Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed El-Bachir El-Ibrahimi Bordj Bou Arréridj
Faculté des mathématiques et de l'informatique
Département de Recherche Opérationnelle

N° ordre.....

Mémoire de Master

Mémoire présenté en vue de l'obtention du diplôme de Master en :
Domaine : Mathématiques et Informatique.
Filière : Mathématiques.
Spécialité : Méthodes et outils pour la recherche opérationnelle.

Optimisation par Colonie de Fourmis Hybride (HACO) pour le Problème de Routage de Véhicule Multi-Dépot (MDVRP)

Présenté par :
- KHELIFA Wail
- HILAL Boubakeur

Sous la direction de :
M. Saha Adel

Soutenu publiquement le ...../...../..... devant le jury composé de :

.....	Université de BBA	Président
.....	Université de BBA	Examinateur
.....	Université de BBA	Examinateur
M. SAHA ADEL	Université de BBA	Encadreur

Année Universitaire

2022/2023

Remerciements

Je tiens tout d'abord à exprimer ma gratitude envers Allah le Tout-Puissant pour m'avoir accordé la patience et la force nécessaires pour réaliser ce travail.

Je souhaite remercier chaleureusement le M. SAHA Adel, Notre superviseur, d'avoir accepté de guider ce mémoire. Je suis extrêmement reconnaissant de l'aide qu'il m'a apportée sur différents aspects théoriques lors de nos nombreuses discussions, ce qui m'a permis de progresser méthodologiquement. Sa disponibilité, ses conseils précieux et sa patience tout au long de ce travail ont grandement contribué à un encadrement idéal. J'ai également beaucoup apprécié son soutien dans la rédaction de ce manuscrit et la préparation de ma soutenance.

Je suis reconnaissant(e) envers tous les membres du jury pour leur temps, leur expertise et leurs précieuses contributions à l'évaluation de ce mémoire, bien que je ne puisse pas les nommer individuellement.

Merci !

Dédicaces de Wail

$\mathcal{A}$ ma famille aimante,

Votre amour et votre soutien ont été essentiels tout au long de mon parcours. Merci d'avoir été mes piliers et de m'avoir encouragé à chaque étape de ce mémoire.

$\mathcal{A}$ mon binôme et mes amis,

Votre collaboration, vos rires et votre présence m'ont apporté une précieuse énergie et un soutien indéfectible. Je suis reconnaissant de vous avoir eus à mes côtés durant cette aventure académique.

$\mathcal{A}$ mes enseignants,

Votre expertise et votre passion pour l'enseignement ont été une source d'inspiration. Je vous remercie pour votre patience, vos conseils et votre contribution à ma formation.

$\mathcal{A}$ tous ceux qui ont contribué à ma réussite,

Que ce soit par des encouragements, des conseils ou des ressources, je suis reconnaissant pour votre soutien précieux.

Ce mémoire est dédié à chacun de vous. Votre impact sur ma vie et mon parcours est inestimable, et je suis honoré de partager cette réussite avec vous.

Merci du fond du cœur.

Dédicaces de Boubakeur

$\mathcal{A}$ ma très chère mère, qui me donne toujours l'espoir de vivre et qui n'a jamais cessé de prier pour moi.
$\mathcal{A}$ mon très cher père, pour ses encouragements, son soutien, et son sacrifice afin que rien n'entrave le déroulement de mes études.

Quoi que de plus que de pouvoir partager les meilleurs moments de sa vie avec les êtres qu'on aime.

Arrivé au terme de mes études, j'ai le grand plaisir de dédier aussi ce modeste travail :
$\mathcal{A}$ mes frères et sœurs,

$\mathcal{A}$ mes cousins et cousines, toute ma grande famille tous mes enseignants à la faculté MI et en général, tous ceux que j'aime et respecte.

Table des matières

Liste des figures

Liste des tableaux

Glossaire

— **ACO** Optimisation par colonies de fourmis.
— **AG** Algorithmes génétiques.
— **B&B** Branch et Bound.
— **CMDVRP** Problèmes de routage de véhicules multi-dépôts avec capacité.
— **CVRP** Problème de routage de véhicule avec capacité .
— **DFJ** Danzig, Fulkerson et Johnson.
— **HACO** Optimisation par colonie de fourmis hybride.
— **MDVRP** Problème du routage de véhicule multi-dépôts.
— **OC** Optimisation combinatoire.
— $\mathbf{P}_i$ Problème d'optimisation combinatoire num i, $\forall\ i = \{1,\ldots,5\}$, $i \epsilon \mathbb{N}$.
— **PO** Problème d'optimisation.
— **POC** Problème d'optimisation combinatoire.
— **PVRP** Problème de routage de véhicule périodique.
— **QAP** Problème d'affectation quadratique.
— **RS** La méthode de recuit simulé.
— **RT** la méthode de recherche tabou.
— **SMA** Les systèmes multi agents.
— **TSP** Problème du voyageur de commerce.
— **VRP** Probléme de routage de Véhicules.
— **VRPPD VRP** avec ramassage et livraisons.
— **VRPTW VRP** avec fenêtres temporelles.

Introduction générale

La recherche opérationnelle est un domaine qui vise à résoudre des problèmes d'optimisation dans divers domaines, tels que la logistique. L'optimisation combinatoire est une branche de la recherche opérationnelle qui se concentre sur la résolution de problèmes combinatoires, où une solution optimale doit être trouvée parmi un grand nombre de combinaisons possibles.

Le Problème de Tournées de Véhicules (**VRP**) est l'un des problèmes d'optimisation combinatoire les plus étudiés. Il consiste à déterminer la meilleure façon de livrer un ensemble de clients à partir de dépôts, en minimisant la distance totale parcourue par les véhicules. Avec la complexité croissante des problèmes réels, le Problème de Tournées de Véhicules Multi-Dépôts (**MDVRP**) est apparu, où plusieurs dépôts doivent être pris en compte.

Il est important de noter que le Problème de Tournées de Véhicules Multi-Dépôts est connu pour être NP-difficile, ce qui signifie qu'il est extrêmement difficile de trouver une solution optimale en un temps raisonnable. C'est pourquoi des approches heuristiques, telles que les métaheuristiques, sont couramment utilisées pour résoudre ces problèmes.

Les métaheuristiques sont des méthodes d'optimisation génériques qui guident la recherche d'une solution optimale en explorant l'espace de recherche de manière intelligente. Parmi les métaheuristiques, on trouve les méthodes approchées à trajectoire et les méthodes approchées à population. Les méthodes approchées à trajectoire, comme l'algorithme génétique, utilisent une seule solution à la fois pour explorer l'espace de recherche. Les méthodes approchées à population, telles que Les algorithmes de colonies de fourmis (Ant colony optimisation) **ACO**, utilisent plusieurs solutions simultanément.

Dans cette étude, nous nous concentrons sur l'**ACO**, une métaheuristique inspirée du comportement des fourmis cherchant de la nourriture. L'**ACO** a été adapté pour résoudre le Problème de Tournées de Véhicules Multi-Dépôts en utilisant des colonies de fourmis virtuelles pour trouver des solutions de haute qualité. Pour améliorer les performances de l'**ACO**, on a combiné la méthode de recherche locale **(2-opt)** avec **ACO** pour essayer d'améliorer la solution.

Le memoire est organisé comme suit :

— Le premier chapitre présente l'optimisation et l'optimisation combinatoire en général.
— Le deuxième chapitre se concentre sur le problème de Tournées de Véhicules **(VRP)** et son extension le problème de Tournées de Véhicules Multi-Dépôts **(MDVRP)**.
— Le troisième chapitre explore en détail l'**ACO**.
— Le quatrième chapitre présente l'approche hybride de l'**ACO** pour le **MDVRP**.
— Enfin, le cinquième chapitre décrit l'implémentation de l'algorithme hybride (**ACO**+(**2-opt)**) pour résoudre le problème **MDVRP** en utilisant Matlab, en prenant l'exemple de 58 wilayas (clients) tel que a chaque itération, l'algorithme sera appliqué pour obtenir des tournées jusqu'à ce que la solution optimale soit trouvée.

1 L'optimisation combinatoire

L'optimisation combinatoire est une discipline majeure en recherche opérationnelle, en mathématiques discrètes et en informatique, en raison de sa grande difficulté et de ses nombreuses applications pratiques. Bien que les problèmes d'optimisation combinatoire soient souvent aisés à formuler, ils sont généralement difficiles à résoudre. En effet, la plupart de ces problèmes appartiennent à la classe des problèmes NP-difficiles, qui ne disposent pas actuellement d'une solution algorithmique efficace pour toutes les données.

1.1 Généralités

1.1.1 L'optimisation

Un problème d'optimisation se définit comme la recherche du minimum ou du maximum d'une fonction donnée. D'une manière général, un probléme d'optimisation peut s'écrire comme suit [Dja15] :

$$(PO)\begin{cases} \text{Minimiser/Maximiser } f(x) \\ \text{S.c (souslescontraintes)} \\ g_i(x) \leq 0 \quad i = 1,...,m \\ h_j(x) = 0 \quad j = 1,...,k \\ x \in S \subset \mathbb{R}^n \end{cases} \tag{1.1}$$

Tel que :

$f : \mathcal{S} \to \mathbb{R}$: Fonction objectif.
$x \in \mathcal{S}$: Vecteur de variables de décision.
$\mathcal{S} \subset \mathbb{R}^n$: Espace de recherche.
n : La dimonsion du problème.
$g_i(x)$ / $h_j(x) : \mathcal{S} \to \mathbb{R}^n$, $i = 1,\ldots,m$ / $j = 1,\ldots,k$: Les contraintes du problème.

1.1.2 Classification des problèmes d'optimisation

Les problèmes d'optimisation peuvent être classés en trois grandes catégories [Dja15] :

1. Par rapport à la nature des variables
   Selon l'espace de recherche, on distingue :
   — L'optimisation continue : l'espace de recherche est continue.
   — L'optimisation discréte : l'espace de recherche est discret et infini.
   — L'optimisation combinatoire : c'est une optimisation discréte mais avec un espace de recherche finie (dénombrable).
2. Par rapport aux critéres
   — Mono-objectif : une seule fonction objectif à optimiser
   — Multi-objectif : plusieurs fonctions objectifs à la fois.
3. Par rapport aux contraintes
   — Avec contraintes : fonction(s) objectif(s) sous contraintes.
   — Sans contraintes : fonction(s) objectif(s) sans contraintes.

1.1.3 L'optimisation combinatoire

L'optimisation combinatoire (**OC**) occupe une place très importante en recherche opérationnelle et en informatique. De nombreuses applications pouvant être modélisées sous la forme d'un problème d'optimisation combinatoire (**POC**) telles que le problème du voyageur de commerce, Le problème d'affectation, le problème de la coloration de graphes, etc. (**POC**) comprend un ensemble fini de solutions, chaque solution doit respecter un ensemble de contraintes liées à la nature du problème, et une fonction objectif pour évaluer chaque solution trouvée. La solution optimale est celle dont la valeur de l'objectif est la plus petite (resp. grande) dans le cas de minimisation (resp. maximisation) parmi l'ensemble de solutions [Dab].

1.1.4 Exemples de problèmes combinaoire

1.1.4.1 Le problème du voyageur de commerce (TSP)

Le problème du voyageur de commerce (**TSP**) est l'un du problèmes combinatoires le plus connus. Il consiste à déterminer un tour qui commence et se terminer à un nœud (sommet) de base donné après avoir visité exactement un ensmeble de nœuds une fois et minimisant la distance total. Le (**TSP**)est divisé en deux catégories, symétrique et asymétrique, en fonction de la distance entre chaque deux nœuds. Donc le (**TSP**) consiste à déterminer un circuit à distance minimale passent par chaque sommet une fois et une seule, un tel ciruit est connu comme un tour ou circuit hamiltonien [BHZMAeh21].

Le modèle de base est le suivant :

$$(P_1)\begin{cases} \text{Minimiser } \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & (1.2) \\ \text{S.c (sous les contraintes)} & \\ \sum_{i=1}^{n} x_{ij} = 1 \quad \forall j = 1,...,n & (1.3) \\ \sum_{j=1}^{n} x_{ij} = 1 \quad \forall i = 1,...,n & (1.4) \\ x_{ij} \in \{0,1\} \quad \forall i,j = 1,...,n & (1.5) \end{cases}$$

Tel que :

— Dans le cas générale on a n villes.
— L'équation (1.2) est la fonction objectif qui minimise la distance totale et c_{ij} est la distance ou le poids de l'arc (i,j).
— Les équations (1.3) et (1.4) sont la contrainte d'affectation, qui garantit que chaque nœud est visité et quitté exactement une fois.
— L'équation (1.5) est exprimer une variable binaire x_{ij}, si x_{ij} indique 1 c'est–à-dire on visite la ville j depuis la ville i, ou 0 sinon.

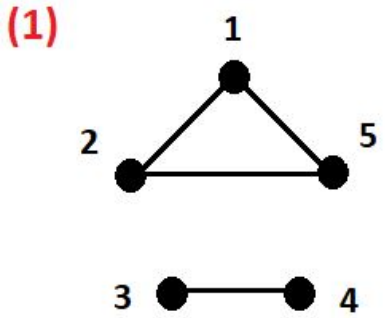

FIGURE 1.1 – Un problème TSP n'a pas de circuit Hamiltonien.

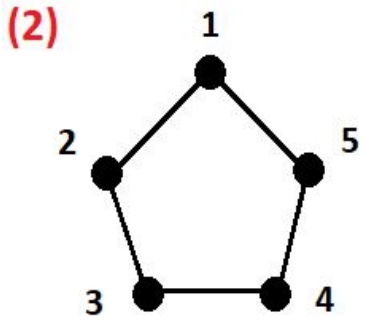

FIGURE 1.2 – Un problème TSP qui a un circuit Hamiltonien.

Définition 1.1.1. *Un graphe $G = (V,E)$ est dit hamiltonien s'il contient un cycle élémentaire passant par tous les sommets [Tou21].*

Définition 1.1.2. *Un cycle élémentaire d'un graphe $G = (V,E)$ est une chaîne élémentaire fermée, ou une séquence finie de sommets $\mathbf{x}_0\mathbf{x}_1\ldots\mathbf{x}_m$ telle que $\mathbf{x}i\mathbf{x}i+1 \in E(G)$, pour tout $0 \leqslant i < m$, où $\mathbf{x}_0$ et $\mathbf{x}_m$ coïncident. Cette chaîne ne passe pas par le même sommet deux fois (sans répétition de sommets) [Tou21].*

Remarque. *La figure (1.1) montre un problème de TSP avec une solution, mais on ne peut pas dire que cette solution est une solution pratique au problème, car cette solution n'a pas de circuit hamiltonien unique qui collecte tous les sommets à la fois !*

Problème. *Le modèle de base mentionné ci-dessus n'est pas complet car il ne prend pas en charge le circuit hamiltonien. Supposons qu'il y ait cinq nœuds et qu'un voyageur veuille tous les visiter, il peut le faire de deux manières illustrées aux figures (1.1) et (1.2).*

Solution. *On peut ajouter l'une des trois contraintes d'élimination de sous-tours. On va examiner la méthode de Danzig, Fulkerson, and Johnson appelée ■ The (DFJ) Formulation ■. L'équipe DFJ a proposé cette méthode depuis 1954, qui consiste à ajouter à* ($\mathbf{P}_1$) *une contrainte pour l'élimination de sous-tours et obtenir un circuit Hamiltonien unique [BHZMAeh21].*

$$\sum_{i\epsilon\mathcal{S},j\epsilon\mathcal{S},i\neq j} x_{ij} \leqslant |\mathcal{S}|-1 \qquad \forall\ \mathcal{S}\subset\{1,2,\ldots,n\},2\leqslant|\mathcal{S}|<n \qquad (1.6)$$

— *Dans chaque sous-ensemble* $\mathcal{S}$*, les sous-tours sont empêchés, garantissant que le nombre d'arcs sélectionnés dans* $\mathcal{S}$ *est inférieur au nombre de nœuds* $\mathcal{S}$ *(*$|\mathcal{S}|$*).*
— $\mathcal{S}$ *est un ensemble de sommets dont les cardinalités sont comprises entre* 2 *et* $n-1$*.*

Quand on a n *nœuds :*
— *On a* 2^n-n-2 *contraintes.*
— 2^n *façons de choisir un sous-ensemble.*
— n *façons de choisir un sous-ensemble ayant une seule*

1.1.4.2 Problème de coloration

Parmi les problèmes classiques d'optimisation combinatoire, on a un problème de coloration minimale, qui consiste à déterminer le nombre minimum de couleurs dans un graphe. On introduit une variable binaire x_{vk} à chaque noeud v et couleur k, valant 1 si le noeud v est coloré en utilisant la couleur k, et 0 sinon (l'équation (1.11)). On introduit aussi une variable binaire ω_k pour chaque couleur, valant 1 si la couleur k est utilisée, et 0 sinon (l'équation (1.12)). on cherche donc à minimiser la somme des ω_k (l'équation (1.7)), tout en s'assurant qu'un noeud est coloré d'une seule couleur (l'équation (1.8)), que sa couleur est accessible (en la comparant à ω_k) (l'équation (1.9)), et en vérifant que toute paire de noeuds reliés par un arête ne partagent pas la même couleur (l'équation (1.10)). Enfin, on dénotera par $\bar{\chi}$ l'ensemble des couleurs disponibles pour colorer les noeuds du graphe [SL07].

La formalisation mathématique du problème de coloration est comme suit :

$$(P_2)\begin{cases} \text{Minimiser } \sum\limits_{k\in\bar{\chi}} w_k & (1.7)\\ \text{S.c (sous les contraintes)} & \\ \sum\limits_{k\in\bar{\chi}} x_{vk}=1 \quad v\in V & (1.8)\\ x_{vk}\leq w_k \quad v\in V, k\in\bar{\chi} & (1.9)\\ x_{vk}+x_{uk}\leq 1 \quad (v,u)\in E, k\in\bar{\chi} & (1.10)\\ x_{vk}\in\{0,1\} \quad v\in V, k\in\bar{\chi} & (1.11)\\ w_k \in\{0,1\} \quad k\in\bar{\chi} & (1.12) \end{cases}$$

Définition 1.1.3. *Soit* $G=(V,E)$ *oú* V *représente l'ensemble des somets et* E *l'ensemble de ses aretes. La coloration des sommets d'un graphe* G *est une fonction* $C : v\rightarrow C(v)$ *associant à tout sommet* $v\epsilon V$ *une couleur* $C(v)$*, en s'assurant que* $C(v)\neq C(u)$ *pour tout arete* $[v,u]\epsilon E$ *[SL07].*

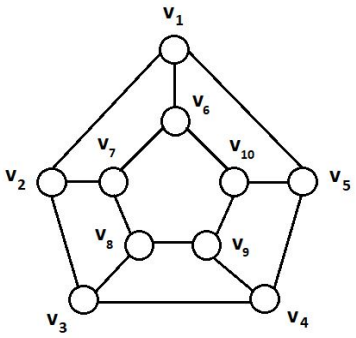

FIGURE 1.3 – Un graphe G.

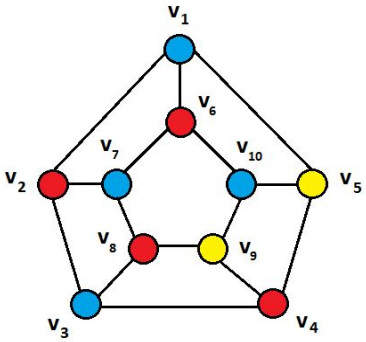

FIGURE 1.4 – Coloration le graphe G.

Définition 1.1.4. *On dit que G est k-coloration c'est-à-dire le nombre des couleurs utilisées égale a k [SL07].*

Définition 1.1.5. *Le nombre chromatique de G noté par $\chi(G)$ est définie comme étant la plus petit entier k tel que G admet une k-coloration [SL07].*

Remarque.

1. *Les figures (1.3) est (1.4) montrons un problème de coloration avec une solution faisable, la coloration d'un graphe est bien évidement pas unique, d'une part les couleurs peuvent etre changées (on peut remplacer la couleur rouge par noire), d'autre part la répartition des couleurs peut varier, ainsi que le nombre de couleurs utilisées.*
2. *La figure (1.4) montre un graphe coloré, le nombre chromatique de G noté par $\chi(G) = 3$.*
3. *Un graphe d'ordre n pourra toujours etre coloré de n couleurs différentes.*

1.1.4.3 Problème d'affectation

Le problème d'affectation est un problème combinatoire qui consiste à établir des liens entre les éléments de deux sous-ensembles distincts (**graphe biparti**), de manière à minimiser un coût et en respectant l'unicité des liens entre chaque couple d'éléments.

On considère un ensemble de tâches et un ensemble d'agents. Pour tout couple (i,j) tel que $(i,j = 1,\ldots,n)$, l'affectation de l'agent i à la tâche j entraine un coût de réalisation noté par C_{ij} (avec $C_{ij} \geqslant 0$). Chaque tâche doit être réaliser exactement une fois et chaque agent doit être réaliser une et une seul tâche. Donc le problème d'affectation consiste à affecter les agents aux tâches, et de minimiser le coût total de réalisation et recpecter les contraintes de réalisation.

Le problème d'affectation s'écrit sous la forme [For13] :

$$(\mathbf{P}_3)\begin{cases} \text{Minimiser } \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & (1.13)\\ \text{S.c (sous les contraintes)} & \\ \displaystyle\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall\ i = 1,\ldots,n & (1.14)\\ \displaystyle\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall\ j = 1,\ldots,n & (1.15)\\ x_{ij}\epsilon\{0,1\} \qquad \forall\ i,\ j = 1,\ldots,n & (1.16) \end{cases}$$

Tel que :
— L'équation (1.13) est la fonction objectif qui doit á minimiser.
— Les équations (1.14) et (1.15) sont la contrainte d'affectation.

$$x_{ij} = \begin{cases} 1 & \text{Si l'agent } i \text{ est affectée à la tâche } j. \\ 0 & \text{Sinon ;} \end{cases}$$

Définition 1.1.6. *On dit qu'un graphe $G = (V, E)$ est **biparti** si l'ensemble V des sommets peut être partitionnée en deux sous-ensembles V_1 et V_2 (c'est-à-dire $V_1 \cup V_2 = V$ et $V_1 \cap V_2 = \emptyset$) de telle sorte qu'un arc ne puisse relier entre eux deux sommets de V_1, ni relier entre eux deux sommets de V_2 [Tou21].*

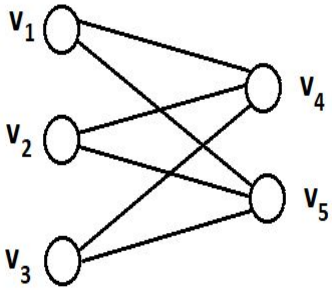

FIGURE 1.5 – Un graphe biparti.

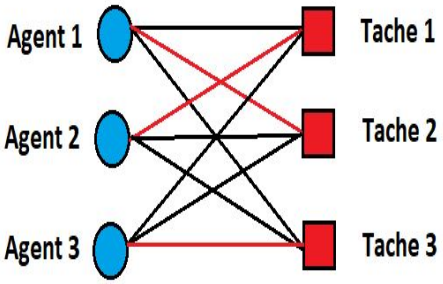

FIGURE 1.6 – L'affectation optimale d'un problème d'affectation.

Remarque.

1. *La figure (1.5) montre un graphe biparti d'après la définition (1.1.6) tel que :* $V_1 = \{v_1, v_2, v_3\}$ *et* $V_2 = \{v_4, v_5\}$.
2. *La figure (1.6) montre l'affectation optimale entre le group d'agents et de taches est représenté par les arcs rouges.*

1.1.4.4 Le problème de sac-à-dos

Le problème du sac-à-dos consiste à trouver le sous-ensemble d'objets à charger dans le sac de capacité W afin de maximiser la somme du profits. Considérons un ensemble de n objets, numérotés par l'indice i de 1 à n, possédant chacun un poids ω_i (weight) et un valeur ρ_i (profit), sans oublier qu'on ne peut pas dépasser la capacité totale du sac.

Mathématiquement, un problème du sac-àc-dos est représenté comme suite [Ham22] :

$$(P_4)\begin{cases} \text{Minimiser } \sum_{i=1}^{n} \rho_i x_i & (1.17) \\ \sum_{i=1}^{n} w_i x_i \leq W & (1.18) \\ x_i \in \{0,1\} \quad \forall i = 1,...,n & (1.19) \end{cases}$$

Tel que :
— Dans le cas générale on a n objets.
— L'équation (1.17) est la fonction objectif du problème sac-à-dos se définit comme la maximisation des profit des objets mis dans le sac, et ρ_i est le profit de l'objet i.

— L'équation (1.18) exprimer que ne doit pas dépasser la capacité du sac **W**, et ω_i est le poid de l'objet i.
— L'équation (1.19) est une contrainte d'intégrité $x_i \epsilon \{0,1\}$, si x_i indique 1 c'est–à-dire on choisi l'objet i, ou 0 sinon.

Remarque. *Le problème du sac-à-dos est un problème de sélectionner des objets, tel que :*

$$x_i = \begin{cases} 1 & \textit{Si l'objet } i \textit{ est choisi.} \\ 0 & \textit{Sinon ;} \end{cases}$$

1.2 Les méthodes de résolutions d'un POC

Les problèmes d'optimisation combinatoire sont souvent des problèmes très difficiles dont la résolution par des méthodes exactes peut s'avérer très longue ou peu réaliste. Trouver une solution optimale est un problème théoriquement facile : il suffit d'essayer toutes les solutions, et de comparer leurs qualités pour en dégager la meilleure. Cependant, en pratique, l'énumération de toutes les solutions peut prendre trop de temps ; or, le temps de recherche de la solution optimale est un facteur très important et c'est à cause de ce temps que les problèmes d'optimisation combinatoire sont réputés si difficiles.

Quelques problèmes d'optimisation combinatoire peuvent être résolus (de manière exacte) en temps polynomial par exemple par un algorithme glouton, un algorithme de programmation dynamique ou en montrant que le problème peut être formulé comme un problème d'optimisation linéaire en variables réelles. En pratique, la complexité physiquement acceptable n'est souvent que polynômiale. C'est pourquoi, parfois, on se contente alors d'avoir une solution approchée au mieux, obtenue par une heuristique ou une métaheuristique.

Une méthode approchée (heuristique ou métaheuristique) est un algorithme qui a pour but de trouver une solution réalisable, sans garantie d'optimalité, contrairement aux méthodes exactes qui garantissent des solutions exactes. Comme les algorithmes de résolution exacte sont de complexité exponentielle pour les problèmes difficiles, il peut être plus judicieux de faire appel aux méthodes approchées pour calculer une solution approchée d'un problème ou aussi pour accélérer le processus de résolution exacte. Généralement une heuristique est conçue pour un problème particulier, mais les métaheuristiques peuvent contenir des principes plus généraux[Dab].

1.2.1 Les méthodes de résolution exactes

Nous présentons d'abord quelques méthodes exacts, ces méthodes donnent une garantie de trouver la solution optimale.

1.2.1.1 La méthode séparation et évaluation (Branch and Bound)

L'algorithme de séparation et évaluation, plus connu sous son appellation anglaise **Branch and Bound** (**B&B**), repose sur une méthode arborescente de recherche d'un solution optimale par séparations du problème en sous-problèmes et évaluation, en représentant les états solutions par un arbre d'états, avec des nœuds, et des feuilles[Dab].

Le Branch-and-Bound est basé sur trois axes principaux :

— L'évaluation,
— La séparation,
— La stratégie de parcours.

1.2.1.2 Programmation dynamique

la programmation dynamique consistant à placer le problème dans une famille de problèmes de même nature mais de difficulté différente puis à trouver une relation de récurrence liant les solutions optimales de ces problèmes[Dab].

1.2.1.3 La méthode de la génération de colonnes

Le principe de la génération de colonnes repose sur le fait que toutes les variables d'un programme linéaire ne seront pas forcément utilisées pour atteindre la solution optimale. L'objectif de cette méthode est de résoudre un problème réduit avec un ensemble limité de variables. Le problème initial est appelé problème maître, et le problème réduit est appelé problème restreint. Le problème restreint est plus simple à résoudre, mais si l'ensemble de ses variables ne contient pas celles qui donne la solution optimale pour le problème maître, pour atteindre la solution optimale du problème maître, il faut rajouter au problème restreint des variables pouvant être utilisées pour améliorer la solution[Dab].

1.2.2 Les méthodes de résolution approchées

1.2.2.1 Les méthodes heuristiques

En optimisation combinatoire, une heuristique est un algorithme approché qui permet d'identifier en temps polynomial au moins une solution réalisable rapide, pas obligatoirement optimale. L'usage d'une heuristique est efficace pour calculer une solution approchée d'un problème et ainsi accélérer le processus de résolution exacte. Généralement une heuristique est conçue pour un problème particulier, en s'appuyant sur sa structure propre sans offrir aucune garantit quant à la qualité de la solution calculée. Les heuristiques peuvent être classées en deux catégories [For13] :

— **Méthodes constructives :** Elles génèrent des solutions à partir d'une solution initiale en ajoutant progressivement des éléments jusqu'à obtenir une solution complète.
— **Méthodes de fouilles locales :** Elles commencent avec une solution initialement complète (probablement moins intéressante) et, de manière répétitive, tentent d'améliorer cette solution en explorant son voisinage.

1.2.2.2 Les méthodes métaheuristiques

Une Métaheuristique est un algorithme d'optimisation visant à résoudre des problèmes d'optimisation difficiles, se veulent des méthodes génétiques pouvant optimiser une large gamme de problèmes différents ,sont souvent inspiré par des systèmes naturels (physique,biologie,...,etc).Les métaheuristique utilise deux approches principales pour résoudre un problème [For13] :

— **Approche de trajectoire :** Elle présente les algorithmes qui évaluent une fonction objectif unique à chaque itération. La stratégie est basée sur la recherche locale : le recuit simulé (RS), la recherche tabou (RT), etc.
— **Approche à population :** Elle désigne les algorithmes qui traitent plusieurs solutions à la fois. Ils maintiennent et améliorent plusieurs solutions candidates en même temps : les algorithmes génétiques (AG), l'algorithme de colonie de fourmis (ACO) qui va participer dans la résolution du problème.

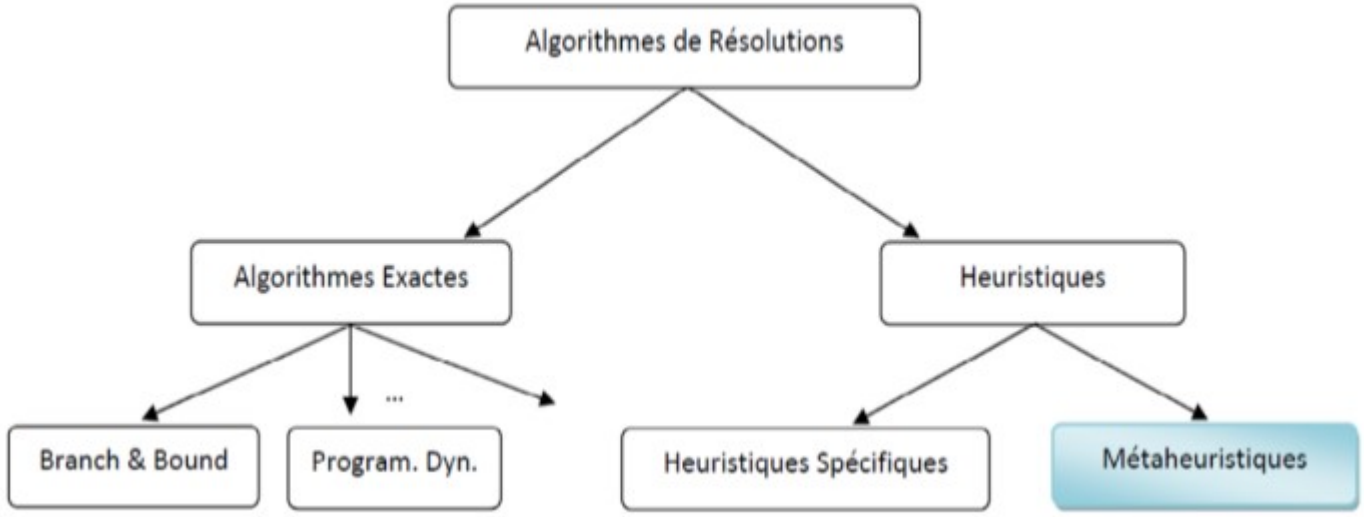

FIGURE 1.7 – Classes des méthodes de résolutions [ALI14].

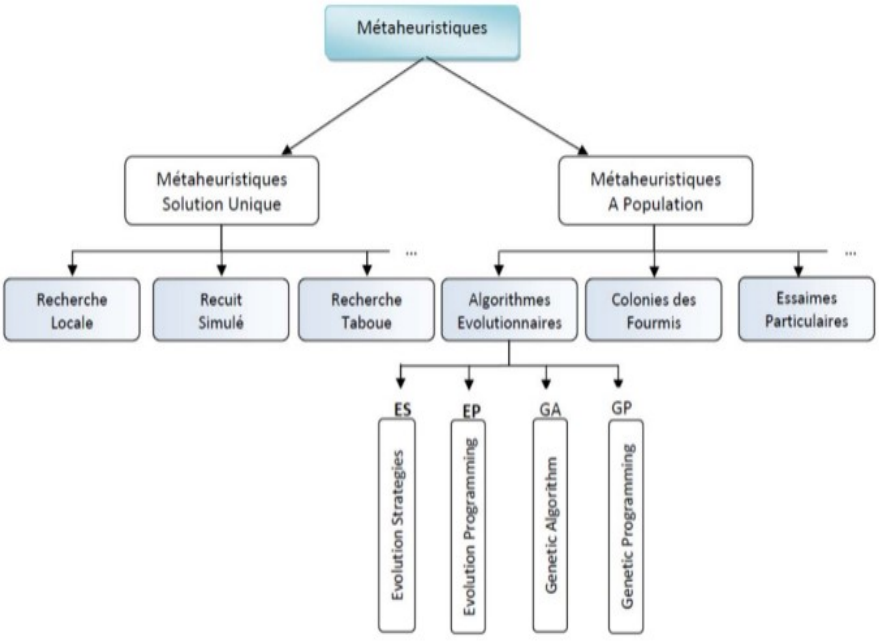

FIGURE 1.8 – Classes des méthaheuristiques [ALI14].

1.3 Conclusion

En conclusion, ce premier chapitre nous a permis de comprendre les fondements de l'optimisation, de classer les problèmes d'optimisation en fonction de leur nature, et d'explorer certains problèmes combinatoires, et de découvrir les méthodes de résolution exactes et approchées.

2

Les problèmes de tournées de véhicules et ses variantes

Le Probléme de Tournées de Véhicules (**VRP**, Vehicle Routing Problem) est l'un des problémes d'optimisation combinatoire les plus étudié. Il pose le problème suivant : visiter des clients à partir d'un dépôt et au moyen d'une flotte de véhicules, avec un coût optimal. De nombreuses variantes existent. Historiquement, le **VRP** est une version étendue du problème du Voyageur de Commerce (**TSP**, Traveling Salesman Problem), qui consiste à visiter l'ensemble des clients avec un seul véhicule [Mal06].

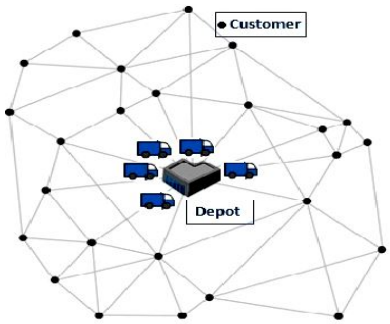

FIGURE 2.1 – Un probléme de tournées de véhicules [GHB15].

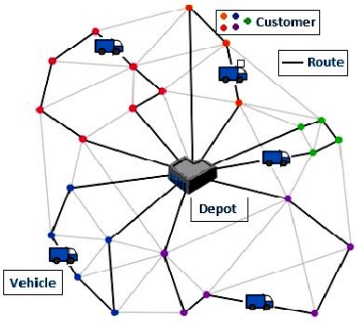

FIGURE 2.2 – la résolution du problème de tournées de véhicules [GHB15].

2.1 Le Probléme de Tournées de Véhicules VRP

2.1.1 Le modèle de base d'un VRP

Le **VRP** constitue une généralisation du **TSP** ($\mathbf{P}_1$) à plusieurs voyageurs. D'un point de vue terminologie, on parle de véhiculs. Un modèle a été donné par (Fisher & Jaikumar, 1978) et (Fisher & Jaikumar, 1981). Il s'agit d'une extension de la formulation du **TSP** vue précédemment. Pour les n

clients et K véhicules, on définit x_{ij}^k, variable binaire indiquant si le véhicule (la tournée) k effectue le trajet (i,j), et y_i^k, variable binaire indiquant si le véhicule k visite le client i [Mal06]. Voici le modèle étendu :

$$(\mathbf{P}_5)\begin{cases} \text{Minimiser} \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{K} c_{ij}x_{ij}^k & (2.1) \\ \text{S.c (sous les contraintes)} & \\ \sum_{k=1}^{K} y_i^k = 1 \qquad \forall\ i = 2,\dots,n & (2.2) \\ \sum_{k=1}^{K} y_1^k \leqslant K & (2.3) \\ \sum_{i=1}^{n} x_{ij}^k = y_i^k \qquad \forall\ j = 1,\dots,n\ ,\ \forall\ k = 1,\dots,K & (2.4) \\ \sum_{j=1}^{n} x_{ij}^k = y_j^k \qquad \forall\ i\ = 1,\dots,n\ ,\ \forall\ k = 1,\dots,K & (2.5) \\ \sum_{i\epsilon\mathcal{S}}\sum_{j\epsilon\mathcal{S}} x_{ij}^k \leqslant |\mathcal{S}|-1 \quad \forall\ k = 1,\dots,K, \mathcal{S}\subseteq\{2,\dots,n\}, |\mathcal{S}|\geqslant 2 & (2.6) \\ x_{ij}^k \epsilon\{0,1\} \qquad \forall\ i\ ,j = 1,\dots,n\ ,\ \forall\ k = 1,\dots,K & \\ y_i^k \epsilon\{0,1\} \qquad \forall\ i = 1,\dots,n\ ,\ \forall\ k = 1,\dots,K & \end{cases}$$

chaque client doit etre visité une fois, ce qui est assuré par la contrainte (2.2). La contrainte (2.3) garntit que chaque tournée passe par le dépôt. Les contraintes (2.4) et (2.5) sont le pendant pour le VRP de (1.2) et (1.3) (on arrive et on part de chez chaque client). Enfin, on retrouve les contraintes d'élimination de sous-tours on (2.6).

2.1.2 Le probléme de tournées de véhicules avec capacité CVRP

Il s'agit du même problème que le **VRP** ($\mathbf{P}_5$), à l'exception que chaque véhicule a maintenant une capacité Q et qu'on associe a chaque demande un poids q_i. Le modèle linéaire associé à ce problème est celui du VRP ($\mathbf{P}_5$) avec une contrainte supplémentaire qui assure le respect des contraintes de capacité [Mal06].

$$\sum_{i=1}^{n} q_i y_i^k \leqslant \text{Q} \quad \forall\ k = 1,\dots,K \qquad (2.7)$$

2.1.3 Problèmes de tournées de véhicules multi-dépôts MDVRP

Le problème de routage de véhicules multi-dépôts (**MDVRP**) est l'un des problèmes classiques de routage de véhicules (**VRP**), chaque dépôt avec leurs clients les plus proches. Le **MDVRP** peut être formalisé comme suit, on a un graphe $G = (V,E)$ tel que l'ensemble de sommets est divise à deux sous-ensembles comme suit $V = \{V_C\ ,\ V_D\}$ (c'est-à-dire, $V_C = \{v_1, v_2, \dots, v_n\}$ est l'ensemble des clients, et $V_D = \{v_{n+1}, v_{n+2}, \dots, v_{n+m}\}$ est l'ensemble des dépôts) et l'ensemble $E = \left\{(v_i, v_j) \text{ tel que } v_i, v_j \epsilon V, i < j\right\}$ est l'ensemble des arêtes. On a aussi la matrice de distance $C = (c_{ij})$ (avec $c_{ij} \geqslant 0$) est la distance

euclidienne entre chaque deux clients v_i et v_j. Chaque client v_i a une demande q_i et doit être visité une fois par un seul véhicule. Il existe aussi un flotte de K véhicules, chacun véhicule a une capacité Q. on définit x_{ij}^k, variable binaire indiquant si le véhicule (la tournée) k effectue le trajet (i,j), et y_i^k, variable binaire indiquant si le véhicule k visite le client i [XPD18]. La formulation mathématique du problème **MDVRP** est comme suit :

$$
(\mathbf{P}_6)\begin{cases}
\text{Minimiser} \displaystyle\sum_{i=1}^{n+m}\sum_{j=1}^{n+m}\sum_{k=1}^{K} c_{ij}x_{ij}^k & (2.8)\\
\text{S.c (sous les contraintes)} & \\
\displaystyle\sum_{i=1}^{n+m}\sum_{k=1}^{K} x_{ij}^k = 1 \qquad \forall\ j=1,\dots,n & (2.9)\\
\displaystyle\sum_{j=1}^{n+m}\sum_{k=1}^{K} x_{ij}^k = 1 \qquad \forall\ i=1,\dots,n & (2.10)\\
\displaystyle\sum_{i=1}^{n+m}\sum_{j=1}^{n+m} q_i x_{ij}^k \leqslant Q \quad \forall\ k=1,\dots,K & (2.11)\\
\displaystyle\sum_{i=n+1}^{n+m}\sum_{j=1}^{n} x_{ij}^k \leqslant 1 \qquad \forall\ k=1,\dots,K & (2.12)\\
\displaystyle\sum_{j=n+1}^{n+m}\sum_{i=1}^{n} x_{ij}^k \leqslant 1 \qquad \forall\ k=1,\dots,K & (2.13)\\
x_{ij}^k \epsilon \{0,1\} \qquad \forall\ i,j=1,\dots,n\ ,\ \forall\ k=1,\dots,K & \\
y_i^k \epsilon \{0,1\} \qquad \forall\ i=1,\dots,n\ ,\quad \forall\ k=1,\dots,K &
\end{cases}
$$

On a l'équation (2.8) est la fonction objective. Les contraintes (2.9) et (2.10) est de arrive et on part de chez chaque client par une seule véhicule. Chaque véhicule ayant une capacité on peut pas déppaser dans certain touenée par la contrainte (2.11). Enfin les contraintes (2.12) et (2.13) vérifier la disponibilité des véhicules.

2.2 Exemple illustratif

Une entreprise de livraison dispose de N dépôts et doit livrer des colis du dépôts á d'autres villes, tous les camions ont la même capacité (**MDVRP** homogène) et chaque ville á une commande (sauf les villes dépôts).

Pour résoudre ce problème de routage des véhicules multi dépôts avec capacité (**MDVRP** homogène), on doit passer par trois étapes :

1. **La classification :** En utilisant l'algorithme de regroupement par distance la plus proche (Nearest distance cluster algorithme) on va affecter chaque ville au dépôt le plus proche.

   Grace a cette classification on va passer du problème de routage des véhicules multi dépôts avec capacité à N problèmes de routage des véhicules mono dépôt avec capacité, et on va passer a l'étape suivante.

2. **Le regroupement :** On regroupe les villes dont ses commandes vont ensemble dans le même véhicule, on fait ce regroupement par rapport à la capacité de véhicule, par exemple si on reçoit 50 commandes de la part de la ville A et 30 commandes de la ville B, sachant que la capacité du véhicule est 80, A et B entre dans la même tournée et on peut pas ajouter une autre commande dans cette tournée (car la capacité de ce vehicule est saturé).

   Après ce regroupement, chaque groupe représentera une tournée par un seul véhicule, et chaque ville sera visité exactement une seule fois, pour chaque tournée, le départ se fait depuis le dépôt et l'arrivée s'effectue également au dépôt

3. **L'ordre de visite :** Dans les groupes choisis lors de l'étape précédente, lorsque nous commencerons notre tournée, nous commencerons forcément par le dépôt. Ensuite, nous choisirons l'ordre de visite des villes de manière à minimiser la distance parcourue.

2.3 Certaines variantes de VRP

Il existe de nombreuses variantes de **VRP** [Kaj20][AFG14][AF07], tel que :

1. **« VRPMO/VRP with multiple objectives » VRP** avec objectifs multiples : Les véhicules doivent minimiser le coût total, tout en satisfaisant d'autres objectifs, tels que minimiser le temps de trajet ou maximiser la satisfaction des clients.
2. **« VRPH/VRP with heterogeneous vehicles » VRP** avec véhicules hétérogènes : Les véhicules ont des capacités et des vitesses différentes.
3. **« VRPHO/VRP with homogeneous vehicles » VRP** avec véhicules homogènes : Les véhicules ont les mêmes capacités et les mêmes vitesses.
4. **« CVRP/Capacitated VRP » VRP** avec capacité : Chaque véhicule doit avoir une capacité qui ne peut être dépassée.
5. **« VRPTW/VRP with Time Windows » VRP** avec fenêtres temporelles : dans un certain délai, chaque client doit être livré. Chaque itinéraire de véhicule doit être commencé et terminé dans la fenêtre horaire associée au dépôt.
6. **« VRPPD/VRP with Pickup and Deliveries » VRP** avec ramassage et livraisons : les clients peuvent retourner certaines marchandises à l'entreprise.
7. **« PVRP/Periodic VRP » VRP** périodique : dans **VRP**, la période de planification est généralement d'une seule journée. Dans le cas du **PVRP**, le **VRP** classique est généralisé en étendant la période de planification à M jours.
8. **« VRPB/VRP with backhauls » VRP** avec retour à vide : Les véhicules doivent visiter des clients pour récupérer des marchandises, puis livrer ces marchandises à d'autres clients.
9. **« VRPSD/VRP with split deliveries » VRP** avec livraisons fractionnées : Les véhicules peuvent diviser leur charge et livrer des marchandises à plusieurs clients dans un seul itinéraire.
10. **« MDVRP/Multiple Depot VRP » VRP** à plusieurs dépôts : Le concessionnaire de la société de logistique peut utiliser plusieurs dépôts pour fournir les produits aux clients.

2.4 Conclusion

Au cours de ce chapitre, nous avons examiné les différentes variantes du problème de routage des véhicules, notamment le **CVRP**, le **MDVRV** et le **CMDVRP**. Nous en concluons que le **VRP** peut être assorti de différentes variantes et restrictions, lesquelles peuvent être représentées par une nouvelle formulation que nous ajoutons au modèle. Cette formulation doit être vérifiée à chaque recherche ou génération de solution réalisable (tournée) dans le **VRP**.

Nous avons également constaté que l'objectif principal de ce problème est d'optimiser la distance dans les itinéraires en établissant le coût C_{ij}. Nous avons vu le **MDVRP**- problème de routage des véhicules multi-dépôts en tenant compte la capacité, avec une description et une formulation mathématique.

Dans le chapitre suivant, nous allons parler de l'algorithme d'optimisation de colonie de fourmis (**ACO**) qui est l'un des algorithmes qui sera utilisé pour la résolution du **MDVRP**.

3

L'Optimisation par Colonie de Fourmis Hybride (HACO) pour le problème de routage des véhicules multi-dépôts (MDVRP)

Au cours des dernières années, les méta-heuristiques ont attiré l'attention des chercheurs pour leur efficacité dans la résolution de problèmes complexes. Ces techniques permettent de trouver des solutions de haute qualité pour des problèmes qui sont souvent difficiles à résoudre avec des méthodes traditionnelles.

Parmi les méta-heuristiques les plus prometteuses figurent les algorithmes inspirés de la nature, qui s'inspirent des comportements observés dans les systèmes biologiques pour concevoir des algorithmes de recherche efficaces.

Les insectes sociaux, tels que les fourmis, sont des exemples de systèmes biologiques qui ont inspiré de nombreux algorithmes de recherche en intelligence artificielle. Les fourmis sont réputées pour leur capacité à accomplir des tâches collectives relativement complexes, telles que la construction de nids, la recherche de nourriture ou la répartition des tâches au sein de leur colonie. Cette capacité collective est nommée intelligence collective et a inspiré de nombreux algorithmes de recherche en intelligence artificielle.

L'algorithme de colonie de fourmis (**ACO**) a été largement étudié et appliqué avec succès à divers problèmes d'optimisation combinatoire et de recherche opérationnelle, tel que le problème du voyageur comemrce, le problème d'affectation, le problème de coloration des graphes, le problème de routage de véhicules ... etc.

Dans ce chapitre, nous allons explorer les méta-heuristiques, en mettant l'accent sur les algorithmes inspirés de la nature, et en particulier l'algorithme de colonie de fourmis pour résoudre des problèmes complexes.

3.1 Les fourmis réelles

Dans un premier temps, les fourmis marchent aléatoirement de nid à la source de nourriture et Vice-versa. Chaque fourmi dépose sur son chemin une substance chimique appelée phéromone qui la guide au retour.

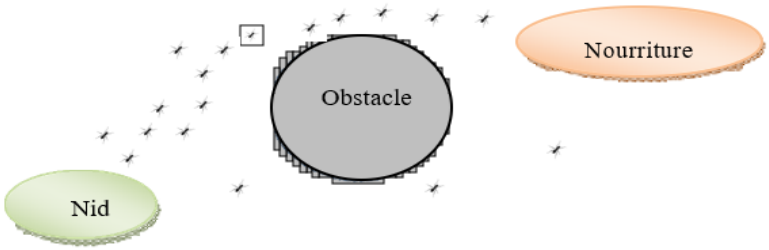

FIGURE 3.1 – La colonie choisit le chemin le plus court

Dans ce dernier cas, la nature de phéromone est différente puisqu'elle contient une information sur la qualité de la source de nourriture. Par la suite, la phéromone incite les autres fourmis de choisir ce chemin. Les fourmis ont aptitude de suivre le chemin qui contient la plus grande quantité de phéromones. Autrement dit, lorsqu'une fourmi se trouve devant plusieurs chemins, elle peut trouver le chemin le plus court grâce à la densité de phéromone. Ce comportement permet à la colonie de trouver le plus court chemin entre le nid et la source de nourriture (figure 3.1) [CDM91].

3.1.1 La résolution de problèmes complexes

La coopération entre les membres de la fourmilière permet la réalisation d'activités complexes, c'est ce que nous allons expliquer à l'aide de l'exemple du double pont binaire (figure 3.2) [DAGP90].

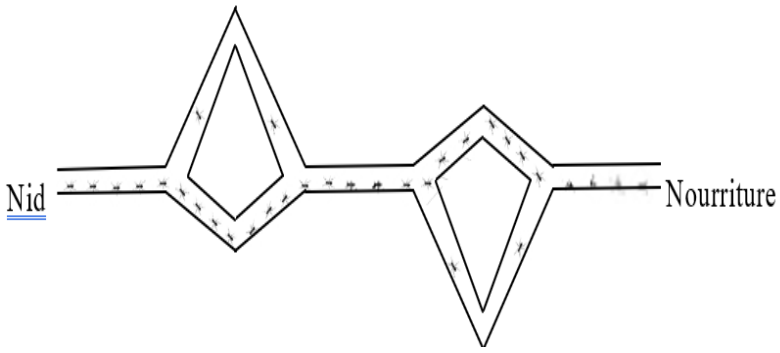

FIGURE 3.2 – L'expérience du double pont

Au départ, les ponts sont dépourvus de toute phéromone et sont donc tous équivalents en termes de probabilité d'être choisis. Les fourmis sélectionnent un chemin au hasard parmi les options disponibles.

Le chemin le plus court est rapidement marqué par une plus grande quantité de phéromone, car les fourmis empruntant ce chemin déposent plus de phéromone pendant la même période que celles empruntant d'autres chemins.

3.1.2 Stigmergie

La stigmergie est une forme de communication indirecte qui opère dans un environnement émergent auto-organisé, où les individus modifient leur environnement pour communiquer entre eux. Ce mode de communication indirecte conduit à l'apparition de comportements complexes de l'ensemble à partir de comportements individuels simples.

3.2 Les fourmis artificielles

En s'appuyant sur le comportement social des fourmis présenté dans la section précédente, des algorithmes d'optimisation appelés **ACO** (Optimisation de Colonie de Fourmis) ont été développés. Ces algorithmes sont à la fois simples et efficaces. Chaque fourmi dans ces algorithmes possède un ensemble de fonctions qui définissent son comportement, qui est très similaire à celui des fourmis réelles lorsqu'elles cherchent de la nourriture (comme illustré dans la figure 3.2). Les fourmis artificielles se déplacent dans un environnement qui est un espace combinatoire composé d'un ensemble d'objets obtenus par une modélisation du système étudié. Les mouvements de chaque fourmi représentent une solution au problème posé, et ces solutions sont évaluées à l'aide d'une fonction qui détermine la meilleure solution possible [OB07].

3.3 L'intelligence collective des fourmis

Les fourmis sont présentes dans la quasi-totalité des écosystèmes terrestres. Il existe plusieurs races différentes de fourmis qui présentent une grande diversité de comportements et de morphologies. Pour des raisons pratiques, les recherches du domaine d'intelligence collective de fourmis s'intéressent uniquement à un nombre limité de races.

Le comportement collectif des fourmis constitue une source d'inspiration pour les chercheurs. La coopération, la communication, la compétition et l'apprentissage de fourmis ont permis la conception de plusieurs algorithmes pour la résolution de divers problèmes [OB07]. Dans ce que suit, nous allons présenter les principales caractéristiques des fourmis que l'on pourra modéliser et l'exploiter dans des systèmes informatiques.

3.3.1 Le partage de tâches

L'une des caractéristiques les plus étonnantes est l'aptitude de la colonie de fourmis à se partager le travail. Les fourmis assurent d'une façon parallèle un ensemble de tâches importantes pour la survie et le développement de la colonie. On peut citer entre autres : la recherche de nourriture, la défense du nid, l'entretien et la construction du nid et l'entretien des larves et leur approvisionnement en nourriture [Mon99].

3.3.2 Auto-organisation

Un autre comportement observé chez les fourmis est l'auto-organisation. Le résultat de ce comportement est la construction de structures vivantes ayant différentes fonctionnalités. Ces structures sont composées de fourmis qui sont accrochées les unes aux autres. L'auto-assemblage représente a représenté une source riche d'inspiration pour plusieurs chercheurs surtout dans le domaine des algorithmes de classification non supervisée [AL04].

3.3.3 La communication

La fourmi possède plusieurs moyens de communication. On peut citer entre autres [Fer95], La communication sonore, tactile, visuelle, chimique.

3.4 Algorithme de colonie de fourmis

3.4.1 Principe de base

L'algorithme de colonie de fourmis (**ACO**) est une méthode d'optimisation inspirée du comportement des fourmis dans la recherche de nourriture. Il a été introduit pour la première fois par Marco Dorigo et ses collègues dans les années 1990. L'**ACO** est basé sur la communication indirecte entre les fourmis via les phéromones, qui sont des substances chimiques déposées sur le sol pour indiquer le chemin vers une source de nourriture. Les fourmis utilisent ces phéromones pour trouver les chemins les plus courts et les plus efficaces vers la nourriture. L'un des principaux avantages de l'**ACO** est sa flexibilité et sa capacité à s'adapter à différents types de problèmes.

De plus, l'**ACO** est une méthode d'optimisation stochastique, ce qui signifie qu'il est moins susceptible de se coincer dans des solutions locales non optimales par rapport aux méthodes d'optimisation déterministes. Enfin, l'**ACO** est facilement parallélisable, ce qui permet d'exploiter la puissance des architectures de calcul modernes pour résoudre des problèmes encore plus rapidement [DC99].

Le psudo-code de l'algorithme de la colonie de fourmis est présenté suivant [DS04] :

Algorithme Psudo-code de la méthode de colonie de fourmis

1. Initialiser les traces, et les paramètres
2. Tant qu'un critère d'arret n'est pas satisfait, répéter en parallèle pour chacune des k fourmis :
   — Construire une nouvelle solution à l'aide des informations contenues dans les traces et une fonction d'évaluation partielle (i.e heurstique).
   — Evaluer la qualité de la solution.
   — Mettre à jour les traces.

3.4.2 Formulation de base de l'ACO

Chaque fourmi doit construire une solution pour se déplacer à travers le graphe. Pour sélectionner la prochaine arête dans son parcours, une fourmi prendra en compte la longueur de chaque arête disponible depuis sa position actuelle, ainsi que le niveau de phéromone correspondant.

Àchaque étape de l'algorithme, chaque fourmi passe d'un état i à un état j, correspondant à une solution intermédiaire plus complète. Ainsi, chaque fourmi k calcule un ensemble $\mathcal{A}_k(i)$ d'extensions réalisables à son état actuel à chaque itération et se déplace vers l'une d'entre elles en fonction de la probabilité.

Pour la fourmi k , la probabilité P_{ij}^k de passer de l'état i à l'état j dépend de la combinaison de deux valeurs, l'attractivité η_{ij} du mouvement, calculée par une heuristique indiquant la désirabilité a priori de ce mouvement, et le niveau de piste τ_{ij} du mouvement, indiquant à quel point il a été efficace dans le passé pour effectuer ce mouvement particulier. Le niveau de piste représente une indication a posteriori de la désirabilité de ce mouvement.[ant]

En général, la fourmi k se déplace de l'état i à l'état j avec une probabilité :

$$P_{ij}^k = \frac{(\tau_{ij})^\alpha (\eta_{ij})^\beta}{\sum_{z \epsilon allowed_i} (\tau_{iz})^\alpha (\eta_{iz})^\beta} \tag{3.1}$$

Où τ_{ij} est la quantité de phéromone déposée pour la transition de l'état i à j, $0 \leqslant \alpha$ est un paramètre pour contrôler l'influence de τ_{ij}, η_{ij} est la désirabilité de la transition d'état ij (connaissance a priori, généralement $1/d_{ij}$ où d est la distance entre l'état i et j), et $\beta \geqslant 1$ est un paramètre pour contrôler l'influence de η_{ij}.

τ_{ij} et η_{ij} représentent le niveau de piste et l'attractivité pour les autres transitions d'état possibles.

3.4.3 Mise à jour des phéromones

Les pistes sont généralement mises à jour lorsque toutes les fourmis ont terminé leur solution, augmentant ou diminuant le niveau des pistes correspondant aux mouvements faisant partie de solutions « bonnes » ou « mauvaises », respectivement.

Un exemple de règle de mise à jour globale des phéromones est :

$$\tau_{ij}^{new} \leftarrow (1-\rho)\tau_{ij}^{\mathbf{ol}d} + \sum_{k}^{m} \Delta\tau_{ij}^k \tag{3.2}$$

Où est la quantité de phéromone déposée pour une transition d'état ij, ρ est le coefficient d'évaporation des phéromones, m est le nombre de fourmis et $\Delta\tau_{ij}^k$ est la quantité de phéromone déposée par la fourmi k, généralement donnée pour le problème **TSP** (avec des mouvements correspondant aux arcs du graphe) par :

$$\Delta\tau_{ij}^k = \begin{cases} \dfrac{Q}{l_k} & \text{Si la fourmi } k \text{ utilisée l'arrete } ij \text{ sur le tour.} \\ 0 & \text{Sinon ;} \end{cases}$$

Où l_k est le coût du parcours de la fourmi k (généralement la longueur) et Q est une constante.[ant]

3.5 Problèmes résolus avec ACO

l'algorithme de colonie de fourmis peut etre utilisée pour résoudre une grande variété de problèmes d'optimisation combinatoire, tel que [DMC96] :

1. Problème du voyageur de commerce (**TSP**) : L'**ACO** est largement utilisé pour résoudre le **TSP**, qui consiste à trouver le chemin le plus court pour visiter un ensemble de villes et revenir à la ville d'origine.
2. Problème de routage de véhicules (**VRP**) : L'**ACO** peut être appliqué au **VRP**, qui consiste à déterminer les itinéraires optimaux pour un ensemble de véhicules devant livrer des marchandises à un ensemble de clients.
3. Problème de coloration de graphe : L'**ACO** peut être appliqué au problème de coloration de graphe, qui consiste à attribuer des couleurs aux sommets d'un graphe de manière à ce que les sommets adjacents aient des couleurs différentes.
4. Problème d'affectation quadratique (**QAP**) : L'**ACO** peut être utilisé pour résoudre le **QAP**, qui consiste à attribuer un ensemble d'installations à un ensemble d'emplacements de manière à minimiser la somme des coûts associés.
5. Problème d'ordonnancement : L'**ACO** peut être utilisé pour résoudre des problèmes d'ordonnancement, tels que l'ordonnancement des tâches sur des machines parallèles pour minimiser le temps d'achèvement total.

— Ces exemples montrent que l'algorithme **ACO** est une méthode d'optimisation puissante et polyvalente qui peut être appliquée à divers problèmes de recherche de chemin et d'optimisation combinatoire.

3.6 L'Optimisation par Colonie de Fourmis Hybride (ACO+(2-opt))

L'algorithme **HACO** (**ACO+(2-opt)**) est pour but de résoudre le problème de routage de véhicules multi-dépôts (**MDVRP**). En combinant la méthode **ACO** avec des techniques de recherche locale, **HACO** (**ACO+(2-opt)**) peut générer un ensemble de solutions initiales qui sont ensuite améliorées par des opérateurs de recherche locale. Les résultats montrent que **HACO** (**ACO+(2-opt)**) peut produire des solutions significativement meilleures que celles générées par l'**ACO** seul.

3.6.1 L'Optimisation par Colonie de Fourmis Hybride

(HACO)

L'Optimisation par Colonie de Fourmi hybride (**HACO**) est un algorithme méta-heuristique récent pour résoudre des problèmes d'optimisation combinatoire difficiles, Il combine l'Optimisation par Colonie de Fourmis (**ACO**) avec d'autres algorithmes pour améliorer ses performances. Dans notre cas on a ajouté un algorithme de regroupement par distance la plus proche « **Nearest Distance Cluster Algorithm** » et une technique d'optimisation locale « **2-opt** » [HGH99].

3.6.2 L'algorithme de regroupement par distance la plus proche

L'algorithme de regroupement par distance la plus proche est une méthode utilisée dans l'algorithme d'Optimisation par Colonie de Fourmis Hybride (**HACO**) pour résoudre le problème de routage de véhicules (**MDVRP**). Dans cette méthode, les clients sont regroupés en groupes en fonction de leur proximité aux dépôts. Ensuite, chaque groupe est traité comme une tournée de véhicules distincte (**VRP**) [NAK18].

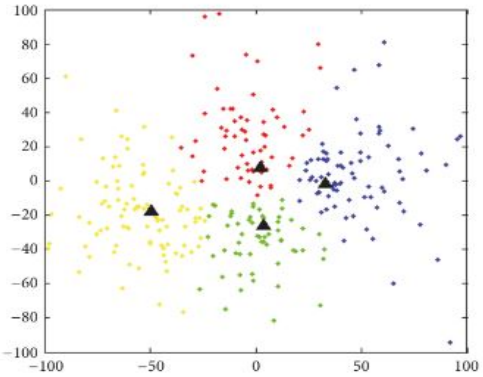

FIGURE 3.3 – Exemple d'application de L'algorithme de regroupement par distance la plus proche [XPD18].

3.6.3 Opération d'échange local 2-opt

La mèthode **2-opt** est une méthode d'optimisation utilisée dans l'algorithme d'Optimisation par Colonie de Fourmis Hybride (**HACO**) pour résoudre le problème de routage de véhicules (**MDVRP**). Cette méthode consiste à échanger deux arêtes dans une tournée de véhicules pour créer une nouvelle tournée. Si la nouvelle tournée est meilleure que l'ancienne, elle est conservée. Sinon, l'ancienne tournée est conservée. Cette méthode est répétée jusqu'à ce qu'aucune amélioration ne soit possible.

Le 2-opt est une méthode simple mais efficace pour améliorer les solutions générées par l'algorithme **ACO** dans le cadre du **MDVRP**. En utilisant cette méthode, les solutions peuvent être améliorées de manière significative sans augmenter le temps de calcul [HGH99].

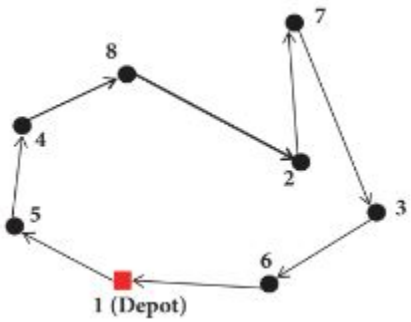

FIGURE 3.4 – Tournée avant l'opération de 2-opt [XPD18].

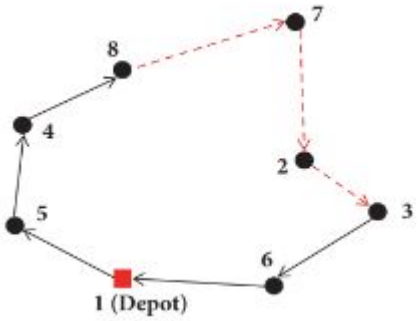

FIGURE 3.5 – Tournée aprés l'opération de 2-opt [XPD18].

3.7 Conclusion

En somme, l'algorithme de colonie de fourmis est un outil précieux dans le domaine de l'optimisation et de la recherche opérationnelle. Les recherches futures pourraient se concentrer sur l'amélioration de l'efficacité de l'**ACO**, l'adaptation de l'algorithme à de nouveaux problèmes et l'exploration de synergies avec d'autres méthodes d'optimisation.

L'**ACO** se distingue par sa capacité à explorer l'espace des solutions de manière efficace et à converger vers des solutions de haute qualité. Grâce à sa nature décentralisée et à sa mémoire collective sous forme de phéromones, l'**ACO** est capable de s'adapter aux changements dynamiques dans les problèmes et d'éviter les optima locaux. De plus, l'**ACO** peut être combiné avec d'autres techniques d'optimisation pour améliorer encore ses performances, cette combinaison nous donne **HACO** (Hybrid Ant Colony Optimization)(**ACO**+(**2-opt)**).

4 Application de HACO (ACO + 2-opt) sur MDVRP

Ce chapitre se concentre sur l'implémentation de l'algorithme d'optimisation par colonie de fourmis hybride (**ACO** combiné avec **2-opt**) pour résoudre le problème du routage de véhicule multi-dépôt (**MDVRP**) statique avec véhicule homogène. Pour illustrer l'efficacité de cette approche, nous avons créé un modèle nommé **''clients.58"** qu'on va par la suite l'expliquer.

L'outil utilisé pour tester cette approche est Matlab R2016a. Nous avons choisi Matlab en raison de ses capacités de programmation et sa performance dans les matrices, Pour exécuter Matlab, nous avons utilisé un ordinateur Lenovo équipé d'un processeur Intel Core i5 de 7e génération, 8 Go de mémoire RAM et un disque SSD de 256 Go.

Dans ce chapitre, nous commençons par décrire notre modèle ''clients.58". Ensuite, nous discutons les résultats obtenus à partir de nos expériences et analysons les performances de cette approche dans la résolution du problème.

4.1 L'implémentation de HACO sur MDVRP

4.1.1 Description du problème

Notre problème, nommé « clients.58 », est un problème du routage de véhicule multi-dépôt statique qui représente un scénario de distribution des commandes entre 58 clients. Voici une description des principales caractéristiques de notre problème :

— 58 clients : Notre exemple implique 58 clients,à l'aide d'une carte d'Algérie on a pu extraire les cordonnées géométriques de chaque wilaya en algérie sous forme d'une matrice, et chaque wilaya a sa propre demande, c'est-à-dire la quantité de marchandises à livrer, voici la figure qui indique les cordonnées géométriques et les demandes de quelques wilayas :

```
function [Problem]=Algeria_Problem()
%Column 1 = customer number, no.1 is the deposit.
%Column 2 is the x-coordinate of the client
%Column 3 is the client's y-coordinate
%Column 4 is the demand of what the customer needs to supply, that of the deposit=0
Problem=...
  [ 1      39.00      60.00      2.00      %Adrar
    2      48.00      7.00       1.00      %Chlef
    3      53.00      21.00      1.00      %Laghouat
    4      74.00      10.00      1.00      %Oum El Bouaghi
    5      68.90      11.90      1.00      %Batna
    6      64.10      5.10       1.00      %Bejia
    7      68.60      16.00      1.00      %Biskra
    8      34.00      33.30      1.00      %Bechar
    9      54.80      5.91       1.00      %Blida
   10      59.50      7.37       1.00      %Bouira
   11      63.10      73.60      2.00      %Tamanrasset
   12      77.50      13.90      1.00      %Tebessa
   13      35.00      15.10      1.00      %Tlemcen
   14      48.50      14.60      1.00      %Triaret
   15      61.10      4.84       1.00      %Tizi Ouzou
   16      55.50      4.75       1.00      %Alger
   17      56.40      16.20      1.00      %Djelfa
   18      68.80      4.51       1.00      %Jijel
   19      66.60      7.93       1.00      %Setif
```

FIGURE 4.1 – les coordonnées géométriques et les demandes pour quelques wilayas.

— La distance entre chaque paire de wilaya est calculée initialement par la loi de la distance euclidienne (les distances calculées sont approximatives et ne reflètent pas les distances réelles telles qu'observées dans la vie réelle) , voici un exemple d'une matrice qui représente la distance entre les 10 premiers wilayas d'Algérie :

	1	2	3	4	5	6	7	8	9	10
1	0	1.6128e+03	1.2431e+03	1.8310e+03	1.6991e+03	1.8110e+03	1.5909e+03	814.9239	1.6905e+03	1.6944e+03
2	1.6128e+03	0	445.9821	785.1751	644.0016	486.3517	674.4064	893.8238	206.6042	345.1785
3	1.2431e+03	445.9821	0	711.1962	549.5980	581.7371	491.4509	679.0147	455.9093	453.0168
4	1.8310e+03	785.1751	711.1962	0	163.2728	331.3880	242.1652	1.3887e+03	588.9238	442.0975
5	1.6991e+03	644.0016	549.5980	163.2728	0	249.7038	123.3288	1.2282e+03	459.5880	313.0380
6	1.8110e+03	486.3517	581.7371	331.3880	249.7038	0	353.7711	1.2374e+03	280.0562	153.8883
7	1.5909e+03	674.4064	491.4509	242.1652	123.3288	353.7711	0	1.1605e+03	512.8580	376.2422
8	814.9239	893.8238	679.0147	1.3887e+03	1.2282e+03	1.2374e+03	1.1605e+03	0	1.0318e+03	1.0910e+03
9	1.6905e+03	206.6042	455.9093	588.9238	459.5880	280.0562	512.8580	1.0318e+03	0	147.6463
10	1.6944e+03	345.1785	453.0168	442.0975	313.0380	153.8883	376.2422	1.0910e+03	147.6463	0

FIGURE 4.2 – Une matrice qui représente la distance entre les 10 premières wilayas d'Algérie.

4.1.2 La fonction de regroupement par distance la plus proche

L'étape de déterminer le nombre de dépôts et de la suite on va déterminer les positions du dépôts :

```
Entrer le nombre de depots a avoir :2
Entrer les positions des depots d une façon décroissante!!
Entrer la position de depot :17

    'la position de depot'    [1]    'est'    'Djelfa '

Entrer la position de depot :1

    'la position de depot'    [2]    'est'    'Adrar '
```

FIGURE 4.3 – la sélection des dépots.

Note : Dans cette implémentation on a sélectionné deux dépôts : **Djelfa** et **Adrar**, sachant qu'on peut choisir n'importe quel dépot.

— Après le choix des dépôts, la fonction va créer une matrice des distances entre les wilayas-dépôts et le reste des wilayas, voici un extrait d'une matrice qui représente la distance entre les dépôts sélectionné et les les 5 premiers wilayas d'Algérie.

```
distancedepotcustomer =

   1.0e+03 *

  Columns 1 through 15

    0.3737    0.1765    0.5598    0.3966    0.4053
    1.6128    1.2431    1.8310    1.6991    1.8110
```

FIGURE 4.4 – La distance entre les dépôts sélectionné et les les 5 premiers wilayas d'Algérie.

— Grace a la matrice des distances, il sera possible de faire la classification (affectation des wilayas au plus proche dépôt).

Après cette étape, on va passer à l'**ACO** pour créer les tournées.

4.1.3 L'Algorithme d'optimisation par colonie de fourmis (ACO)

```
clc;
clear;
close all;
clear all;
tic;
capacidad=15;
numberdepots=input('Entrer le nombre de depots a avoir :');
[model,Peter]=CreateModel(capacidad,numberdepots);
TOSC=0;
tota=0;
for idd=1:numberdepots
hold off
img = imread ('amp.jpg');
min_x = 0;
max_x = 100;
min_y = 0;
max_y = 100;
figure(idd);
x_min = min_x;
x_max = max_x;
y_min = min_y;
y_max = max_y;
imagesc ([x_min x_max ], [y_min y_max], img);
MaxIt=40;
nAnt=50;
Q=1;
alpha=0.9;
beta=3;
rho=0.65;
Totalcost_route=[];
disp(['LES TOURNEES DE DEPOT :' model(idd).municipio(1)])
for r=1:model(idd).rutas
tau0=10*Q/(nVar*mean(model(idd).D(:)));
eta=1./model(idd).D;
tau=tau0*ones(nVar,nVar);
BestCost=[];
empty_ant.Tour=[];
empty_ant.Cost=[];
ant=repmat(empty_ant,nAnt,1);
BestSol.Cost=inf;
if capacidad > max(model(idd).d)
cpt=MaxIt+1;
for it=1:MaxIt
```

```
for k=1:nAnt
ant(k).Tour=model(idd).z(r).ff(r).ruta(1);
generated=(randi([2 nVar1]));
ant(k).Tour= [ant(k).Tour model(idd).z(r).ff(r).ruta(generated)];
for l=2:nVar1-1
i=ant(k).Tour(end);
P=tau(i,:).^alpha.*eta(i,:).^beta;
P(ant(k).Tour)=0;
P=P/sum(P);
j=RouletteWheelSelection(P);
AUX= zeros(size(model(idd).z(r).ff(r).ruta));
if ismember(model(idd).z(r).ff(r).ruta,j)==AUX
while ismember(model(idd).z(r).ff(r).ruta,j)==AUX
i=ant(k).Tour(end);
P=tau(i,:).^alpha.*eta(i,:).^beta;
P(ant(k).Tour)=0;
P=P/sum(P);
j=RouletteWheelSelection(P);
end
end
ant(k).Tour=[ant(k).Tour,j];
end
ant(k).Tour=[ant(k).Tour 1];
ant(k).Cost=CostFunction(ant(k).Tour);
if ant(k).Cost<BestSol.Cost
BestSol=ant(k);
end
end
for k=1:nAnt
tour=ant(k).Tour;
for l=1:nVar1
i=tour(l);
j=tour(l+1);
tau(i,j)=tau(i,j)+Q/ant(k).Cost;
end
end
tau=(1-rho)*tau;
BestCost(it)=BestSol.Cost;
if it==MaxIt [BestSol.Tour,BestSol.Cost,Best]
=exchange2(BestSol.Tour,model(idd).D,BestSol.Cost);
tour=BestSol.Tour;
sizebes=numel(Best);
BestCost(it)=BestSol.Cost;
for i=1:numel(Best)
BestCost(end:end+i)=Best(i);
end
Totalcost_route=[Totalcost_route BestSol.Cost];
```

```
ruta=displayTour(model(idd).municipio,BestSol.Tour);
disp(['La tourn e num ro' num2str(r) ruta]);
tota=tota+BestSol.Cost;
PlotSolution(BestSol.Tour,model(idd),r);
title(['Les tourn es des v hicules ded potmodel(idd).municipio(1)])
pause(0.000000000000001);
end
end
end
disp('------------------------------------')
disp(['Meilleur co t est ' num2str(BestCost(it)) ' Km']);
disp('------------------------------------')
disp(' ')
end
figure(idd+numberdepots);
stem(Totalcost_route,'LineStyle','-.',...
'MarkerFaceColor','red',...
'MarkerEdgeColor','green');
title([' Temp Total De Similation: ' num2str(toc) ' s']);
xlabel('Tourn es');
ylabel(['Les co ts des tourn es de d pot : ' model(idd).municipio(1)]);
grid on;
pause(0.9);
TOSC=TOSC+Totalcost_route(end);
end
disp('************************************************************');
disp(['LE COUT TOTAL DE LA SOLUTION GLOBALE : ' num2str(tota) ' Km']);
disp(['LE TEMPS TOTALE DE SIMULATION : ' num2str(toc) ' s']);
disp('************************************************************');
clear;
```

4.1.4 La méthode d'echange (2-OPT)

```
function [tour,Cost,Best] = exchange2(tour,D,Cost)
n = numel(tour);
Best=zeros([],1);
Best(1,1)=Cost;
zmin = -Cost;
k=1;
while zmin/Cost < -1e-6
    k=k+1;
    zmin = 0;
    i = 0;
    b = tour(n);
```

```
    while i < n-2
        a = b;
        i = i+1;
        b = tour(i);
        Dab = D(a,b);
        j = i+1;
        d = tour(j);
        while j < n
            c = d;
            j = j+1;
            d = tour(j);
            z = (D(a,c) - D(c,d)) + D(b,d) - Dab;
            if z < zmin
                zmin = z;
                imin = i;
                jmin = j;
            end
        end
    end
    if zmin < 0
        tour(imin:jmin-1) = tour(jmin-1:-1:imin);
        Cost = Cost + zmin;
        Best(k,1)=Cost;
    end
end
```

4.2 Exécution et résultats

Dans cette section, on a testé l'algorithme d'optimisation par colonie de fourmis hybride (**HACO**) sur notre modèle ''clients.58" avec deux dépôts et de véhicules homogènes de capacité 15, et les résultats sont discutés par la suite :

En premier nous avons effectué un test pour évaluer l'impact de la méthode de recherche locale 2-opt. Par la suite, nous avons modifier les paramètres à chaque exécution (5 exécutions) pour évaluer leur influence sur le résultat.

Notre objectif est d'obtenir le meilleur paramétrage pour notre problème.

4.2.1 L'impact de la méthode de recherche locale 2-opt

On a effectué deux tests pour voir l'impact de 2-opt :

Le premier test (par rapport à le nombre d'itérations (MaxIt)) : dans ce test on a testé sur quatre nombres d'itérations différents (5, 50, 100, 200) sans et avec 2-opt.

Cap = 15, $Maxit = 5$, nAnt = 40, alpha=1, Beta=1, $Q = 1$, rho=0,65 (Sans 2-opt)			Cap = 15, $Maxit = 5$, nAnt = 40, alpha=1, Beta=1, $Q = 1$, rho=0,65 (Avec 2-opt)		
Exécution	Résultat (Km)	Temps d'exécution (sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	22448,9881	8,9915	1	20475,1383	16,4257
2	21485,3204	9,9807	2	20618,5667	10,9528
3	22160,6441	10,8840	3	20771,7934	7,6704
4	22558,9540	8,7513	4	20328,6855	7,7198
5	21950,0476	13,1077	5	20771,7934	6,7632
Moy	22120,7890	10,3430	Moy	20593,1955	9,9063
Cap = 15, $Maxit = 50$, nAnt=40, alpha=1, Beta = 1, $Q = 1$, rho=0,65 (Sans 2-opt)			Cap = 15, $Maxit = 50$, nAnt = 40, alpha=1, Beta = 1, $Q = 1$, rho = 0,65 (Avec 2-opt)		
Exécution	Résultat (Km)	Temps d'exécution (Sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	20737,5331	15,0241	1	20272,1908	15,4023
2	20314,2702	14,8295	2	20328,6855	10,9889
3	20498,1934	12,1537	3	20328,6855	11,2284
4	20445,8565	13,1244	4	20439,0426	11,2127
5	20568,1099	12,1452	5	20392,1118	11,6767
Moy	20514.7926	12,2553	Moy	20352,1432	12,1018

TABLE 4.1 – test d'impact de 2-opt sur le nombre d'iterations (MaxIt) (partie 1).

Cap = 15, $Maxit = 100$, nAnt=40, alpha=1, Beta=1, $Q = 1$, rho=0,65 (Sans 2-opt)			Cap = 15, $Maxit = 100$, nAnt = 40, alpha=1, Beta=1, $Q = 1$, rho=0,65 (Avec 2-opt)		
Exécution	Résultat (Km)	Temps d'exécution (sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	20430,9570	20,7348	1	20328,6855	19,1596
2	20476,7890	21,6241	2	20272,1908	16,1191
3	20496,2189	19,3237	3	20317,6914	15,0480
4	20361,2995	21,0105	4	20317,6914	18,1742
5	20442,4855	20,2854	5	20328,6855	16,1169
Moy	20441,5500	20,5957	Moy	20312,9889	16,9235
Cap = 15, $Maxit = 200$, nAnt=40, alpha=1, Beta = 1, $Q = 1$, rho=0,65 (Sans 2-opt)			Cap = 15, $Maxit = 200$, nAnt = 40, alpha = 1, Beta = 1, $Q = 1$, rho=0,65 (Avec 2-opt)		
Exécution	Résultat (Km)	Temps d'exécution (Sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	20367,3645	26,1433	1	20357,1118	33,3354
2	20408,5751	36,0898	2	20261,1968	25,1583
3	20457,3529	32,9750	3	20272,1908	25,6528
4	20344,6553	32,2159	4	20261,1968	29,1922
5	20362,9836	33,1222	5	20392,1118	26,6958
Moy	20388,1869	32.1092	Moy	20308,7616	28.0069

TABLE 4.2 – test d'impact de 2-opt sur le nombre d'iterations (MaxIt) (partie 2).

Dans ce test on a observé que :

— Les tests effectués avec 2-opt ont des résultats meilleurs que les tests sans 2-opt en termes de solution locale (en jaune) et en termes de la moyenne des 5 exécutions.

On a constaté que :

— En prenant chaque algorithme a part en augmentant le nombre d'itération la solution s'améliore mais si on compare les deux approches ensemble chaque fois le nombre d'itération augmente, la différence entre les solutions donner par **HACO** (avec 2-opt) et **ACO** (sans 2-opt) diminué c'est-à-dire l'impact de 2-opt diminue.

Le deuxième test (par rapport le nombre de fourmis (nAnt)) : dans ce test on a testé sur quatre nombres de fourmis différents (5, 40, 80, 150) sans et avec 2-opt.

Cap = 15, Maxit = 50, $nAnt = 5$, alpha = 1, Beta=1, $Q = 1$, rho=0,65 (Sans 2-opt)			Cap = 15, Maxit = 50, $nAnt = 5$, alpha = 1, Beta = 1, $Q = 1$, rho=0,65 (Avec 2-opt)		
Exécution	Résultat (Km)	Temps d'exécution (sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	22223,4477	9,3952	1	20373,1374	6,4830
2	21896,9743	7,9501	2	29350,6371	7,0265
3	21635,0087	8,2407	3	20369,6288	7,6288
4	22087,7896	7,8076	4	20457,4278	5,9307
5	22464,7865	8,9890	5	20272,1908	6,2177
Moy	22061,6014	8,4765	Moy	20364,6044	6,6573
Cap = 15, Maxit = 50, $nAnt = 40$, alpha=1, Beta = 1, $Q = 1$, rho = 0,65 (Sans 2-opt)			Cap = 15, Maxit = 50, $nAnt = 40$, alpha = 1, Beta = 1, $Q = 1$, rho = 0,65 (Avec 2-opt)		
Exécution	Résultat (Km)	Temps d'exécution (Sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	20612,5884	14,3358	1	20272,1908	15,4023
2	20431,2268	13,7220	2	20328,6855	10,9889
3	20471,1925	12,9284	3	20328,6855	11,2284
4	20496,6657	13,3898	4	20439,0426	11,2127
5	20505,0321	14,1316	5	20392,1118	11,6767
Moy	20503,3411	13,7015	Moy	20352,1432	12,1018

TABLE 4.3 – test d'impact de 2-opt sur le nombre de fourmis (nAnt) (partie 1).

Cap = 15, Maxit = 50, $nAnt = 80$, alpha=1, Beta = 1, $Q = 1$, rho = 0,65 (Sans 2-opt)			Cap = 15, Maxit = 50, $nAnt = 80$, alpha = 1, Beta = 1, $Q = 1$, rho = 0,65 (Avec 2-opt)		
Exécution	Résultat (Km)	Temps d'exécution (sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	20359,9493	18,1739	1	20317,6914	15,5462
2	20344,6553	19,4735	2	20272,1908	15,4183
3	20431,0700	20,1528	3	20328,6855	15,7467
4	20327,7521	20,8520	4	20261,1968	16,1968
5	20317,6914	20,9070	5	20345,0024	16,4459
Moy	20356,2236	19,9118	Moy	20306,7534	15,8707
Cap = 15, Maxit=50, $nAnt = 150$, alpha=1, Beta = 1, $Q = 1$, rho = 0,65 (Sans 2-opt)			Cap = 15, Maxit = 50, $nAnt = 150$, alpha = 1, Beta = 1, $Q = 1$, rho = 0,65 (Avec 2-opt)		
Exécution	Résultat (Km)	Temps d'exécution (Sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	20376,8650	24,5553	1	20261,1968	24,1711
2	20261,1968	26,3558	2	20272,1908	25,3111
3	20346,7690	31,5487	3	20261,1968	24,7335
4	20368,6511	26,0674	4	20317,6914	26,9813
5	20327,7521	25,9623	5	20328,6855	22,2374
Moy	20334,4468	26,8979	Moy	20288,1923	24,6868

TABLE 4.4 – test d'impact de 2-opt sur le nombre de fourmis (nAnt) (partie 2).

Dans ce test on a observé que :

— Les tests effectués avec 2-opt ont des résultats meilleurs que les tests sans 2-opt en termes de solution locale (en jaune) et en termes de la moyenne des 5 exécutions.

On a constaté que :

— En prenant chaque algorithme a part en augmentant le nombre d'itération la solution s'améliore mais si on compare les deux approches ensemble chaque fois le nombre d'itération augmente, la différence entre les solutions donner par **HACO** (avec 2-opt) et **ACO** (sans 2-opt) démunie c'est-à-dire l'impact de 2-opt diminue.

4.2.2 Les tests HACO selon les paramètres d'ACO

TEST 1 : **les tests selon alpha** : dans ce test on a testé sur huit valeurs différentes de alpha (0 ; 0,3 ; 0,5 ; 0,7 ; 0,9 ; 1 ; 2 ; 3).

Cap = 15, Maxit = 50, nAnt=40, $alpha = 0$, Beta=1, Q=1 , rho=0,65			Cap = 15, Maxit = 50, nAnt = 40, $alpha = 0,3$, Beta=1 , Q=1 , rho=0,65		
Exécution	Résultat (Km)	Temps d'exécution (sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	20400,7179	33,2961	1	20445,7355	13,6219
2	20618,2918	32,2885	2	20526,1792	14,0938
3	20455,0067	35,6036	3	20261,1968	15,1669
4	20335,6172	33,5224	4	20526,1792	13,1880
5	20460,8632	32,6993	5	20444,3676	13,3709
Moy	20454,1006	33,4819	Moy	20440,7317	13,8883
Cap = 15, Maxit = 50, nAnt=40, $alpha = 0,5$, Beta=1 , Q=1 , rho=0,65			Cap = 15, Maxit = 50, nAnt = 40, $alpha = 0,7$, Beta=1 , Q=1 , rho=0,65		
Exécution	Résultat (Km)	Temps d'exécution (sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	20550,1051	13,0272	1	20354,0024	15,2556
2	20386,4428	12,0305	2	20261,1968	18,9143
3	20501,2419	12,4373	3	20389,2408	14,4232
4	20261,1968	14,7102	4	20326,2066	16,6549
5	20392,1118	13,9469	5	20271,1908	14,4298
Moy	20418,2197	13,2304	Moy	20320,5675	15,9355

TABLE 4.5 – Test selon des valeurs différentes d'alpha (partie 1).

Cap = 15, Maxit = 50, nAnt=40, $alpha = 0,9$, Beta=1 , Q=1 , rho=0,65			Cap = 15, Maxit = 50, nAnt = 40, $alpha = 1$, Beta=1 , Q=1 , rho=0,65		
Exécution	Résultat (Km)	Temps d'exécution (sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	20317,6914	16,8349	1	20272,1908	15,4023
2	20261,1968	13,2462	2	20328,6855	10,9889
3	20272,1908	13,6612	3	20328,6855	11,2284
4	20328,6855	18,1112	4	20439,0426	11,2127
5	20369,6288	13,7378	5	20392,1118	11,6767
Moy	20309,8787	15,1183	Moy	20352,1432	12,1018
Cap = 15, Maxit = 50, nAnt=40, $alpha = 2$, Beta=1 , Q=1 , rho=0,65			Cap = 15, Maxit = 50, nAnt = 40, $alpha = 3$, Beta=1 , Q=1 , rho=0,65		
Exécution	Résultat (Km)	Temps d'exécution (sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	20328,6855	12,3556	1	20354,0024	12,4361
2	20392,1118	11,8670	2	20706,7835	12,2587
3	20317,6914	11,8885	3	20397,4368	12,8100
4	20375,6162	11,8998	4	20486,6788	11,8671
5	20427,5202	13,1726	5	20335,6172	13,8462
Moy	20368,3790	12,2367	Moy	20456,1037	12,6436

TABLE 4.6 – Test selon des valeurs différentes d'alpha (partie 2).

Dans ces tests on a observé que :

- Lorsque alpha augmente dans l'intervalle [0 ; 0,9], la qualité de solution est améliorée au niveau des deux termes (solution locale, la moyenne).
- Lorsque alpha augmente dans l'intervalle [0,9 ;3], la qualité de solution est diminuée au niveau des deux termes (solution locale, la moyenne).
- La meilleure solution est obtenue quand la valeur d'alpha=0,9.
- Dans ces tests la moyenne des solutions pour les huit cas différents de alpha passe de la mauvaise valeur **20456,1037km** (alpha =3) à la meilleure valeur **20309,8787 km** (alpha= 0,9), on a un gain de **145km**.

TEST 2 : **Les Tests sur BETA** : dans ce test on a testé sur huit valeurs différentes de Beta (0 ; 0,3 ; 0,6 ; 0,7 ; 0,9, 1, 2, 3) :

Cap = 15, Maxit = 50, nAnt=40, alpha=1, Beta = 0, Q=1 , rho=0,65			Cap = 15, Maxit = 50, nAnt = 40, alpha=1, Beta = 0,3, Q=1 , rho=0,65		
Exécution	Résultat (Km)	Temps d'exécution (sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	20492,5300	11,3621	1	20422,3539	19,4132
2	20389,2408	11,3495	2	20613,429	12,9019
3	20998,0610	11,1364	3	20272,1908	12,4410
4	20635,0045	11,5143	4	20272,1908	13,5714
5	20272,1908	11,3365	5	20335,6172	13,4880
Moy	20557,3694	11,3394	Moy	20383,1563	14,3631
Cap = 15, Maxit = 50, nAnt = 40, alpha=1, Beta = 0,6, Q=1 , rho = 0,65			Cap = 15, Maxit = 50, nAnt = 40, alpha = 1, Beta = 0,7, Q=1 , rho=0,65		
Exécution	Résultat (Km)	Temps d'exécution (Sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	20369,5945	13,6300	1	20261,1968	14,6918
2	20261,1968	12,9210	2	20272,1908	12,2074
3	20444,3676	13,1968	3	20261,1968	13,6196
4	20410,4971	13,0150	4	20272,1908	16,4173
5	20272,1908	13,2423	5	20364,6222	12,1001
Moy	20351,5505	13,2190	Moy	20286,2794	69,0329

TABLE 4.7 – Test selon des valeurs différentes Beta (partie 1).

Cap = 15, Maxit = 50, nAnt=40, alpha=1, Beta = 0.9, Q = 1, rho=0,65			Cap = 15, Maxit = 50, nAnt = 40, alpha=1, Beta = 1, Q = 1, rho=0,65		
Exécution	Résultat (Km)	Temps d'exécution (sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	20337,2007	13,3659	1	20272,1908	15,4023
2	20369,6288	12,6665	2	20328,6855	10,9889
3	20328,6855	12,3031	3	20328,6855	11,2284
4	20272,1908	13,0612	4	20439,0426	11,2127
5	20328,6855	12.4306	5	20392,1118	11,6767
Moy	20327,2783	12.7662	Moy	20352,1432	12,1018
Cap = 15, Maxit = 50, nAnt = 40, alpha=1, Beta = 2, Q = 1, rho = 0,65			Cap = 15, Maxit = 50, nAnt = 40, alpha = 1, Beta = 3, Q = 1, rho=0,65		
Exécution	Résultat (Km)	Temps d'exécution (Sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	20375,6162	20,5767	1	20272,1908	23,6901
2	20272,1908	16,1654	2	20261,1968	26,1686
3	20272,1908	14,9216	3	20272,1908	24,0816
4	20272,1908	12,8021	4	20272,1908	25,0511
5	20261,1968	13,0641	5	20354,0024	24,8973
Moy	20290,6771	15,5960	Moy	20286,3543	24,7777

TABLE 4.8 – Test selon des valeurs différentes Beta (partie 2).

Dans ces tests on a observé que :

— Lorsque Beta augmente dans l'intervalle [0 ; 0,7], la qualité de solution est améliorée au niveau des deux termes (solution locale, la moyenne).
— Lorsque Beta augmente dans l'intervalle [0,7 ; 1], la qualité de solution est diminuée au niveau des deux termes (solution locale, la moyenne).
— Lorsque Beta augmente dans l'intervalle [1 ; 3], la qualité de solution est améliorée au niveau des deux termes (solution locale, la moyenne).
— La meilleure solution est générée quand la valeur de Beta=0,7.
— Dans ces tests la moyenne des solutions pour les quatre cas différents de Beta passe de la mauvaise valeur **20557.3694** km (Beta=0) à la meilleure valeur **20286.2794** km (Beta=0.7), on a un gain de **271 km**.

TEST 3 : **Les tests selon rho** : dans ces tests on a testé sur quatre valeurs différentes de rho :

— rho = 0,01 (pas d'évaporation) : nous allons prendre en compte à 100% les expériences précédentes, Cependant, il convient de noter que nous pourrions faire face au risque d'une convergence prématurée car on est en train d'inciter l'opération de L'exploitation.
— rho= 0,99 (évaporation totale) : Dans chaque itération, nous recommencerons à zéro, en favorisant le processus d'exploration.

Cap = 15, Maxit = 50, nAnt=40, alpha=1, Beta = 1, $Q = 1$, $rho = 0,01$			Cap = 15, Maxit = 50, nAnt = 40, alpha=1, Beta = 1, $Q = 1$, $rho = 0,5$		
Exécution	Résultat (Km)	Temps d'exécution (sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	20358,6374	15,7652	1	20337,2007	11,2797
2	20272,1908	12,6809	2	20261,1968	12,0381
3	20410,4971	13,1024	3	20364,6222	11,5344
4	20317,6914	11,4128	4	20328,6855	13,7264
5	20328,6855	11,2582	5	20272,1908	13,8631
Moy	20373,5399	12,8439	Moy	20312,7792	12,4883
Cap = 15, Maxit = 50, nAnt = 40, alpha=1, Beta = 1, $Q = 1$, $rho = 0,65$			Cap = 15, Maxit = 50, nAnt = 40, alpha = 1, Beta = 1, $Q = 1$, $rho = 0,99$		
Exécution	Résultat (Km)	Temps d'exécution (Sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	20272,1908	15,4023	1	20369,6288	16,3813
2	20328,6855	10,9889	2	20408,5619	12,3664
3	20328,6855	11,2284	3	20272,1908	16,9931
4	20439,0426	11,2127	4	20350,6371	12,1801
5	20392,1118	11,6767	5	20364,6222	12,3165
Moy	20352,1432	12,1018	Moy	20353,1282	14,0474

TABLE 4.9 – Test selon des valeurs différentes rho.

Dans ces tests on a observé que :

— Lorsque rho est égal à 0,01, 0,65 et 0,99, nous avons obtenu des solutions similaires au niveau de la solution locale, et au niveau de la moyenne c'est presque identique pour tous les cas.
— Lorsque rho est égal à 0,5, nous avons obtenu de meilleurs résultats pour les deux termes.
— Dans ces tests la moyenne des solutions pour les quatre cas différents de rho passe de la mauvaise valeur **20373.5399 km** (rho=0.01) à la meilleure valeur **20312.7792 km** (rho=0.5), on a un gain de **52 km**.

TEST 4 : **selon le nombre d'itération (MaxIt)** : dans ces tests on a testé sur quatre valeurs différentes de nombre d'itération (5 ; 50 ; 100 ; 200) :

Cap = 15, *Maxit* = 5, nAnt = 40, alpha=1, Beta=1, $Q = 1$, rho=0,65			Cap = 15, *Maxit* = 50, nAnt = 40, alpha=1, Beta = 1, $Q = 1$, rho = 0,65		
Exécution	Résultat (Km)	Temps d'exécution (sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	20475,1383	16,4257	1	20272,1908	15,4023
2	20618,5667	10,9528	2	20328,6855	10,9889
3	20771,7934	7,6704	3	20328,6855	11,2284
4	20328,6855	7,7198	4	20439,0426	11,2127
5	20771,7934	6,7632	5	20392,1118	11,6767
Moy	20593,1955	9,9063	Moy	20352,1432	12,1018
Cap = 15, *Maxit* = 100, nAnt=40, alpha=1, Beta=1, $Q = 1$, rho=0,65			Cap = 15, *Maxit* = 200, nAnt = 40, alpha = 1, Beta = 1, $Q = 1$, rho=0,65		
Exécution	Résultat (Km)	Temps d'exécution (Sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	20328,6855	19,1596	1	20357,1118	33,3354
2	20272,1908	16,1191	2	20261,1968	25,1583
3	20317,6914	15,0480	3	20272,1908	25,6528
4	20317,6914	18,1742	4	20261,1968	29,1922
5	20328,6855	16,1169	5	20392,1118	26,6958
Moy	20312,9889	16,9235	Moy	20308,7616	28,0069

TABLE 4.10 – Test selon des valeurs différentes nombre d'itération .

Dans ces tests on a observé que :

— Lorsque le nombre d'itération augmente la qualité de solution s'améliore au niveau des deux termes (solution locale et la moyenne).
— Dans ces tests la moyenne des solutions pour les quatre cas différents de MaxIt passe de la mauvaise valeur **20593.1955km** (MaxIt=5) à la meilleure valeur **20308.7616 km** (MaxIt=200) , on a un gain de **285 km**.

TEST 5 : **Les tests selon le nombre de fourmis (nAnt)** : dans ces tests on a testé sur quatre valeurs différentes de nombre de fourmis :

Cap = 15, Maxit = 50, $nAnt = 5$, alpha = 1, Beta = 1, $Q = 1$, rho=0,65			Cap = 15, Maxit = 50, $nAnt = 40$, alpha=1, Beta = 1, $Q = 1$, rho = 0,65		
Exécution	Résultat (Km)	Temps d'exécution (sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	20373,1374	6,4830	1	20272,1908	15,4023
2	29350,6371	7,0265	2	20328,6855	10,9889
3	20369,6288	7,6288	3	20328,6855	11,2284
4	20457,4278	5,9307	4	20439,0426	11,2127
5	20272,1908	6,2177	5	20392,1118	11,6767
Moy	20364,6044	6,6573	Moy	20352,1432	12,1018
Cap = 15, Maxit = 50, $nAnt = 80$, alpha=1, Beta = 1, $Q = 1$, rho = 0,65			Cap = 15, Maxit = 50, $nAnt = 150$, alpha = 1 Beta = 1, $Q = 1$, rho = 0,65		
Exécution	Résultat (Km)	Temps d'exécution (Sec)	Exécution	Résultat (Km)	Temps d'exécution (Sec)
1	20317,6914	15,5462	1	20261,1968	24,1711
2	20272,1908	15,4183	2	20272,1908	25,3111
3	20328,6855	15,7467	3	20261,1968	24,7335
4	20261,1968	16,1968	4	20317,6914	26,9813
5	20345,0024	16,4459	5	20328,6855	22,2374
Moy	20306,7534	15,8707	Moy	20288,1923	24,6868

TABLE 4.11 – Test selon des valeurs différentes nAnt.

Dans ces tests on a observé que :

— Lorsque le nombre de fourmis augmente la qualité de solution améliore au niveau des deuxtermes (solution locale et la moyenne).
— Dans ces tests la moyenne des solutions pour les quatre cas différents de nAnt passe de la mauvaise valeur **20364.6044 km** (nAnt=5) à la meilleure valeur **20288.1923 km** (nAnt=150), on a un gain de **76 km**.

TEST 6 : **Test qui regroupe les meilleurs paramètre** : dans ce test on a travaillé avec les meilleurs paramètres de chacune des cas précédents :

Exécution	Résultat (Km)	Temps d'exécution (sec)
1	20261,1968	100,1061
2	20261,1968	104,3053
3	20261,1968	96,2484
4	20272,1908	75,0040
5	20261,1968	87,4886
Moy	20263,3956	94,3824

TABLE 4.12 – Test selon les meilleurs valeurs (Cap = 15, Maxit=200, nAnt=150, alpha=0,9, Beta=1, Q=1, rho=0,50).

Test Finale : Comparaison entre le résultat (Meilleure solution locale est la moyenne) de meilleur paramétrage et le meilleur résultat de chaque des tests précédents :

	Test 6 Meilleurs paramètres	Test 1 Alpha	Test 2 Beta	Test 3 Rho	Test 4 MaxIt	Test 5 nAnt
Meilleur solution locale (Km)	20261,1968	20261,1968	20261,1968	20261,1968	20261,1968	20261,1968
Moyenne (Km)	20263,3956	20309,8787	20286,2794	20312,7792	20308,7616	20288,1923

TABLE 4.13 – Comparaison entre tous les tests.

Dans ces tests on a observé que :

— La meilleur solution local est la même pour tous les tests précédents.
— La moyenne le test qui regroupe les meilleurs paramètres est la meilleure moyenne obtenue tous les tests précédents.

4.3 Conclusion

En conclusion, dans ce chapitre on a présenté notre modèle "Algérie.58" ainsi que la fonction de regroupement par distance la plus proche et la méthode d'échange (**2-opt**) avec des exemples illustratifs en détail.

Par la suite, nous avons introduit l'algorithme de Colonie de Fourmis **(ACO)** et fourni son code Matlab correspondant. Nous avons ensuite effectué deux tests pour vérifier l'efficacité de la méthode d'échange **2-opt** dans notre modèle

En poursuivant nos expérimentations, nous avons réalisé des tests en modifiant les valeurs des paramètres de l'**ACO**. Cette exploration nous a permis de comprendre l'impact de ces paramètres sur les performances du modèle. Nous avons ainsi pu identifier les configurations de paramètres les plus performantes pour notre problème.

Enfin, nous avons procédé à un dernier test en regroupant les meilleurs paramètres de chacun des tests précédents. Ce test a été réalisé en combinant les meilleurs paramètres identifiés dans les tests précédents et les résultats ont été présentés sous forme d'un tableau comparatif.

Conclusion générale

En conclusion générale, cette mémoire a abordé différents aspects de l'optimisation et l'optimisation combinatoire. Dans le premier chapitre, nous avons introduit les concepts de l'optimisation et de l'optimisation combinatoire et présenter des exemples académiques connues. Nous avons examiné les méthodes de résolution exactes et approchées utilisées pour aborder ce problème complexe.

Le deuxième chapitre s'est concentré sur le **VRP**, en explorant ses différentes variantes et en mettant en évidence les défis associés à sa résolution.

Le troisième chapitre a présenté l'algorithme de Colonie de Fourmis **(ACO)**, en expliquant son origine dans la nature et la notion d'intelligence collective des fourmis. Nous avons décrit en détail les étapes de l'algorithme de colonie de fourmis et illustré son application à travers quelques problèmes résolus.

Dans le quatrième chapitre, nous avons proposé une application hybride de l'**ACO** pour le problème du VRP à plusieurs dépôts **(MDVRP)**, en combinant l'**ACO** avec la méthode d'échange **(2-opt)** et nous avons présenté la procédure de cette approche.

Le cinquième chapitre a exposé notre application de l'approche hybride **ACO + 2-opt** au **MDVRP**, en décrivant les étapes spécifiques de notre méthode. Nous avons réalisé des tests pour évaluer l'impact de la méthode d'échange **(2-opt)** ainsi que des tests en changeant les paramètres de l'**ACO**. Nous avons également effectué un test en regroupant les meilleurs paramètres identifiés précédemment et comparé les résultats obtenus avec les tests antérieurs.

En guise de perspectives futures, il serait intéressant d'explorer la version dynamique du problème **MDVRP**, car dans la réalité, ce problème est souvent soumis à des variations et des contraintes dynamiques. La recherche de solutions adaptées à ces situations changeantes serait un domaine de recherche prometteur.

De plus, il est recommandé de trouver une solution pour obtenir les coordonnées géométriques exactes des wilayas d'Algérie. Cela permettrait une représentation plus précise du problème et faciliterait l'application des méthodes d'optimisation, telles que l'**ACO**, dans un contexte réel.

En résumé, cette mémoire a exploré divers aspects de l'optimisation combinatoire, en se concentrant sur le **VRP** et en appliquons une approche hybride **(ACO + 2-opt)** pour résoudre le **MDVRP**. Les résultats obtenus ont montré l'efficacité de cette méthode, et des perspectives futures intéressantes ont été identifiées pour améliorer et étendre les recherches dans ce domaine.

Bibliographie

[AF07] C. Archetti and M. Fischetti. A hybrid genetic algorithm for the heterogeneous fleet vehicle routing problem with time windows. *European Journal of Operational Research*, 176(1), 2007.

[AFG14] C. Archetti, M. Fischetti, and F. Guerriero. A tabu search heuristic for the vehicle routing problem with multiple objectives. *Computers amp ; Operations Research*, 41(11), 2014.

[AL04] Talbi-E.-G. Azzag, M. and B. Lepagnot. Ant colony optimization for feature selection. In *Proceedings of the 2004 IEEE International Conference on Evolutionary Computation (CEC 2004)*, pages 2141–2148, 2004.

[ALI14] Dr.LEMOUARI ALI. Introduction aux métaheuristiques, 2014.

[ant23] Ant colony optimization algorithms. https ://en.wikipedia.org/wiki/Ant colony optimization algorithms, 2023. Accessed on May 8, 2023 at 10 :30 PM.

[BHZMAeh21] Ramin Bazrafshan, Sarfaraz Hashemkhani Zolfani, and S. Mohammad J. Mirzapour Al-e hashem. Comparison of the sub-tour elimination methods for the asymmetric traveling salesman problem applying the seca method. *Axioms*, 10(1) :19, 2021.

[CDM91] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In *Proceedings of the European Conference on Artificial Life, ECAL '91*, pages 134–142. Elsevier Publishing, 1991.

[Dab] Ali Dabba. Méthodes de résolution exactes heuristiques et métaheuristiques, optimisation des réseaux. Course material. Université de M'sila.

[DAGP90] J. L. Deneubourg, S. Aron, S. Goss, and J. M. Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior*, 3(2) :159–176, 1990.

[DC99] M. Dorigo and G. D. Caro. Ant colony optimization : A new meta-heuristic. In *Proceedings of the 1999 Congress on Evolutionary Computation*, volume 2, pages 1470–1477, 1999.

[Dja15] Zaouache Djaafar. Optimisation combinatoire, October 19 2015.

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system : Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1) :29–41, 1996.

[DS04] Marco Dorigo and Thomas Stützle. *Ant colony optimization*. MIT press, 2004.

[For13] Bernard Fortz. Recherche opérationnelle et application, 2013.

[GHB15] Bin Ge, Yue Han, and Chen Bian. Hybrid ant colony optimization algorithm for solving the open vehicle routing problem. *Journal of Computer Science and Technology*, 30(4) :773–782, 2015.

[Ham22] Sylvie Hamel. Algorithmes voraces - sac à dos 1, 2022. Cours dispensé à l'Université de Montréal, Code du cours IFT2125.

[HGH99] Jin-Kao Hao, Philippe Galinier, and Michel Habib. Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. *Revue d'Intelligence Artificielle*, 13(2) :129–154, 1999.

[Kaj20] Sai Chandana Kaja. A new approach for solving the disruption in vehicle routing problem during the delivery : A comparative analysis of vrp meta-heuristics. *Unknown Journal*, 2020. Master of Science in Computer Science, May 2020.

[Mal06] Arnaud Malapert. Optimisation de tournées de véhicules pour l'exploitation de réseau telecom. Rapport de stage diplôme master intelligence artificielle et décision, Université Paris 6, 2006.

[McC12] James McCaffrey. Test run - ant colony optimization, February 2012.

[Mon99] N. Monmarche. Ant colony optimization and its application to image segmentation. In *Proceedings of the International Conference on Image Analysis and Processing (ICIAP '99)*, pages 512–517, 1999.

[NAK18] Abolfazl Nouri, Mohammadreza Amini, and Amir Hossein Karimi. Hybrid ant colony optimization for capacitated multiple-allocation cluster hub location problem. *AI EDAM*, 32(2), 2018.

[OB07] S. Ouadfel and M. Batouche. An efficient ant algorithm for swarm-based image clustering. *Journal of Computer Science*, 3(3) :162–167, 2007.

[SL07] Patrick St-Louis. Nouveaux algorithmes, bornes et formulations pour les problèmes de la clique maximum et de la coloration minimum. 2007.

[Tou21] Hillal Touati. Eléments de théorie des ghraphes, October 25 2021.

[XPD18] Haitao Xu, Pan Pu, and Feng Duan. Dynamic vehicle routing problems with enhanced ant colony optimization. *Discrete Dynamics in Nature and Society*, 2018 :1–13, 2018.

Résumé

Dans ce mémoire, on a appliqué une optimisation par colonies de fourmis hybride pour le problème de routage de véhicules multidépôts statique à l'aide de l'algorithme de de regroupement par distance la plus proche , et la méthode d'échange local (2-opt) et opération de mutation afin de Gagner de la distance et ainsi réduire le coût en général avec la condition de la capacité limitée desvéhicules.

Mots clés : Optimisation combinatoire , Optimisation par colonie de fourmis, 2-opt, MDVRP , Algorithme hybride.

Abstract

In this thesis, we applied a hybrid Ant Colony Optimization for Multidepot Vehicle Routing Problem using the Nearest Distance Cluster Algorithm, 2-opt and mutation operation in order to Gain the distance and thus reduce the cost in general with the condition of the limited capacity of Vehicles.

keywords : Combinatorial optimization, Ant colony optimization, 2-opt, MDVRP, hybrid Algorithm.

الملخص

في هذه المذكرة، قمنا بتطبيق تحسينا هجينا لمستعمرة النمل لمشكلة توجيه المركبات الديناميكية متعددة النقاط باستخدام خوارزمية أقرب مسافة للمجموعة، عملية تبادل المحلي و عملية الطفرة من أجل كسب المسافة و بالتالي تقليل التكلفة بشكل عام مع اخذ بعين الاعتبار السعة المحدودة للمركبات.

الكلمات المفتاحية :تحسين مستعمرة النمل الهجين، تحسين مستعمرة النمل، مشكلة توجيه المركبة، الخوارزمية الهجينة.