et méta heuristiques. Dans la littérature, on compte deux types d'heuristiques : Celles qui construisent une tournée en ne partant de rien et celles qui sont efficaces pour améliorer une tournée déjà existante appelée heuristique d'amélioration de tour [4]. Parmi les recherches sur les méthodes heuristiques pour résoudre le TSP, on note les résultats de Dorigo et al [2, 6] qui, s'inspirant du comportement collectif de fourmis et de leurs similarités avec le TSP, ont proposés le premier algorithme des colonies de fourmis pour un TSP : "'*l'Ant System (AS)* "'[2]. Malheureusement, le choix purement probabiliste de l'AS le rend moins efficace en termes de diversification favorisant ainsi les risques de convergence prématurée. Pour améliorer les performances de l'AS sur des problèmes de grande taille, Dorigo et al [2], modifient la règle de transition d'AS pour offrir un meilleur équilibre entre l'exploration de nouvelles zones de l'espace de solutions et l'exploitation des connaissances accumulées sur le problème pour intensifier la solution. Ils introduisent à cet effet une nouvelle règle qui dépend d'un paramètre $q_0 (0 < q_0 < 1)$. Q est un nombre généré aléatoirement, distribué uniformément entre $[0, 1]$. Avec cette nouvelle version, le choix des prochaines villes n'est pas seulement probabiliste mais aussi suivant un paramètre q. L'algorithme ainsi obtenu est appellé OCF [3]. Seulement OCF a montré quelques faiblesses en termes d'exploration de nouvelles pistes et d'intensification des solutions.

À côté de ces heuristiques dites heuristiques de construction, nous avons les heuristiques d'amélioration comme l'Heuristique K-opt (K =2,3,) qui améliore les solutions en testant si les permutations de k trajets produisent des tournées plus optimales [1] ou l'heuristique de Lin and Kernighan (LK) [3] qui est une amélioration de l'heuristique K-opt avec k variables, ou encore l'heuristique de Lin and Kernighan modifiée par Helsgaun (LKH) . Ces heuristiques améliorent efficacement les solutions en partant d'une solution déjà connue. Une autre classe d'heuristique est les heuristiques hybrides qui gagne de plus en plus en popularité ces dernières années. Ces heuristiques hybrides se subdivisent en plusieurs groupes. D'un côté nous avons l'hybridation des heuristiques de construction entre elles qui consiste à faire remplacer un mécanisme d'une heuristique par un mécanisme d'une autre heuristique pour combler ce qui est limité chez l'une par ce qui est vu comme avantage chez l'autre. A ce propos, on peut citer lhybridation (AG-OCF) [4, 10]. De l'autre coté, nous avons l'hybridation d'heuristiques de construction (AG,

OCF) avec les heuristiques des recherches locales ou heuristiques d'amélioration (LK, LKH, RL). Un exemple de cette technique a été celle proposée par [10]. Ce travail propose pour la resolution du TSP, deux nouvelles heuristiques hybrides denommées «AG-LK» et «ACS-LK» résultant respectivement de l'hybridation d'un Algorithme Génétique (AG) et de l'heuristique (LK), et la seconde entre l'algorithme des colonies de fourmi (ACS) et l'heuristique (LK). Cependant les auteurs utilisent une seule colonie. Les résultats obtenus de cette expérience ont démontré à suffisance l'efficacité de ces approches hybrides sur plusieurs instances de TSP [10][3]. Malheureusement, le temps de résolution issu de cette expérience reste énorme.

Pour améliorer l'efficacité des heuristiques hybrides proposées dans [4], Nguimeya et al en 2016 implémentent deux heuristiques hybrides pour le TSP. La première entre un Algorithme Génétique (AG) et l'heuristique (LKH) qui est une amélioration de LK par Helsgaun [3] et la seconde entre l'algorithme des colonies de fourmi (ACS) et l'heuristique (LKH). Les heuristiques obtenues fut appelées respectivement «AG-LKH» et «ACS-LKH». Les stratégies d'hybridation diffèrent d'un auteur à l'autre [4] .

Comme nous pouvons le constater, la recherche des solutions au TSP est une quête permanente dans le domaine de l'informatique notamment dans la recherche opérationnelle. Plusieurs auteurs ont déjà travaillé dans ce sens en abordant diverses approches comme le témoigne bien cette revue de l'art. Seulement aucune de ces méthodes n'a pu venir à bout du TSP. Dans la section suivante, nous proposons une nouvelle heuristique hybride pour le TSP.

5. Proposition d'une approche d'optimisation multi colonies de fourmis pour le TSP

5.1. Idée de base

Pour atteindre les objectifs cités plus haut, nous présentons l'OCF de base proposé par Dorigo et al dans [2]. Ensuite, nous modifions cet algorithme en utilisant plusieurs colonies. Cela nous conduit à une nouvelle version que nous avons appelée MOCF (OCF-multi colonies). MOCF propose une nouvelle façon de faire itérer les fourmis des différentes colonies en définissant plus ou moins un pas d'itération afin d'optimiser la décision des fourmis artificielles à prendre préférentiellement les bons chemins tout en évitant les pièges de l'optimum local. Les stratégies d'échanges ou de communication entre les fourmis lors de leurs déplacements se feront en temps réel afin de favoriser la détection des échecs ou mauvais chemins par les fourmis le plus rapidement possible. La mise à jour des phéromones par les fourmis se fait comme dans l'OCF [2]. L'algorithme obtenu est l'implémenté en langage C et testé sur des instances du TSP tirées de la bibliothèque TSPLIB [5].

5.2. Description ou Mécanisme des approches OCF

Pour l'étude, nous considérons que notre problème est modélisé sous forme de graphe. Chaque noeud du graphe représente une ville ou tâche à effectuer, d_{ij} la distance entre les villes i et j, et le couple (i, j) le chemin entre ces deux villes. Nous allons également considérer les éléments suivants :

– La fourmi représente une solution au problème traité ;

– Les phéromones informatiques sont des valeurs associées à des solutions trouvées. Ces valeurs dépendent de la qualité des solutions.

5.2.1. Etape 1 : Construction du trajet par chaque fourmi (Solution)

Pour aller d'un noeud (ville) du graphe à un autre, l'itération et le déplacement des fourmis se fait comme suit : Initialement (au temps $t = 0$), l'algorithme positionne m fourmis sur n villes et l'intensité de la trace pour toutes les paires de villes (i, j) est fixée à une petite valeur positive To dans la matrice phéromone. Une liste taboue est maintenue pour garantir qu'une ville ne peut être visitée deux fois durant le même tour. Chaque fourmi k va donc posséder sa propre liste de villes Vk-taboue qui gardera en mémoire les villes déjà visitées. Durant une itération de l'algorithme, plusieurs fourmis visitent à tour de rôle une séquence de villes. Un cycle (NC) est terminé au moment où la dernière des m fourmis a terminé sa construction.

A t $\neq 0$, à partir d'une séquence de villes déjà visitées, chaque fourmi k choisit la prochaine ville à ajouter à sa liste Vk-taboue en utilisant une règle probabiliste donc la formule est donnée dans l'équation (1).

$$\eta_{ij} = \frac{1}{d_ij}$$

$$\mathrm{p}\ {}^{k}_{ij} = \frac{[\eta_{ij}]^{\beta}[\tau_{ij}(t)]^{\alpha}}{\Sigma u \in v_k [\eta_{iu}]^{\beta}[\tau_{iu}(t)]^{\alpha}} \quad (1)$$

$$\tau_{ij}(t+1) = \rho\ \mathrm{P}\ {}^{k}_{ij}(t) + \triangle\ \tau_{ij} \quad (2)$$

[2]

Celle-ci est basée sur un compromis entre la visibilité ($\frac{1}{d_ij}$) et la quantité de phéromone (t_{ij}) présente entre i et j. Les paramètres α et β sont utilisés afin de déterminer respectivement l'importance relative de l'intensité de la trace et de la visibilité dans la construction d'une solution. Ce mode d'itération permet de privilégier les plus courts chemins puisque les fourmis auront besoin de moins d'itérations pour en arriver au bout.

5.2.2. Etape 2 : Dépôt de phéromones et évaluation des solutions

Quand une fourmi se déplace d'une ville i à la ville j, elle laisse une certaine quantité de phéromone (valeur) sur l'arc (i, j). Une matrice dite matrice de phéromone enregistre l'information sur l'utilisation de l'arc (i, j). À chaque étape du tour, la matrice est mise à jour de sorte que plus cette utilisation a été importante dans le passé, plus élevée est la probabilité que ces arcs soit soient utilisés à nouveau dans le futur. Dans cette version d'OCF [2], il y a une mise à jour globale de l'intensité de la trace de phéromone après chaque transition en fonction de l'évaluation des solutions trouvées. Ainsi, une fourmi se déplaçant d'un nud i à j pourra signaler aux fourmis suivantes qui arriveront en i si le déplacement en j est un déplacement justifié. L'équation (2) permet la mise à jour des phéromones.

$$\tau_{ij}(t+1) = \rho\ \tau_{ij}(t) + \triangle\ \tau_{ij}$$

[3]

5.2.3. Algorithme d'Otimisation par Colonie de Fourmis (OCF)

Algorithme 5.1 : ALGORITHME OCF

```
1  Début
2      Entrée m : nombre de fourmis par colonies ;
3      n : nombre de villes ;
4      N_c ← 0 ;
5      Initialiser listeTaboue // avec la ville de départ de chaque fourmi ;
6      Initialiser la matrice τ_ij
7      Tantque ( N_c < N_max) et ( convergence non atteinte) Faire
8          pour i ← 1 à n faire
9              pour j ← 1 à m faire
10                 Sélectionner la ville j à être ajoutée ou tour en cour selon la
                     formule p_ij^k (t) ;
11                 Effectuer la mise à jour locale de la trace selon la paire de
                     villes(i,j) ;
12             fin pour
13         fin pour
14     Fintantque
15     pour chaque fourmi k faire
16         Evaluer la solution k pour chacune des étapes ;
17         Inserer la solution k dans la listeTABOU ;
18         Effectuer la mise à jour Globale de la trace selon la meilleure solution du
             cycle ;
19     fin pour
20     NC ← NC + 1;
21 Fin
```

5.3. Approche d'optimisation multi colonie de fourmi pour le TSP

Dans cette section, nous présentons une approche nommée Optimisation Multi Colonie de Fourmi pour le TSP (MOCF), basée sur l'optimisation par colonies de fourmis [2] entre plusieurs colonies. Pour le faire, nous commençons par présenter quelques caractéristiques et principes des approches OCF, ensuite, nous abordons la description de notre approche de résolution MOCF et enfin, nous présentons l'algorithme MOCF.

Nous mettons en compétition plusieurs colonies de fourmis artificielles en fonction de la taille du problème pour gagner en termes d'exploration des solutions. Ainsi, les différentes colonies construisent de façon indépendante leur tournée et collaborent à travers une sorte de mémoire partagée (matrice globale de phéromones) en stimulant un pseudo parallélisme pour atteindre l'objectif commun. En effet, chaque colonie est placée dans une ville de départ, un pas de déplacement est autorisé pour chaque colonie à tour de rôle. Si une fourmi appartenant à colonie pendant son itération trouve une solution meilleure que celle connue par les autres fourmis des différentes colonies, elle informe rapidement les autres en mettant à jour la solution à travers la matrice globale de communication. Pour optimiser le temps de solution, les informations concernant la matrice de phéromones sont données en temps réel immédiatement après chaque transition et non différé.

5.4. Nouvelle Approche OCF par une multi colonies(MOCF)

5.4.1. Description de l'approche MOCF

Notre approche dénommée MOCF est basée sur les approches OCF à la différence que MOCF utilise plusieurs colonies de fourmis afin de permettre une meilleure exploration des chemins et optimise les solutions trouvées par une autre heuristique d'amélioration LKH. En effet, l'un des défauts majeurs dans OCF mono colonie [2] est la difficulté pour celle-ci à pouvoir explorer efficacement les plages de solutions surtout pour les instances de grandes tailles. Avec MOCF, nous comptons résoudre le problème en utilisant plusieurs colonies. Les différentes colonies de l'algorithme MOCF sont indépendantes et construisent indépendamment leurs solutions tout en travaillant en étroite collaboration pour atteindre l'objectif final à travers les mécanismes de communication rendu possible grâce à une matrice de communication et des valeurs appelées phéromone.

Chaque colonie (les fourmis) représente une solution au problème traité. La matrice de phéromones est commune à toutes les colonies. Elle est utilisée comme mémoire partagée pour permettre aux fourmis de communiquer ensemble afin de guider la recherche des autres. Cette matrice est dynamique et est utilisée par les colonies pour construire de manière incrémentale les solutions. Chaque fourmi a un comportement simple, mais les échanges entre fourmis sont réalisés par l'intermédiaire de la matrice de phéromones, permettent de produire progressivement une solution proche, voir optimale. Les fourmis communiquent indirectement via des modifications dynamiques des pistes de phéromones et construisent ainsi une solution à un problème, en s'appuyant sur leurs expériences collectives.

5.4.2. Description du principe de l'algorithme MOCF

Etape 1 : Initialisation de l'algorithme: Initialement, il n'y pas de phéromone sur les arcs. La matrice globale de phéromones est initialisée avec les valeurs 0 pour chaque paire de noeuds. Pour initialiser cette matrice pour la premiere fois, les fourmis se déplacent aléatoirement et en parallèle par colonie et construisent leur premier tour. Chaque colonie possède sa propre matrice locale de communication. Une fois tous les points visités, les fourmis reviennent à leur point (ville) de départ. À la fin de cette première étape, les fourmis modifient la matrice globale de communication en mettant à jour ses entrées avec celle de la meilleure colonie trouvée (la meilleure solution à la fin d'une tournée).

Etape 2: Construction des solutions: Après la phase d'initialisation, à partie de cette séquence des villes déjà visitées, au sein des différentes colonies, les fourmis se déplacent cette fois sur les différents noeuds du graphe suivant une probabilité et donc l'équation est celle de la formule (3). Elle permet aux colonies de favoriser les plus courts chemins pendant les différentes itérations. Lors d'une tournée, chaque fourmi k d'une colonie L enregistre dans sa liste taboue (mémoire) la liste des villes déjà visitées.

La nouvelle formule probabiliste de sélection d'un noeud par la fourmi k de la colonie L est définie par l'expression de $p_{ij}^{k,l}$ (3). Cette probabilité est calquée sur celle utilisée dans l'heuristique OCF [2] décrite plus haut.

$$p_{ij}^{k,l}(t) = \begin{cases} \frac{[\eta_{ij}]^{\beta}[\tau_{ij}(t)]^{\alpha}}{\Sigma u \in v_{k_l}[\eta_{iu}]^{\beta}[\tau_{iu}(t)]^{\alpha}} & \text{Si j} \in v_{k_l} \\ 0 \quad \text{sinon} \end{cases} \quad (3)$$

[4]

Où v_{k_l} est l'ensemble des noeuds (villes) non visités par les fourmis de la colonie L.

Description de la matrice de phéromones: Il s'agit d'un tableau de i lignes et j colonnes. A $t = 0$ chaque entrée de la matrice est initialisée avec la valeur 0. $\tau_{ij}(t)$ représente l'intensité des phéromones sur l'arête (i, j) au cycle t. Pendant la construction d'un chemin, après chaque transition d'une fourmi k appartenant à une colonie L, les fourmis des différentes colonies utilisent directement la matrice globale de communication pour communiquer et procèdent immédiatement à une mise à jour de phéromones au fur et à mesure qu'elles évoluent. L'équation d'évolution la matrice de phéromone décrite dans l'équation (4) par :

$$\tau_{ij}(t+1) = \rho\ \tau_{ij}(t) + \triangle\ \tau_{ij} \quad (4)$$

[5]

L'équation (4) est la même que l'équation (2) dans OCF [2]. L'objectif principal visé par la une mise à jour après chaque transition est de simuler une sorte de parallélisme afin de gagner en temps de calcul en donnant l'information sur la matrice globale de communication en temps réel et non en différée. Comme dans OCF, pour réduire les échecs, il y à évaporation des phéromones[2]. Il s'agit de diminuer les phéromones (valeur) sur chaque chemin suivant la qualité de la solution. Ceci permet aux fourmis après un certain temps d'éviter d'emprunter certains chemins pouvant mener aux échecs. Cette caractéristique est modélisée via le paramètre ρ ($0 < \rho < 1$) . L'équation d'évaporation de la matrice de phéromone est décrite par la formule (5)

$$\tau_{ij}(t+1) = (1 - \rho)\ \tau_{ij}(t) \quad (5)$$

[6]

L'inverse de la distance entre les villes $\eta_{ij} = \frac{1}{C_{ij}}$ appelée visibilité est une information statique utilisée pour guider le choix des fourmis vers des villes proches et éviter les villes trop lointaines. Les paramètres α et β sont utilisés afin de déterminer l'importance relative de l'intensité de la trace et de la visibilité dans la construction d'une solution. Celle-ci est basée sur un compromis entre la visibilité (η_{ij}) et la quantité de phéromone (t_{ij}) présente entre i et j au cycle t. Ces paramètres sont les mêmes que ceux utilisés dans OCF[2]. À chaque colonie correspond une solution potentielle au problème, c'est-à-dire un tour complet. Lors de la récupération d'une solution, chaque fourmi des différentes colonies fournit une partie de la solution au problème. L'assemblage des différentes sous-solutions permet d'avoir la solution au problème original. Si à un instant t donné, toutes les fourmis d'une colonie ont la même solution, on arrête les itérations. La procédure finalement obtenue pour l'optimisation par multi colonie de fourmis est l'algorithme (5.2).

5.5. Complexité de l'algorithme MOCF

L'étude de la complexité de notre algorithme MOCF, se fait en subdivisant l'algorithme en 4 sections pour calculer la complexité.

- Initialisation: Etant donné que l'on a supposé une interconnexion totale entre les villes, nous avons une complexité de O (n^2 + m. L).

-Constructions du trajet, et dépôts progressif de phéromones plus évaluation des solutions: La complexité est O (n^2 . m. L)

Algorithme 5.2 : ALGORITHME MOCF

```
Début
    Entrée m : nombre de fourmis par colonies ;
    L : nombre de colonies ;
    n : nombre de villes ;
    N_c ← 0 ;
    D_ij ← 0 // matrice Globale ;
    d_ij ←t_0 // matrice local de phéromones ;
    Initialiser listeTaboue // avec la ville de départ de chaque fourmi ;
    // L'initialisation de la listeTaboue et de la matrice Dij ;
    Placer chaque colonie au hasard dans un point (ville) de départ ;
    // Construction parallèle des tours par les différentes colonies ;
    pour K ← 1 à m faire
        Construction d'une tournée par chaque fourmi de manière aléatoire ;
        Dépôt progressif des phéromones dans la matrice dij de chaque colonie ;
        Evaluation et choix de la meilleure colonie ;
        Initialisation de la matrice Dij avec celle de la meilleure colonie ;
    fin pour
    // Construction de la solution optimale ;
    Les colonies sont positionnées dans une ville de départ;
    Tantque ( N_c < N_max) Faire
        pour i ← 1 à n faire
            pour j ← 1 à L faire
                pour K ← 1 à m faire
                    Sélectionner la ville V_i à être ajoutée ou tour en cour selon la
                      formule (1);
                    Evaluer la solution de la fourmi K sur le trajet (i,j) ;
                    Effectuer la mise à jour globale de la trace selon la paire de
                      ville (i, j) si elle est meilleure que celle des fourmis
                      précédentes selon la formule 5.2 et 5.3(evaporation) ;
                    Insérer au fur et à mesure (i,j) dans liste tabou de façon à
                      construire progressivement la solution de K ;
                fin pour
            fin pour
        fin pour
    Fintantque
    S ← la meilleure solution : Chaque colonie fournit une solution partielle ;
Fin
```

- Evaporation des phéromones: La complexité est O (n^2)

- Evaluation des trajets après chaque transition: La complexité O (n.m.L). En effet pour les L colonies, on compare les tours de m fourmis de chaque colonies. Chaque tour a une longueur de n éléments.

En résumé, nous obtenons la complexité globale en additionnant la complexité de l'étape 1, au produit du nombre total de cycle (soit NCmax) par la complexité globale des étapes 2 à 4. Ce qui donne O (n^2 + m. L + NCmax . n^2 . m .L).
soit finalement **O(NCmax . n^2 . m .L)**.

6. Résultats expérimentaux - Discussions et interprétations

Afin de démontrer la performance de l'approche que nous proposons, nous effectuons des simulations informatiques sur une machine présentant les caractéristiques suivantes : Intel Dual core, processeur 2Ghz, RAM 2Go, Système d'exploitation Kali Lunix. Dans toutes les expériences de cette section, nous définissons comme paramètres de départ dans un premier temps $m = 100$ et $L = 10$, puis dans un second temps $m = 100$ et $L = 50$. Ces expériences sont menées sur les problèmes de référence Lin105, Pr124, LIN318, att532, ALi535, rat783, std1655, Vm1748, pr2392, Usa13509 et pla33810 tous issus de la librairie TSPLIB[5] comportant des instances de TSP accessibles au public.

Dans les mêmes conditions, nous les avons soumis à notre heuristique et nous avons comparé les résultats obtenus avec les meilleurs heuristiques de l'heure pour le TSP au moment des tests. Plusieurs simulations (100 au total) ont permis d'apprécier la pertinence et l'efficacité de notre approche. En faisant une étude comparative entre l'heuristique MOCF avec l'heuristique AS de base et ACS obtenu par Dorigo [**?**] ainsi que ACS-LK l'une des meilleures heuristiques pour le TSP de la littérature proposée dans [10] ou encore celles obtenues dans [4] (AG -LKH, ACS -LKH) et dans les conditions de test identique nous obtenons respectivement les résultats des figures 1, 2,3 et 4.

INSTANCES OF TSP	TAILLE	AS- D'origine		
		best soluiton	Temps	reussite/ 100
etl51	51	252,46	/	/
st70	70	675	/	/
LIN 105	105	ND	/	/
Pr124	124	ND	/	/
LIN318	318	ND	/	/
Attr532	532	ND	/	/
attr535	535	ND	/	/
Rat783	783	ND	/	/
std1655	1655	ND	/	/
Vm1748	1748	ND	/	/
pr2392	2392	ND	/	/
Usa13509	13509	ND	/	/
Pla33810	33810	ND	/	/

Figure 1. *Résultats AS depart*

INSTANCES OF TSP	TAILLE	ACS- départ		
		best solution	Temps	réussite/100
d198	198	585000	/	15
pcb442	442	595000	/	51
Attr532	532	830858	/	28
Rat783	783	991276	/	9
fl 1577	1577	942000	/	22

Figure 2. *Résultats ACS-DORIGO [2]*

Dans les résultats présentés par la figure 2, les lignes vertes représentent les cas où les gains observés avec MOCF sont en rapport avec les coûts et les temps d'exécution. Les lignes marron quant à elles symbolisent les cas où le gain observé se situe uniquement au niveau des temps d'exécution la solution optimale étant déjà atteinte. On note

INSTANCES OF TSP	TAILLE	MOCF			ACS_LK				
		best soluiton	Temps	reussite/100	Instances	TAILLE	best soluiton	Temps	Réussite /100
LIN 105	105	14379	0,03	100	Lin 105	105	14379	0,03	100
Pr124	124	58537	0,03	100	Pr124	124	58537	0,92	100
LIN318	318	42029	0,34	100	Lin318	318	42029	0,8	100
Attr532	532	276787,7	5,3	100	Att532	532	276787.7	13,26	100
attr535	535	202339	1,94	100	Ali535	535	202339	30,94	100
Rat783	783	88060	0,28	100	Rat783	783	88060	0,28	100
std1655	1655	6218,6	72,18	100	std1655	1655	62128.6	73,28	100
Vm1748	1748	336557	41,86	100	Vm1748	1748	336557	71,86	100
pr2392	2392	378032,2	0,97	100	Pr2392	2392	378034.25	1,19	90
Usa13509	13509	19849706	21103,3	100	Usa13509	13509	19984330	1864,74	84
Pla33810	33810	ID	/	./	Pla33810	33810	ID	/	/

Figure 3. *Comparaison MOCF avec l'algorithme hybride ACS-LK*

un temps d'exécution meilleur pour MOCF par rapport à ceux obtenus jusqu'ici avec les meilleurs algorithmes (ACS-LK, AG-LK, ACS-LKH, ACS) au moment des tests. On obtient également un coût meilleur et toujours optimal comparé aux meilleures solutions connues jusqu'ici avec des déviations allant de 0.1 à près de 1000 unités de distance sur certaines instances de taille raisonnable (Usa13506, Pr2392...). Les chiffres dans les parenthèses représentent les temps d'exécution. Les figures 5 et 6 présentent la comparaison de la performance de notre approche comparée à celles des approches améliorées AG - LKH et ACS -LKH, en utilisant les problèmes de référence sus-cités et terme de qualité de solution et en temps d'exécutions respectivement.

Les résultats de nos expériences montrent clairement la supériorité de notre algorithme sur les autres notamment en termes de qualité des solutions trouvées et de temps mis pour atteindre cette solution optimale. Cependant, pour certaines des instances dont on ne connait pas encore de solution, notre approche n'a pas pu donner une solution optimale.

INSTANCES OF TSP	TAILLE	ACS_LKH			AG_LKH			MOCF		
		best soluiton	Temps	reussite/100	best soluiton	Temps	reussite/100	best soluiton	Temps	reussite/100
LIN 105	105	14379	0,03	100	14379	0,03	100	14379	0,03	100
Pr124	124	58537	0,92	100	58537	0,46	100	58537	0,03	100
LIN318	318	42029	0,8	100	42097,4	1,6	100	42029	0,34	100
Attr532	532	276787,7	15,62	100	27691	13,74	100	276787,7	13,26	100
attr535	535	202339	30,94	100	202339	38,26	100	202339	1,94	100
Rat783	783	88060	0,28	100	88060	0,32	100	88060	0,28	100
std1655	1655	62128,6	73,28	100	63128	151,5	100	6218,6	72,18	100
Vm1748	1748	336557	71,86	100	336557	111,07	100	336557	41,86	100
pr2392	2392	378034,3	1,19	90	378034,8	1114,1	80	378032,2	0,97	100
Usa13509	13509	19984330	2113,3	100	ID	ID	ID	19849706	21103,3	100
Pla33810	33810	ID	/	/	ID	/	/	ID	/	./

Figure 4. *Comparaison MOCF avec les algorithmes hybrides ACS-LKH, AG-LKH*

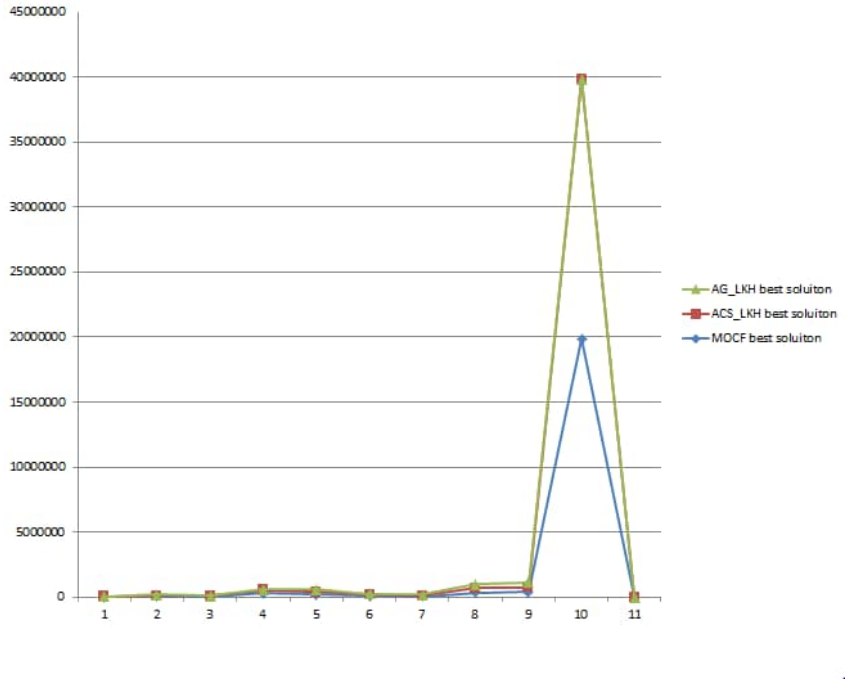

Figure 5. *Comparaison MOCF, AG-LKH, ACS-LKH en terme de qualité de solution*

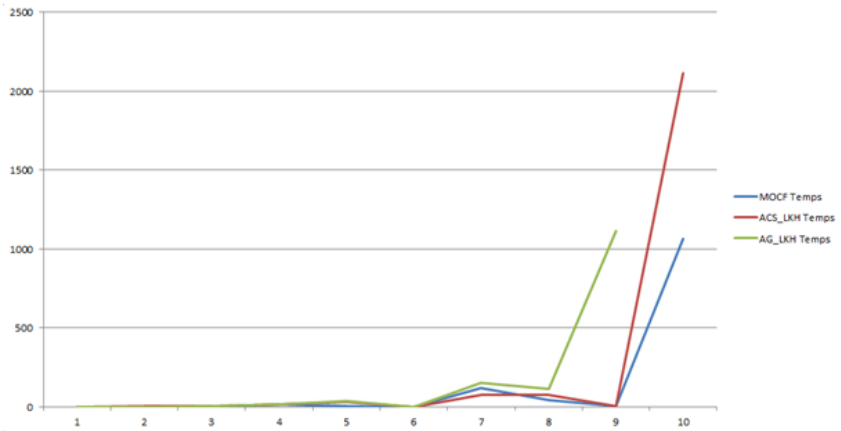

Figure 6. *Comparaison MOCF, ACS-LKH, AG-LKH en temps dexécutions*

7. Conclusion et perspectives

Dans ce travail, nous avons proposé une nouvelle heuristique multi-colonie de fourmis (MOCF) pour améliorer la recherche des solutions optimales au TSP et en temps optimal. L'algorithme MOCF proposé est bati sur l'utilisation de plusieurs colonies de fourmis. Il montre de nettes améliorations aussi bien en coûts qu'en temps d'exécutions, prouvant ainsi qu'il est bien plus compétitif que les meilleures heuristiques de lheure pour le TSP. Une limite liée à notre approche multi-colonies sera sans doute l'augmentation du temps d'exécution nécessaire pour trouver une solution dans certains cas, ou encore l'utilisation de beaucoup plus de ressources (mémoire et processeur). Nous pensons que la parallélisation de l'algorithme OCF [2] ou MOCF serait d'un apport considérable aux métaheuristiques pour réduire les temps de résolution du TSP.

8. Bibliographie

[1] C. REGO, D. GAMBOA, F. GLOVER, C. OSTERMAN, « Traveling salesman problem heuristics: Leading methods, implementations and latest advances », *European Journal of Operational Research* , vol. 211, pp 427-441, 2011.

[2] M. DORIGO, T. STUETZLE, « Ant Colony Optimization: Overview and Recent Advances », *IRIDIA Technical Report Series Technical Report* , vol. No.TR/IRIDIA/2009-013), n° 34 pages , 2009.

[3] K. HELSGAUN, « An effective implementation of the Lin-Kernighan traveling salesman heuristic », *AIEEE Transactions on Evolutionary Computation*, *Ochoa S.F., Roman GC. (eds)* vol. no 12, n° pp 106-130, 2000.

[4] B. NGUIMEYA TSOFACK, M. SOH, L.P. FOTSO, « Algorithmes hybrides pour la résolution du problème du voyageur de commerce », *Proceedings of CARI 2016*, *Ochoa S.F., Roman GC. (eds)* vol. , n° pp 63-74, 2016.

[5] TSPLIB95, « Traveling Salesman Problem Library », *http://www.iwr.uniheidelberg.de/iwr/comopt/software/TSPLIB95*, Last visit:December 23,2019.

[6] T. STUETZLE, « The Traveling Salesman Problem: State of the Art », *TUD SAP AG Worshop on Vehicle Routing*, *Ochoa S.F., Roman GC. (eds)* vol. 12, n° july 10, 2003.

[7] BP. DELISLE, « Parallélisation d'un algorithme d'optimisation par Colonies de Fourmis pour la résolution dun problème d'ordonnancement industriel », *IEEE*, vol. 12, n° pp 5366, 2002

[8] C. CHAUHAN , R. GUPTA , K. PATHAK, « Survey of Methods of Solving TSP along with its Implementation using Dynamic Programming Approach », *International Journal of Computer Applications (0975 8887)* vol. 52, n° 4, pp 12-19, 2012.

[9] I. ALAYA, « Optimisation multi-objectif par colonies de fourmis Cas des problèmes de sac à dos », *PhD Thesis, Université de la Manouba*, vol. n° , 2009.

[10] B. TADUNFOCK TETI, L.P. FOTSO, « Heuristiques du problème du voyageur de commerce », *Proceedings of CARI 2006*, *Fowler, D. and Dawson, L. (eds.)* vol. n° pp 1-8, 2006.