République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene

Faculté de Mathématiques

Département de Recherche Opérationnelle

Rapport

En vue de l'obtention du Diplôme de LICENCE
en Recherche Opérationnelle

Thème

Problème du voyageur de commerce dans l'algorithme de colonie de fourmis TSP-AC

Présenté par : YADEL Gaya
SOUIDI Mohamed-Said

Suivi par : KHALFI Abderaouf

Année universitaire 2022/2023

REMERCIEMENTS

Je tiens à prendre quelques instants pour exprimer ma profonde gratitude envers toutes les personnes qui ont apporté leur soutien et leur contribution à la réalisation de ce mémoire. Tout d'abord, je voudrais remercier chaleureusement M.KHALFI ABDERAOUF pour sa précieuse aide, son soutien inconditionnel et ses précieux conseils tout au long de ce projet. Sans votre expertise et vos encouragements, ce mémoire n'aurait pas été possible.

Je voudrais également exprimer ma reconnaissance envers ma famille et mes amis pour leur soutien et leur compréhension pendant les moments où j'ai dû me concentrer sur ce projet. Leur présence à mes côtés et leur patience ont été essentielles pour me permettre de mener à bien ce travail sans me soucier des distractions extérieures.

Enfin, je souhaite remercier toutes les personnes qui ont collaboré avec moi, qui m'ont soutenu et encouragé tout au long de ce parcours. Votre contribution a été déterminante pour la réalisation de ce travail. Merci infiniment pour votre soutien et votre confiance.

Contents

List of Figures

1 Introduction

i)Définition de la recherche opérationnelle :
La recherche opérationnelle est une méthode qui consiste à utiliser des techniques mathématiques, informatique et économique pour résoudre des problèmes de décision dans des situations incertaines. Bien qu'elle soit souvent associée aux opérations militaires de la Seconde Guerre mondiale, elle est en réalité beaucoup plus ancienne. Les premiers travaux sur la recherche opérationnelle ont été menés par des mathématiciens tels que Blaise Pascal et Pierre de Fermat au XVIIe siècle, qui cherchaient à résoudre des problèmes de décision dans des situations incertaines. Au cours des siècles suivants, d'autres mathématiciens ont contribué à cette méthode en développant des théories telles que la théorie mathématique des jeux et la programmation linéaire. Ainsi, la recherche opérationnelle est une méthode importante qui a été développée au fil du temps pour aider à résoudre des problèmes complexes.[4]

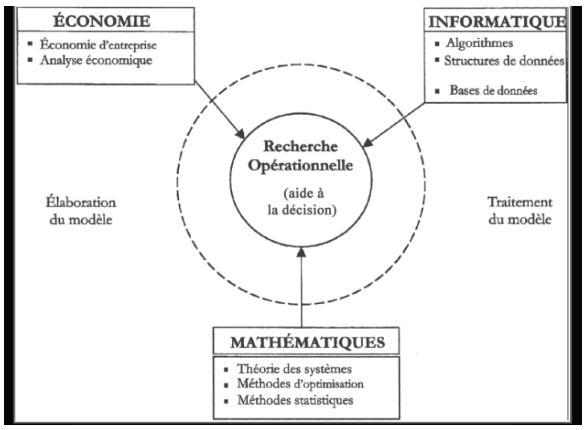

Figure 1: RO

ii)Domaine de la recherche opérationnelle :
La recherche opérationnelle trouve des applications dans divers domaines tels que la gestion des opérations, la logistique, la finance, l'économie, l'ingénierie, la santé, les transports, l'environnement, etc. Elle étudie trois types de problèmes dans les domaines mentionnés ci-dessus : les problèmes combinatoires, les domaines de l'aléatoire et les situations de concurrence, chacun caractérisé par des défis spécifiques. En utilisant des outils mathématiques et informatique, la recherche opérationnelle propose des solutions optimales pour résoudre ces problèmes.

Exemples d'utilisation de la recherche opérationnelle :
Voici quelques exemples d'applications de la recherche opérationnelle :

1. Gestion de la chaîne d'approvisionnement : la recherche opérationnelle est utilisée pour optimiser la planification de la production, la gestion des stocks, le transport et la distribution des produits afin d'améliorer l'efficacité et de réduire les coûts.

2. Planification des horaires : la recherche opérationnelle est utilisée pour planifier les horaires des employés en prenant en compte les contraintes de disponibilité, les coûts et les exigences de service.

3. Gestion de la production : la recherche opérationnelle est utilisée pour optimiser les processus de production, minimiser les temps d'arrêt, améliorer la qualité et réduire les coûts de production.

4. Gestion de la logistique : la recherche opérationnelle est utilisée pour optimiser la planification des itinéraires, la gestion des flottes de véhicules, la gestion des entrepôts et la distribution des marchandises.

2 Généralités et notions de bases :

2.1 Généralités sur la théorie des graphes :

i) Introduction :
La théorie des graphes est née au XVIIe siècle grâce au mathématicien Leonhard Euler qui a brillamment résolu le fameux "problème du pont de Königsberg". Ce problème, représenté dans la figure 1.2, consistait à déterminer si on pouvait parcourir les sept ponts de Königsberg une seule fois, en empruntant un trajet continue, et si une telle solution existait, où se trouvait-elle. Autrement dit, Euler s'est penché sur la question de la recherche d'un cycle fermé [7].

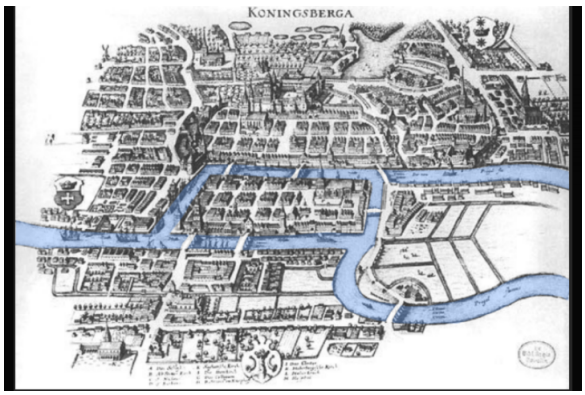

Figure 2: Problème des ponts de Königsberg

ii) Définitions :

Un graphe orienté G est constitué d'un ensemble X des sommets telle que $X \neq \emptyset$ et d'un ensemble U des arcs, ou l'arc $u = (x_i, x_j)$ relie le sommet x_i au sommet x_j. On note : $G = (X, U)$.
On appelle un graphe non orienté $G = (X, E)$ où E est l'ensemble des arêtes telle que $e \in E; e = (x_i, x_j) = (x_j, x_i)$.
Le nombre de sommets dans un graphe est appelé l'ordre du graphe : $X = n$.
Le nombre d'arcs (ou arêtes) dans un graphe est appelé la taille du graphe : $|U| = m$.
Pour un arc $u = (x, y)$, le sommet x est appelé son extrémité initiale, et le sommet y est appelé son extrémité terminale .
Un élément (x, y) de X peut apparaître plus d'une fois dans l'ensemble des arc (ou arêtes).
Un graphe dans lequel aucun élément de X n'apparaît plus de p fois est appelé un p-graphe.
On appelle une boucle un arc de G de la forme (x, x).
Deux arêtes sont dites multiples si elles ont les mêmes extrémités.
Si le graphe G est sans boucle ni arête multiple alors G est un graphe simple, autrement dit un graphe simple est un 1-graphe sans boucle.
Le sommet y est appelé successeur du sommet x s'il existe un arc dont le point d'extrémité initiale est y et le point d'extrémité terminale est x. L'ensemble de tous les successeurs de x est noté par : $\Gamma^+_G(x)$.
Le sommet y est appelé prédécesseur du sommet x s'il existe un arc de la forme (y, x). L'ensemble de tous les prédécesseurs du sommet x est noté par : $\Gamma^-_G(X)$.
L'ensemble des sommets voisins du sommet x est noté par : $\Gamma_G(x) = \Gamma^+_G(X) \bigcup \Gamma^-_G(x)$
On appelle le degré d'un sommet x le nombre d'arcs ayant x comme extrémité.
Le degré de x est noté par : $d_G(x) = d^+_G(x) + d^-_G(x)$
Une chaîne P est une succession d'arcs de G avec $p = (u1, u2, ..., uq)$ telle que chaque arc de la séquence partage un point d'extrémité avec son prédécesseur dans la séquence et l'autre point d'extrémité avec son successeur.
La longueur de la chaîne P est égale au nombre d'arcs dans la séquence.
Une chaîne élémentaire est une chaîne qui ne passe pas deux fois par le même sommet.
Un chemin de x a y est une séquence ordonnée d'arcs continus commençant du sommet x et permettant un déplacement de proche en proche vers le sommet y.
Un cycle C est une chaîne dans laquelle aucun arc n'apparaît deux fois dans la séquence et les deux extrémités de la chaîne sont le même sommet.
On appelle un graphe pondéré, un graphe G dont toutes ses arcs (ou arêtes) sont étiqueté par des nombres $p_i \geq 0$, tels que les p_i sont les poids des arcs (ou arêtes).
On note $G = (X, E, p)$. (dans ce qui suit les poids des arêtes représentent les distances entre les villes (sommets)).

Soit $G = (X, E)$ un graphe, on appelle un cycle Hamiltonien, un cycle qui passe par tous les sommets du graphe G une et une seule fois. Si le graphe $G = (X, E)$ contient un tel cycle alors G est un graphe Hamiltonien.

2.2 Généralités sur la programmation Linéaire en nombres entiers PLNE :

i) Définition de la PLNE :
La programmation linéaire en nombre entier est une extension de la programmation linéaire standard dans laquelle les variables de décision sont restreintes à prendre des valeurs entières, elle est utilisé pour modéliser des problèmes de décision qui nécessitent des solutions entières. Cependant, l'ajout de contraintes entiers peut rendre le problème considérablement plus difficile à résoudre. En effet, la programmation linéaire en nombre entier est connue pour être un problème NP-difficile (non-déterministes polynomial), ce qui signifie que la résolution exacte du problème peut être très coûteuse en termes de temps et de ressources. Pour cette raison, diverses techniques de résolution ont été développées pour obtenir des solutions optimales ou approchées, telles que les méthodes de coupes et les algorithmes de branch-and-bound. Ces techniques ont été mises en œuvre dans des solveurs commerciaux efficaces qui permettent de résoudre des problèmes de grande taille en un temps raisonnable.[1]

ii) Domaine d'utilisation de la PLNE :
la programmation linéaire en nombre entier est un outil d'optimisation très polyvalent qui peut être appliqué à de nombreux problèmes d'optimisation dans divers domaines, tels que :
- La logistique et la chaîne d'approvisionnement.
- Les télécommunications et la planification des réseaux.
- La finance et la gestion des investissements.
- La planification de la production.[8]

2.3 Généralités sur l'optimisation combinatoire :

L'optimisation combinatoire est un domaine de recherche qui s'intéresse à la résolution de problèmes d'optimisation qui impliquent un grand nombre de solutions possibles. Ces problèmes se posent dans différents domaines tels que l'informatique, l'économie, la biologie ou la physique, en raison de leurs complexité, ces problèmes nécessitent souvent des techniques de modélisation et d'optimisation spécifiques pour obtenir des solutions précises ou des approximations satisfaisantes.[6]

2.3.1 Description des problèmes d'optimisation combinatoire :

Les problèmes d'optimisation combinatoire sont présents dans de nombreux domaines et consistent à trouver la meilleure solution parmi un ensemble fini de solutions possibles. Cependant, la résolution de ces problèmes est complexe car elle implique de prendre en compte plusieurs variables, contraintes et objectifs, ce qui peut rendre leur résolution difficile. De plus, la complexité de ces problèmes peut augmenter rapidement en fonction de la taille de l'ensemble de solutions possibles. Les problèmes d'optimisation combinatoire peuvent être classés en deux catégories en fonction de leur complexité :
Les problèmes de complexité polynomiale peuvent être résolus efficacement à l'aide d'algorithmes.
les problèmes de complexité exponentielle nécessitent des approches plus sophistiquées, telles que l'utilisation de méthodes d'approximation ou de méthodes heuristiques.
la recherche de méthodes efficaces pour résoudre les problèmes d'optimisation combinatoire est un enjeu majeur, étant donné les nombreuses implications pratiques que ces problèmes peuvent avoir dans des domaines tels que la logistique, la production, la planification d'itinéraires, etc.

Voici une petite sélection des nombreux problèmes d'optimisation combinatoire existants:
.Le problème du voyageur de commerce.
.Le problème du sac à dos.
.Le problème d'affectation.
.Le problème de planification de la production .
.Le problème de coloration de graphe.

2.3.2 Approche de résolution des problèmes d'optimisation combinatoire:

Les problèmes d'optimisation combinatoire consistent à trouver la meilleure solution parmi un grand nombre de possibilités. Pour cela, deux approches principales sont utilisées : les méthodes exactes et les méthodes approchées,chacune avec ses propres avantages et limites.

Les méthodes exactes, comme la méthode de Branch Bound, la méthode Branch Cut, etc sont basées sur des algorithmes mathématiques rigoureux qui garantissent de trouver la solution optimale. Cependant, leur temps d'exécution peut rapidement devenir prohibitif pour les problèmes de grande taille, car le nombre de solutions

à examiner augmente de manière exponentielle avec la taille du problème.Elles sont également utilisées pour la vérification de solutions trouvées par d'autres méthodes.

Les méthodes approchées cherchent à fournir une solution satisfaisante dans un temps raisonnable, sans garantir la solution optimale. Ces méthodes sont souvent plus rapides que les méthodes exactes, car elles utilisent des stratégies heuristiques pour explorer efficacement l'espace des solutions à partir d'une solution initiale, en l'améliorant progressivement pour obtenir une solution de meilleure qualité.Elles sont utiles pour résoudre des problèmes plus volumineux ou pour des problèmes pour lesquels il est difficile de définir une fonction objectif précise ou de respecter des contraintes complexes.

Le choix de la méthode la plus appropriée dépendra du contexte de l'application, des caractéristiques du problème à résoudre et des ressources disponibles.

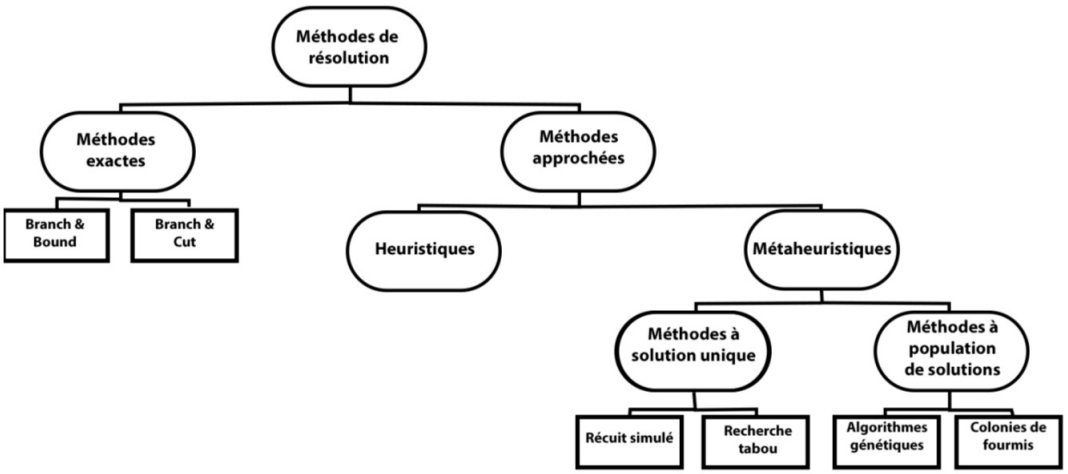

Figure 3: les méthodes de résolution des problèmes d'optimisation combinatoire

3 Problème voyageur de commerce

3.1 Introduction :

Le problème du voyageur de commerce est un problème d'optimisation combinatoire qui consiste à trouver le plus court chemin possible permettant à un voyageur de visiter un ensemble de villes données une seule fois avant de revenir à son point de départ. C'est un problème difficile, car le nombre de chemins possibles à explorer augmente de manière exponentielle avec le nombre de villes à visiter.
Cependant, le problème du voyageur de commerce a de nombreuses applications pratiques, notamment dans le domaine de la logistique, où il est utilisé pour optimiser la planification de tournées de livraison et de collecte de marchandises. Il est également utilisé dans la conception de circuits imprimés pour minimiser la distance parcourue par les machines de forage. Bien que les solutions exactes soient souvent difficiles à obtenir pour les grandes instances, de nombreuses méthodes heuristiques et métaheuristiques ont été développées pour résoudre le problème de manière efficace pour des instances plus petites à moyennes.[2]

3.2 Modélisation en utilisant la théorie des graphes :

Le TSP peut être défini sur un graphe complet non orienté $G = (V, E)$ s'il est symétrique ou sur un graphe orienté $G = (V, A)$ s'il est asymétrique.
L'ensemble $V = \{1, \ldots, n\}$ est l'ensemble des sommets.
$E = \{(i, j) : i, j \in V, i < j\}$ est un ensemble d'arêtes.
$A = \{(i, j) : i, j \in V, i \neq j\}$ est un ensemble d'arcs.
Une matrice de coûts $C = (c_{ij})$ est définie sur E ou sur A. cette dernier satisfait l'inégalité triangulaire lorsque $c_{ij} \leq c_{ik} + c_{kj}$ pour tout i, j, k.
En particulier, c'est le cas des problèmes planaires pour lesquels les sommets sont des points $P_i = (X_i, Y_i)$ dans le plan,
et $c_{ij} = \sqrt{(X_i - X_j)^2 + (Y_i - Y_j)^2}$ est la distance euclidienne.
L'inégalité triangulaire est également satisfaite si c_{ij} est la longueur d'un plus court chemin de i à j sur G.
Voici un exemple illustratif : $G = (V, E)$ un graphe complet pondéré à 4 sommets, chaque sommet représente une ville $V = \{A, B, C, D\}$, les poids des arêtes représentent les distances entre ces villes telle que :

$$C_{ij} = \begin{pmatrix} 0 & 25 & 10 & 15 \\ 25 & 0 & 10 & 45 \\ 10 & 10 & 0 & 5 \\ 15 & 45 & 5 & 0 \end{pmatrix}$$

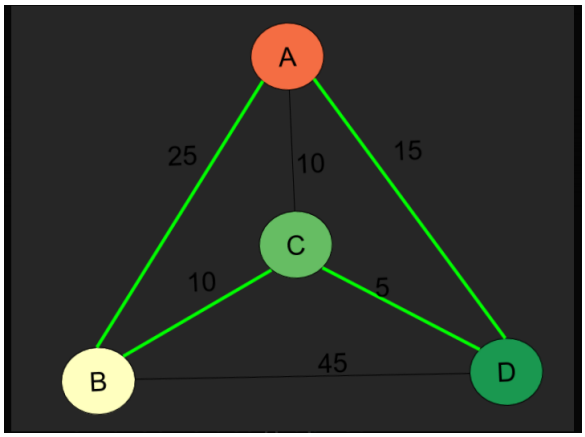

Figure 4: Exemple d'un cycle hamiltonien

le cycle hamiltonien (colorie en vert) représente une solution réalisable du problème, comme le montre la figure ci-dessus,
[5]

3.3 Modélisation en utilisant la PLNE :

Le problème de voyageur de commerce (TSP) peut être traduit mathématiquement au programme linéaire en nombres entiers suivant :

$$(PL) = \begin{cases} \text{Min } Z(X) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\ \sum_{i=1}^{n} x_{ij} = 1 \quad \forall j \in N, i \neq j \\ \sum_{j=1}^{n} x_{ij} = 1 \quad \in N, i \neq j \\ \sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 \quad \forall S \subsetneq V, |S| \geq 2, i \neq j \\ x_{ij} \in \{0,1\} \quad , j \in N, i \neq j \end{cases}$$

La fonction objectif $Z(X)$ est la somme des distances c_{ij} des trajets effectués en parcourant toutes les villes une seule fois et en retournant au point de départ, le but est de minimiser la fonction Z tout en satisfaisant les contraintes suivantes:
la première contrainte nous permet d'assurer qu'un seul trajet vers la ville j a été effectué, $\forall j \in N$.
la deuxième contrainte nous permet d'assurer qu'un seul trajet allant la ville i a été effectué, $\forall i \in N$.
la troisième contrainte est une contrainte d'élimination des sous tours avec S est un sous-ensemble de sommets de V , on limite le nombre maximum d'arcs (trajets effectués) dans S
la quatrième contrainte est appelée contrainte d'intégrité, consiste à spécifier les valeurs que la variable de décision x_{ij} peut prendre, telle que :

$$x_{ij} = \begin{cases} 1 & \text{on effectue le trajet allant de i a j,} \\ 0 & \text{sinon} \end{cases}$$

4 Approche de résolution :

4.1 Introduction

Le problème du voyageur de commerce est l'un des plus célèbres problèmes d'optimisation combinatoire. Il est NP-complet, ce qui signifie qu'il est extrêmement difficile à résoudre exactement. Plus précisément, il n'existe pas d'algorithme connu capable de le résoudre en un temps polynomial en fonction de la taille de l'entrée. Cette difficulté est due à la combinaison de deux facteurs : d'une part, le nombre de permutations possibles des villes augmente de manière exponentielle avec le nombre de villes, et d'autre part, il est difficile d'éliminer rapidement les permutations non optimales. Sa complexité a été démontrée en le réduisant à un autre problème NP-complet, le problème du circuit hamiltonien. Ce dernier consiste à déterminer s'il existe un circuit qui passe par chaque sommet d'un graphe donné exactement une fois.

Bien que le TSP soit difficile à résoudre exactement,il existe plusieurs approches pour ce problème. L'approche brute consiste à énumérer toutes les permutations possibles des villes, ce qui est faisable pour un petit nombre de villes, mais devient rapidement impraticable pour un grand nombre de villes. D'autres approches ont été développées pour fournir des solutions raisonnables en temps polynomial, incluent des méthodes heuristiques comme l'algorithme du plus proche voisin,des méthodes métaheuristiques comme l'algorithme génétique, l'algorithme de colonies de fourmis et l'algorithme de recuit simulé.Ces algorithmes garantissent de trouver une solution mais pas optimale. Ils sont largement utilisés dans la pratique pour résoudre des instances du problème du voyageur de commerce de taille raisonnable.

Le problème du voyageur de commerce est considéré comme l'un des grands défis de la recherche opérationnelle , et il continue de susciter l'intérêt des chercheurs du monde entier. De nombreuses heuristiques ont été proposées pour tenter de trouver des solutions meilleures en pratique, mais la recherche de la solution optimale reste un problème ouvert et difficile.[2]

4.2 Description de l'algorithme de colonie de fourmis :

L'algorithme de colonie de fourmis (ACO) est une méthode d'optimisation métaheuristique qui s'inspire du comportement des fourmis en quête de nourriture. Le processus de recherche de nourriture chez les fourmis est connu sous le nom de "marche aléatoire" et implique que les fourmis se déplacent au hasard dans leur environnement et laissent une trace de phéromone sur leur chemin. Cette trace de phéromone agit comme un signal pour les autres fourmis, qui sont alors plus susceptibles de suivre le même chemin en renforçant la trace de phéromone.

L'ACO utilise un processus similaire pour résoudre le problème du voyageur de commerce. Voici les étapes principales de l'algorithme:
1. Définition des villes et de leur position aléatoire.

2. Génération de la matrice de distance pour toutes les paires de villes.
3. Initialisation des paramètres de l'algorithme, y compris le nombre de fourmis, les valeurs α , β , rho et Q, et le nombre maximum d'itérations.
4. Initialisation des phéromones sur chaque arc entre les villes.
5. Boucle principale pour un certain nombre d'itérations:
a. Construction de solutions pour chaque fourmi, en utilisant la probabilité de transition entre les villes.

$$p_{kij} = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{l \in J_i^k} \tau_{il}^{\alpha}\eta_{il}^{\beta}}$$

où τ_{ij} est la quantité de phéromone déposée sur l'arc (i, j) par toutes les fourmis, η_{ij} est la visibilité de l'arc (i, j), α et β sont des paramètres qui régulent l'importance respective de la quantité de phéromone et de la visibilité, et $\sum_{l \in J_i^k} \tau_{il}^{\alpha}\eta_{il}^{\beta}$ est la somme des valeurs de $\tau_{il}^{\alpha}\eta_{il}^{\beta}$ pour tous les arcs l partant de la ville i et non encore visités par la fourmis k.
b. Évaluation de chaque solution construite par les fourmis.
c. Mise à jour d'évaluation de phéromones sur chaque arc en utilisant une formule basée sur la qualité des solutions trouvées par les fourmis.

$$\Delta\tau_{ij} = \frac{Q}{L_k}$$

où Q est une constante définissant la quantité de phéromone , et L_k est la longueur du parcours emprunté par la fourmis k.
d.Mise à jour d'évaporation de La quantité de phéromone sur tous les arcs est donnée par la formule:

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij}$$

où ρ est le taux d'évaporation de la phéromone.
6. Recherche de la meilleure solution trouvée parmi toutes les solutions évaluées.
7. Affichage de la meilleure solution et de sa distance.[3]

4.3 L'application

```
import random
import math
import numpy as np

# Definition des villes et de leur position
nb_villes = int(input("Entrez le nombre de villes: "))

# Generation de la matrice de distance # La matrice  de distances entre les villes est generee de maniere aleatoire a l'aide de la fonction 'np.random'
villes = []
for i in range(nb_villes):
    villes.append((random.random(), random.random()))

distances = np.zeros((nb_villes, nb_villes))
for i in range(nb_villes):
    for j in range(nb_villes):
        if i != j:
            distances[i][j] = math.sqrt((villes[i][0]-villes[j][0])**2 + (villes[i][1]-villes[j][1])**2)

# Parametres de l'algorithme
nb_fourmis = nb_villes #Un nombre plus eleve de fourmis permetre d'explorer davantage l'espace de recherche ,mais cela peut egalement entrainer un temps d'execution plus long.
a = float(input("Entrez la valeur d'alpha: "))# Para metre influencant l'importance de la pheromone dans le choix de la ville suivante.
b = float(input("Entrez la valeur de beta: "))# Parametre influencant l'importance de la distance dans le choix de la ville suivante.
rho = float(input("Entrez la valeur de rho: "))# Taux d'evaporation de la pheromone a chaque iteration.
Q = float(input("Entrez la valeur de Q: "))# Constante de depot de la pheromone, il est recommande que Q soit plus grande mais pas trop pour eviter la saturation.
max_iter = int(input("Entrez le nombre maximum d'iterations: "))

# Initialisation des pheromones
pheromones = np.ones((nb_villes, nb_villes))
```

```
29
30 # Fonction d'evaluation du parcours
31 def eval_parcours(parcours):
32     distance_parcours = 0
33     for i in range(nb_villes-1):
34         ville1 = parcours[i]
35         ville2 = parcours[i+1]
36         distance_parcours += distances[ville1][ville2]
37     ville1 = parcours[-1]
38     ville2 = parcours[0]
39     distance_parcours += distances[ville1][ville2]
40     return distance_parcours
41
42 # Boucle principale
43 meilleure_solution = None
44 meilleure_distance = float('inf')
45
46 for iter in range(max_iter):
47
48     # Construction des solutions
49     solutions = []
50     for i in range(nb_fourmis):
51
52         # Initialisation du parcours
53         parcours = [random.randint(0, nb_villes-1)]
54
55         # Construction du parcours
56         while len(parcours) < nb_villes:
57             ville_courante = parcours[-1]
58             probas = []
59             for j in range(nb_villes):
60                 if j not in parcours:
61                     pheromone = pheromones[ville_courante][j]
62                     distance = distances[ville_courante][j]
63                     proba = math.pow(pheromone, a) * math.pow(1/distance, b)
64                     probas.append(proba)
65                 else:
66                     probas.append(0)
67             probas = probas / np.sum(probas)
68             choix = np.random.choice(np.arange(nb_villes), p=probas)
69             parcours.append(choix)
70
71         # Ajout de la solution    la liste
72         solutions.append(parcours)
73
74     # Evaluation des solutions
75     evaluations = []
76     for solution in solutions:
77         evaluations.append(eval_parcours(solution))
78
79     # Mise a jour des pheromones
80     delta_pheromones = np.zeros((nb_villes, nb_villes))
81     for i in range(nb_fourmis):
82         for j in range(nb_villes-1):
83             ville1 = solutions[i][j]
84             ville2 = solutions[i][j+1]
85             #Mise a jour de la quantite de pheromones deposee par la fourmi
86             delta_pheromones[ville1][ville2] += Q/evaluations[i]
87         #Ajout de la quantite de pheromones deposee par la fourmi pour la boucle de retour
88         ville1 = solutions[i][-1]
89         ville2 = solutions[i][0]
90         delta_pheromones[ville1][ville2] += Q/evaluations[i]
91
92     #Evaporation des pheromones
93     pheromones= (1 - rho) * pheromones + delta_pheromones
94
95     # Ajout de la solution    la liste
96     solutions.append(parcours)
97
98 # Evaluation des solutions
99 evaluations = []
100 for solution in solutions:
101     evaluations.append(eval_parcours(solution))
102
103 # Recherche de la meilleure solution
104 indice_meilleure_solution = np.argmin(evaluations)
```

```
meilleure_solution = solutions[indice_meilleure_solution]
meilleure_distance = evaluations[indice_meilleure_solution]

#Ajout de la ville de depart a la fin de la liste
meilleure_solution.append(meilleure_solution[0])

 # Affichage de la meilleure solution
print("La meilleure solution trouv e est: ", meilleure_solution)
print("La distance de la meilleure solution est: ", meilleure_distance)
```

Listing 1: Algorithme de colonie de fourmis pour le probleme du voyageur de commerce

4.4 Exemple d'application:

```
Entrez le nombre de villes: 20
Entrez la valeur d'alpha: 1
Entrez la valeur de beta: 2
Entrez la valeur de rho: 0.5
Entrez la valeur de Q: 100
Entrez le nombre maximum d'itérations: 500
La meilleure solution trouvée est:  [8, 16, 9, 18, 1, 15, 12, 0, 7, 10, 6, 14, 4, 11, 13, 3, 17, 5, 19, 2, 8]
La distance de la meilleure solution est:  4.041831998243511
```

Figure 5: Exemple de l'éxecution

4.5 Remarque:

les paramètres a et b contrôlent l'influence de la phéromone et de la distance dans le choix de la prochaine ville à visiter, ont un impact significatif sur les performances de l'algorithme.
Une valeur plus élevée de 'a' favorise l'exploration des solutions, tandis qu'une valeur plus élevée de 'b' favorise l'exploitation des solutions connues, la meilleure valeur et entre '1' et '5' .
Et le rho (Taux) de dépôt élevé peut aider à renforcer la trace de phéromone sur les chemins les plus courts, ce qui peut conduire à une exploration plus efficace de l'espace de recherche et une convergence plus rapide vers la solution optimale. Cependant, un taux de dépôt trop élevé peut également entraîner une convergence prématurée vers une solution sous-optimale, tandis qu'un taux de dépôt trop faible peut ralentir la convergence de l'algorithme. Si le rho proche de 1 'Disparaisse plus lentement' ,si le rho proche de 0 'Disparaisse plus rapidement'.
De plus, la constante Q, qui contrôle la quantité de phéromone déposée sur les chemins, a également un effet important sur les performances de l'algorithme, car elle affecte la convergence et la diversité des solutions.
Les résultats ont montré que des valeurs optimales pour ces paramètres peuvent conduire à une amélioration significative de l'efficacité de l'algorithme de colonie de fourmis pour résoudre le problème du voyageur de commerce.

5 Conclusion

En conclusion, cette recherche a exploré la résolution du problème du voyageur de commerce (TSP) qui est connu par sa facilite de description et sa difficulte de résolution car il appartient à la classe des problémes NP-difficiles.
Notre rapport avait comme but d'étudier et résoudre ce problème par la méthode connue sous le nom de **colonie de fourmis** qui est l'une des plusieurs méthode approximative ce qui veut dire que le resultat obtenue est réalisable mais pas forcément optimale. Le principe de cette méthode est basé sur une population initial de fourmis qui évoluent sur un graphe représentant les villes et les distances entre elles.Cette approche peut être utilisée pour résoudre une variété de problèmes d'optimisation combinatoire, y compris le problème du voyageur de commerce (TSP), comme le démontre cette recherche, et nous avons términé notre étude par une implémentation sous le language de programmetion **PYTHON.**
Ce travail modeste que nous avons réalisé n'est qu'une petite contribution à ce qui pourrait être accompli à l'avenir pour le problème du voyageur de commerce (TSP). Nous espérons que ce travail sera examiné de manière critique et pourra être amélioré pour des recherches futures dans ce domaine.

References

[1] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena scientific Belmont, MA, 1997.

[2] Donald Davendra and Magdalena Bialic-Davendra. *Novel Trends in the Traveling Salesman Problem.* BoD–Books on Demand, 2020.

[3] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006.

[4] Joseph G Ecker, Michael Kupferschmid, et al. *Introduction to operations research.* Wiley New York, 1988.

[5] Jonathan L Gross and Jay Yellen. *Graph theory and its applications.* CRC press, 2005.

[6] John Riordan. *Introduction to combinatorial analysis.* Courier Corporation, 2012.

[7] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

[8] wikipedia. wikipedia.