جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

République Algérienne Démocratique et Populaire
Université Abderrahmane MIRA de Béjaia
Faculté des Sciences Exactes

Département de Recherche Opérationnelle

Mémoire Présenté pour L'obtention du Diplôme de Master
en Mathématiques Appliquées

Spécialité : Sciences de données et aide à la decision

Résolution hiérarchique du problème du voyageur de commerce par clustering et optimisation par colonies de fourmis

Présenté par :
Djerrah Hakim

Sous la direction de : Dr. Ghazli Kahina

Défendu le 01/07/2025, devant le jury composé de :

M[e] C. Boughani	M.C. Classe A	Président du jury	UAMB - Bejaia.
M[e] S. Kendi	M.C. Classe B	Examinateur	UAMB - Bejaia
M[r] M. Saadi	M.C. Classe A	Examinateur	UAMB - Bejaia.

Année Universitaire 2024 – 2025

Remerciements

Remerciements :
Je tiens à exprimer ma sincère reconnaissance à toutes les personnes qui ont contribué, de près ou de loin, à l'élaboration de cette mémoire.

Je remercie tout particulièrement mes parents et ma sœur pour leur soutien indéfectible, les efforts matériels et les nombreux sacrifices qu'ils ont consentis tout au long de mon parcours académique.

J'adresse mes remerciements les plus chaleureux à Madame Krimat , mon encadrante, pour son encadrement rigoureux, ses conseils avisés et son accompagnement bienveillant tout au long de ce travail. Sa disponibilité et son professionnalisme ont été essentiels à l'aboutissement de cette mémoire.

Je n'oublie pas mes amis Anis D. Ikhlef, Remini Billal, Lounis Massinissa, Sonia et Suzana, pour les échanges fructueux, l'esprit d'entraide et leur soutien constant tout au long de cette aventure.

Enfin, j'exprime toute ma gratitude à ma famille et à mes proches pour leur présence, leur patience et leurs encouragements, qui ont toujours été une source de motivation pour moi.

Table des matières

Remerciements I
Liste des figures V
Liste des algorithmes VI
Liste des tables VII

Introduction générale **1**

1 Problème du voyageur de commerce **3**
Introduction 3
1.1 Origine du TSP 4
1.2 Position du problème de voyageur de commerce 4
1.3 Formulation mathématique du TSP 5
1.4 Le TSP symétrique vs TSP asymétrique 6
1.5 Domaines d'application du TSP 6
1.6 Méthodes exactes de résolution 6
1.6.1 Algorithme de force brute 6
1.6.2 Méthode Branch-and-Bound 7
1.7 Méthodes approchées 8
1.7.1 Heuristiques 8
1.7.2 Métaheuristiques 11
1.8 Complexité du TSP 20
Conclusion 21

2 Techniques de clustering appliquées au TSP **22**
Introduction 22
2.1 Définition générale du clustering 23
2.2 Clustering partitionnel 23
2.2.1 Clustering K-Means 23
2.3 Clustering hiérarchique 25
2.3.1 Clustering hiérarchique d'agglomération 25
2.3.2 Clustering hiérarchique de division 27
2.4 Clustering basé sur la densité 29
2.4.1 DBSCAN (Density-Based Spatial Clustering of Applications with Noise) 29
2.4.2 OPTICS (Ordering Points To Identify the Clustering Structure) 32
2.5 Clustering probabiliste 35
2.5.1 Gaussian Mixture Model (GMM) 36
2.5.2 Étapes de l'algorithme GMM 37

2.6 Domaines d'application du clustering 39
2.7 Illustration des techniques de clustering sur une instance de TSP 40
2.8 Utilisation du clustering dans la résolution du TSP 47
Conclusion 49

3 ACO/Clustering pour la résolution du TSP **50**
Introduction 50
3.1 Méthodologie de résolution 50
3.1.1 Choix de l'algorithme de clustering 50
3.1.2 Intégration de l'ACO 51
3.1.3 Application de l'ACO sur chaque cluster 51
3.1.4 Stratégie de connexion des solutions locales 51
3.2 Algorithme ACO/Clustering (Pseudocode) 53
3.3 Exemple ilustratif 53
Conclusion 57

4 Implémentation et expérimentations **59**
Introduction 59
4.1 Instances testées (TSPLIB) 59
4.2 Environnement de travail 60
4.3 Choix des paramètres 61
4.3.1 Le choix des paramètres de la méthode ACO 61
4.3.2 Choix des paramètres de clustering 62
4.4 Comparaison des performances 62
Conclusion 69

Conclusion générale **70**

Résumé **70**

Table des figures

1.1 Illustration d'une transformation dans un graphe où une arête entre x_2 et x_3 est remplacée pour réduire le croisement avec les arêtes y_1, y_2 et y_3. 9
1.2 Illustration d'un mouvement 2-Opt : l'arête entre t_1 et t_2 ainsi que celle entre t_3 et t_4 sont supprimées, puis remplacées par les arêtes croisées t_1–t_4 et t_2–t_3 pour réduire la longueur totale de la tournée. 10
1.3 Organigramme de l'algorithme des colonies de fourmis (ACO) 14

2.1 Diagramme de flux de l'algorithme de clustering k-Means 24
2.2 Diagramme du lustering agglomératif 26
2.3 Clustering divisif 28
2.4 Illustration du clustering hiérarchique : à gauche, l'approche **descendante** (divise successivement un cluster global), à droite, l'approche **ascendante** (agglomératif : fusionne progressivement les objets similaires en clusters). 29
2.5 Illustration des types de points dans DBSCAN : **Central** (densité suffisante), **Frontière** (proche d'un noyau mais sans assez de voisins), et **Bruit** (isolé). Paramètres : $Eps = 1$, $MinPts = 5$. 30
2.6 Clustering DBSCAN 31
2.7 clustering OPTICS 34
2.8 Résumé des étapes de la méthode GMM 38
2.9 Visualisation des villes att48 40
2.10 Clustering k-means de `att48` 42
2.11 Clustering hiérarchique d'agglomération de `att48` 43
2.12 Clustering hiérarchique de division de att48 44
2.13 Clustering des villes ATT48 avec Clustering DBSCAN 45
2.14 Clustering des villes ATT48 avec Clustering optics 46
2.15 Clustering des villes ATT48 avec Clustering GMM 47
2.16 Principe générale de clustering + TSP 48

3.1 Schéma de la méthodologie de résolution du TSP par clustering et ACO 52
3.2 Visualisation des villes ATT48 54
3.3 Résultat du clustering GMM sur `att48.tsp` avec $k = 3$ clusters. 55
3.4 Chemins locaux optimisés par ACO dans chaque cluster 55
3.5 Chemin global 57

4.1 Le langage python et Jupyter 61
4.2 Comparaison des écarts relatifs (%) des méthodes Clustering + ACO sur différentes instances TSP 64

4.3 Évolution du temps de calcul selon la méthode de clustering utilisée sur l'ensemble des instances testées 66
4.4 Comparaison des écarts relatifs entre ACO et ACO + Clustering sur plusieurs instances du TSP 67
4.5 Évolution du temps de calcul (en secondes) pour l'approche K-means + ACO sur de grandes instances TSP 68

Liste des Algorithmes

1 Algorithme de force brute pour le TSP 7
2 Branch and Bound pour le TSP 8
3 Algorithme 3-Opt pour le TSP 9
4 Algorithme de Lin-Kernighan (LK) pour le TSP 10
5 Algorithme du Plus Proche Voisin (Nearest Neighbor) pour le TSP 11
6 Recherche Tabou pour le Problème du Voyageur de Commerce (TSP) 12
7 Algorithme général ACO 15
8 Algorithme ACO appliqué au TSP 16

9 Algorithme K-Means 25
10 Algorithme hiérarchique agglomératif 27
11 Clustering hiérarchique divisif 28
12 Algorithme DBSCAN 32
13 OPTICS (Ordering Points To Identify the Clustering Structure) 35
14 Méthode des Mélanges de Gaussiennes (GMM) avec EM 39

15 Résolution du TSP avec ACO/K-Means clustering 53

Liste des tableaux

1.1 Complexité temporelle de la recherche exhaustive pour le TSP 20

2.1 Caractéristiques des clusters obtenus par K-Means sur l'instance `att48` 41
2.2 Caractéristiques des clusters obtenus par le clustering agglomératif sur l'instance `att48` 42
2.3 Caratéristiques des clusters obtenus par Bisecting K-Means sur l'instance `att48` 43
2.4 Résumé des clusters DBSCAN obtenus sur l'instance `att48.tsp` 44
2.5 Caractéristiques des clusters obtenus par OPTICS sur `att48` 45
2.6 Résumé des caractéristiques des clusters obtenus par GMM sur l'instance `att48` 46

4.1 Paramètres de l'algorithme ACO utilisés pour la résolution du TSP 62
4.2 Caractéristiques des jeux de données TSP utilisés 63
4.3 Comparaison des performances des méthodes de clustering (GMM, K-means, DBSCAN, OPTICS) combinées à l'algorithme ACO sur différentes instances du TSP. Les valeurs indiquent la longueur totale des circuits obtenus. 63
4.4 Écarts relatifs (%) entre les longueurs obtenues par les méthodes de clustering combinées à ACO et les longueurs optimales pour chaque instance TSP 64
4.5 Comparaison des temps d'exécution (en secondes) des méthodes de clustering (GMM, K-means, DBSCAN et OPTICS) combinées à l'algorithme ACO pour la résolution de plusieurs instances TSP. 65
4.6 Comparaison des performances entre ACO sans cluster et ACO avec clustering sur plusieurs instances TSP 67
4.7 Application de ACO/K-means sur de grandes instances de TSP 68

Introduction générale

L'optimisation combinatoire occupe une place centrale dans de nombreux domaines scientifiques, industriels et technologiques. Elle s'intéresse à la recherche de la meilleure solution parmi un ensemble fini, mais extrêmement vaste, de configurations possibles, souvent sous contraintes. Les problèmes d'optimisation combinatoire sont omniprésents dans les systèmes logistiques, la gestion des réseaux, la planification industrielle ou encore la bioinformatique.

Parmi ces problèmes, le problème du voyageur de commerce (TSP – Travelling Salesman Problem) constitue l'un des cas les plus emblématiques et les plus étudiés.

Formellement, il s'agit de déterminer le plus court circuit permettant de visiter un ensemble de villes exactement une fois chacune, avant de revenir à la ville de départ. Bien que sa définition soit simple, le TSP est un problème NP-difficile, ce qui signifie qu'il n'existe aucun algorithme connu capable de le résoudre efficacement dans le cas général, lorsque la taille de l'instance devient grande.

L'importance du TSP ne se limite pas à sa dimension théorique : la modélisation de nombreux problèmes concrets, notamment la logistique des transports, la gestion de tournées de véhicules, la fabrication assistée par ordinateur, la robotique mobile, ou encore le séquençage génomique. Il constitue également un problème de référence dans l'évaluation des performances des algorithmes d'optimisation. Face à cette complexité, les méthodes exactes (telles que l'énumération exhaustive ou la méthode Branch and Bound) deviennent rapidement inexploitables dès que la taille du problème dépasse quelques dizaines de villes. Pour pallier ces limitations, les chercheurs se tournent vers des méthodes approchées, et en particulier vers les métaheuristiques.

Parmi ces dernières, l'algorithme à colonies de fourmis (ACO) est l'un des plus populaires. Inspirée du comportement collectif des fourmis dans la nature, cette approche stochastique a montré son efficacité pour approcher des solutions de très bonne qualité sur des instances de TSP de grande taille. Toutefois, l'efficacité de l'ACO peut décroître lorsque le nombre de villes devient très important.

Pour y parvenir, l'une des approches efficaces consiste à combiner le TSP avec des techniques de clustering. Le clustering permet de diviser l'ensemble des villes en sous-ensembles plus petits, appelés clusters, au sein desquels le TSP est résolu. Une fois les solutions locales obtenues, une stratégie de connexion permet de construire une solution globale. Cette stratégie améliore considérablement le temps d'exécution tout en maintenant une qualité de solution compétitive.

L'objectif de ce mémoire est d'étudier et d'adapter une approche hybride combinant des techniques de clustering avec l'algorithme ACO pour résoudre efficacement le TSP. Le TSP étant un problème NP-difficile, il devient rapidement intractable à mesure que le nombre de villes augmente. L'idée centrale est de diviser l'espace de recherche en sous-ensembles plus simples à explorer en appliquant des méthodes de clustering (K-Means, DBSCAN, GMM, etc.),

puis de résoudre localement chaque sous-TSP à l'aide d'ACO, une métaheuristique inspirée du comportement collectif des fourmis. Enfin, une stratégie de connexion des solutions locales est mise en œuvre pour construire un chemin global. Cette démarche vise à améliorer les performances en termes de qualité de solution et de temps de calcul, en comparant les résultats obtenus avec ceux des approches classiques sur des instances connues issues de la bibliothèque TSPLIB.

Ce manuscrit s'articule autour de quatre chapitres principaux :

Le premier chapitre est consacré à la définition du problème du voyageur de commerece, à ses caractéristiques, à ses champs d'application ainsi qu'à une brève déscription des principales méthodes de résolution. Une attention particulière est portée à l'optimisation par colonies de fourmis (ACO).

Le deuxième chapitre couvre les techniques classiques de clustering.

Dans le troisième chapitre, nous présentons une approche de résolution hybride combinant l'algorithme ACO et les méthodes de clustering.

Enfin, le quatrième chapitre est dédié à une étude expérimentale visant à évaluer l'impact de l'intégration du clustering dans l'algorithme ACO pour résoudre le problème du voyageur de commerce.

1

Problème du voyageur de commerce

Introduction

Le problème du voyageur de commerce, couramment appelé TSP (Travelling Salesman Problem) est l'un des problèmes d'optimisation combinatoire les plus explorés en matière de recherche opérationnelle et d'informatique. Il s'agit de trouver le trajet le plus court qui permet à un voyageur de parcourir une liste de villes une seule fois, avant de retourner à son point de départ.

Ce problème, qui est à la fois facile à définir et extrêmement complexe à résoudre lorsqu'il est de grande envergure.

Ce chapitre est focalisé sur l'analyse détaillée du TSP. Il débute par une introduction à ses racines historiques et à son importance tant théorique que pratique. Par la suite, une formulation mathématique précise du problème est présentée, accompagnée d'une différenciation entre les versions symétrique et asymétrique. Ensuite, nous exposons les techniques traditionnelles employées pour le résoudre, y compris les méthodes exactes, avant de passer en revue les méthodes approchées comme les heuristiques et métaheuristiques, parmi lesquelles figure l'algorithme des colonies de fourmis (ACO). Pour finir, on aborde la complexité informatique du problème afin de souligner les difficultés associées à sa résolution.

Sommaire

Introduction 3
1.1 Origine du TSP 4
1.2 Position du problème de voyageur de commerce 4
1.3 Formulation mathématique du TSP 5
1.4 Le TSP symétrique vs TSP asymétrique 6
1.5 Domaines d'application du TSP 6
1.6 Méthodes exactes de résolution 6

1.7 Méthodes approchées 8
1.8 Complexité du TSP 20
Conclusion 21

1.1 Origine du TSP

Né au XIX siècle des travaux précurseurs de Kirkman et Hamilton ce dernier ayant même créé un jeu pédagogique (l'Icosian Game) pour illustrer le concept, le problème du voyageur de commerce s'est imposé comme le défi ultime en optimisation combinatoire. Les années 1930 marquent un tournant avec les recherches de Karl Menger à Harvard, qui en explore les fondements topologiques. La décennie suivante voit Whitney et Flood, à Princeton, analyser sa complexité croissante, révélant comment l'ajout de villes démultiplie la difficulté. La véritable révolution survient en 1954 : Dantzig, Fulkerson et Johnson résolvent pour la première fois un cas concret de 49 villes grâce à une approche novatrice combinant programmation linéaire et méthode coupe-et-branche. Ce succès ouvre la voie à une série de records impressionnants, En 1975, 100 villes ont été résolues par (Camerini et al.), et en 1987 2392 villes par (Padberg et Rinaldi), et en 1998, un problème de 13509 villes américaines a été résolu. Le nouveau millénaire propulse le TSP dans une autre dimension. En 2001, une équipe menée par Applegate et Cook pulvérise le record avec 15,112 villes allemandes. Trois ans plus tard, la Suède devient terrain d'étude avec 24 978 villes connectées optimalement sur 72,500 km. Le sommet est atteint en 2005 avec 33,810 villes résolues. Cette progression fulgurante repose sur trois piliers : L'invention d'algorithmes hybrides combinant méthodes exactes et heuristiques Le développement de techniques avancées de réduction de graphes et L'explosion des capacités de calcul. D'un simple casse-tête mathématique, le TSP est devenu à la fois un banc d'essai pour l'optimisation discrète et un outil indispensable pour des applications concrètes allant de la logistique mondiale à la conception de circuits intégrés. Son histoire reflète l'extraordinaire synergie entre abstraction mathématique et innovation algorithmique [30].

1.2 Position du problème de voyageur de commerce

Le problème du voyageur de commerce, en anglais Traveling Salesman Problem (TSP), est l'un des problèmes d'optimisation combinatoire les plus connus et l'un des problèmes les plus étudiés en recherche opérationnelle étant donné qu'il a de multiples applications réelles comme la logistique, le transport, ... etc L'objectif consiste à trouver le chemin le plus court pour visiter les n villes une et une seule fois et on revient à la ville de départ, tout en étant généralement au minimisant la distance totale à parcourir.
Le touriste entame son parcours depuis n'importe quelle ville, visite chacune des autres villes précisément une fois, et conclut son périple en retournant à sa ville d'origine. La problématique centrale se pose donc comme suit : quel trajet choisir pour parcourir toutes les villes tout en minimisant la distance totale ?
Formellement, étant donné un graphe non orienté, simple et sans boucles item $G = (V, E)$: graphe complet, $V = \{1, 2, \ldots, n\}$ est l'ensemble des villes, le TSP consiste à trouver un cycle hamiltonien (appelé également tournée) de longueur minimale, permettant ainsi d'optimiser la distance totale parcourue [17].

1.3 Formulation mathématique du TSP

Le TSP peut se modéliser comme un problème linéaire en nombre entiers binaires. (BIP). En effet, étant données n villes, notons $C = (c_{ij})$ la matrice des coûts et x_{ij} les variables de décision définies par

$$x_{ij} = \begin{cases} 1 & \text{si le voyageur va immédiatement de la ville } i \text{ vers la ville } j, \\ 0 & \text{sinon ;} \end{cases}$$

Le problème consiste à minimiser la distance parcourue, sachant qu'on visite toutes les villes une et une seule fois.

$$\begin{cases} \text{Minimiser} & \displaystyle\sum_{i\in V}\sum_{\substack{j\in V\\ j\neq i}} c_{ij}\, x_{ij} & (1.1)\\ \text{s.c.} & \displaystyle\sum_{\substack{j\in V\\ j\neq i}} x_{ij} = 1, \quad \forall i \in V & (1.2)\\ & \displaystyle\sum_{\substack{i\in V\\ i\neq j}} x_{ij} = 1, \quad \forall j \in V & (1.3)\\ & \displaystyle\sum_{i\in S}\sum_{\substack{j\in S\\ j\neq i}} x_{ij} \leq |S| - 1, \quad \forall S \subset V,\ 2 \leq |S| \leq n-1 & (1.4)\\ & x_{ij} \in \{0,1\}, \quad \forall i, j \in V,\ i \neq j & (1.5) \end{cases}$$

Les contraintes (1.2) et (1.3) s'appellent les contraintes de degré qui assurent qu'une ville est visitée qu'une seule fois : on y arrive une et une seule fois (1.2), on en part une et une seule fois (1.3). Ces contraintes ne sont pas suffisantes pour décrire les tours, d'où la nécessité d'introduire les contraintes (1.4) appelées contraintes d'élimination des sous-tours, avec S un sous ensemble de V et S son complémentaire dans V , |S| est le cardinal de S. Enfin, les contraintes (1.5) sont les contraintes d'intégrité de variables [17].

La contrainte d'élimination des sous-tours (1-4) peut se modéliser sur plusieurs formules :
Forme cutset

$$\sum_{i\in S}\sum_{j\in V\setminus S} x_{ij} \geq 1, \quad \forall S \subset V,\ 2 \leq |S| \leq n-2 \qquad (1.6)$$

Formulation de Miller–Tucker–Zemlin (MTZ)

$$\begin{cases} 1 \leq u_i \leq n-1, \ \ \forall i \in V \setminus \{1\} \\ u_i - u_j + n x_{ij} \leq n-1, \ \ \forall i, j \in V \setminus \{1\},\ i \neq j \end{cases} \qquad (1.7)$$

Formulation par flux (Flow-based)

$$\begin{cases} f_{ij} \leq (n-1) x_{ij}, \quad \forall i, j \in V,\ i \neq j \\ \sum_{j\in V\setminus\{1\}} f_{1j} = n-1 \\ \sum_{\substack{j\in V\\ j\neq i}} f_{ji} - \sum_{\substack{j\in V\\ j\neq i}} f_{ij} = 1, \quad \forall i \in V \setminus \{1\} \end{cases} \qquad (1.8)$$

1.4 Le TSP symétrique vs TSP asymétrique

Dans le problème du voyageur de commerce , on a deux types de TSP, le TSP symétrique (sTSP) et le TSP asymétrique (aTSP). Dans le TSP symétrique le coût pour relier une ville i vers une ville j est le même que pour relier j vers i, alors que dans le TSP asymétrique , il est permis d'avoir des coûts différents : la distance $i \to j$ peut différer de la distance $j \to i$. Cela peut par exemple se produire sur une carte routière contenant des voies à sens unique ou des routes plus encombrées dans un sens que dans l'autre.

En théorie des graphes, cela se traduit par :

— **TSP symétrique** :

$$\forall i, j \in V, \quad c_{ij} = c_{ji}$$

— **TSP asymétrique** :

$$\exists (i, j) \in V \times V, \quad \text{tel que } c_{ij} \neq c_{ji}$$

Un TSP asymétrique peut être transformé en un TSP symétrique en doublant à peine la taille du problème, sans que cela ne change fondamentalement le TSP à résoudre. Dans ce qui suit, nous ne considérerons donc que le cas du TSP symétrique [29].

1.5 Domaines d'application du TSP

Le TSP trouve de nombreuses applications dans des domaines variés. En logistique et transport, il est utilisé pour planifier des tournées de livraison optimales, minimisant les coûts et les distances. Dans les télécommunications, il aide à optimiser le routage des câbles ou des fibres optiques. Le TSP est également appliqué en robotique pour planifier les parcours de nettoyage ou d'inspection. En bio-informatique, il intervient dans l'alignement de séquences génétiques et la reconstruction de cartes génomiques. Enfin, il est utilisé en fabrication assistée par ordinateur (CFAO), notamment pour optimiser les trajets d'outils de coupe ou de perçage. Grâce à sa formulation simple et sa complexité, le TSP sert également de benchmark pour tester les performances d'algorithmes d'optimisation.

1.6 Méthodes exactes de résolution

Le TSP étant un problème NP-difficile, les méthodes exactes garantissent la solution optimale, mais elles sont souvent coûteuses en temps de calcul[14].

1.6.1 Algorithme de force brute

La méthode de force brute est une approche exacte pour la résolution d'un problème du voyageur de commerce (TSP), consistant à générer toutes les solutions possibles du problème et à sélectionner la meilleure. Cependant, cette méthode a une complexité de calcul en temps extrêmement élevée. Le nombre de circuits hamiltoniens possibles dans un graphe complet de

n villes est donné par $\frac{(n-1)!}{2}$[7].

Algorithme 1 : Algorithme de force brute pour le TSP

Entrées : Graphe $G = (V, E)$ avec n villes et une matrice de distances D
Output : Circuit optimal avec un coût minimal

```
1  MeilleurCircuit ← ∞;
   ;                                          // Stocke le coût minimal trouvé
2  MeilleurChemin ← ∅;
   ;                                                  // Stocke le chemin optimal
3  ForceBrute(Villes) pour chaque permutation possible de Villes faire
4  |  Calculer la distance totale du chemin;
5  |  si la distance est inférieure à MeilleurCircuit alors
6  |  |  Mettre à jour MeilleurCircuit;
7  |  |  Mettre à jour MeilleurChemin;
8  |  fin
9  fin
10 retourner MeilleurChemin, MeilleurCircuit;
11 ForceBrute{1, 2, ..., n};
   ;                              // Lancer l'algorithme avec toutes les villes
```

1.6.2 Méthode Branch-and-Bound

L'énumération exhaustive des solutions pour les problèmes combinatoires est souvent impraticable en raison de la complexité exponentielle du calcul. La méthode Branch and Bound permet d'optimiser la recherche en décomposant le problème en sous-problèmes plus simples, tout en évaluant des bornes inférieures et supérieures pour limiter l'espace de recherche. Elle repose sur la construction d'un arbre où chaque nœud représente un sous-ensemble de solutions et les branches ajoutent de nouvelles contraintes. Initialement développée pour la programmation linéaire en nombres entiers, cette méthode s'est avérée efficace pour des problèmes complexes en optimisation globale, comme la minimisation concave et la différence de fonctions convexes. La méthode consiste à diviser le problème de manière récursive en sous-problèmes tels que l'union des ensembles des solutions des racines forme l'ensemble de solution du problème d'origine. Le nœud sélectionné en priorité est celui qui présente la borne inférieure la plus basse. Un sous-ensemble non séparable est désigné comme un ensemble sondé. Pour chaque sous-problème, on calcule une borne inférieure, cela permet de prédire si le sous-problème peut attendre une solution optimale [15].

Algorithme 2 : Branch and Bound pour le TSP

Entrées : Graphe $G = (V, E)$ avec n villes et matrice de distances D
Output : Tournée optimale avec coût minimal

```
Initialiser la borne inférieure LB (ex. par réduction de matrice);
MeilleureTour ← ∞;
Placer la ville de départ dans la tournée partielle;
BranchAndBound(CheminActuel, CoutActuel, Visit) si toutes les villes ont été visitées alors
    Ajouter le coût du retour à la ville de départ;
    si CoutActuel < MeilleureTour alors
        MeilleureTour ← CoutActuel;
    fin
    retourner ;
fin
pour chaque ville i ∉ Visit faire
    Calculer le coût partiel NouveauCout;
    si NouveauCout < MeilleureTour alors
        Ajouter i à CheminActuel;
        Marquer i comme visitée;
        BranchAndBoundCheminActuel, NouveauCout, Visit;
        Retirer i de CheminActuel;
        Décocher i de Visit;
    fin
fin
BranchAndBound{0}, 0, {0};
retourner MeilleureTour
```

1.7 Méthodes approchées

Les algorithmes exacts ne présentant que des complexités impraticables, les chercheurs se sont orientés en parallèle vers la construction d'heuristiques et de métaheuristiques permettant d'obtenir de bons tours rapidement, mais sans garantie d'optimalité.

1.7.1 Heuristiques

L'origine du mot heuristique, de la langue grec, vient du verbe heuriskein qui signifie trouver. La méthode heuristique est des algorithmes qui permet de trouver dans un temps polynomial une solution réalisable, tenant en compte d'une fonction objectif, mais il garantie pas l'optimalité pour un problème d'optimisation difficile. Ce type de méthodes traduit une stratégie (une manière de penser) en s'appuyant sur la connaissance du problème. Une heuristique est spécifique au problème et ne peut pas être généralisée.

Méthode 3-opt

La méthode 3-opt est une heuristique locale, elle commence par une solution réalisable trouvée et cherche ensuite dans le voisinage d'une solution courante toute tournée améliorant la configuration courante. Dans chaque étape de l'itération, l'algorithme examine si l'échange de 3 arêtes produit une tournée plus courte. L'algorithme se poursuit tant qu'il est possible d'améliorer la solution, et s'arrête dès qu'aucune amélioration supplémentaire n'est envisageable. L'avantage de la méthode est simple à implémenter et efficace. Mais elle peut rester bloqué dans un minimum local [10].

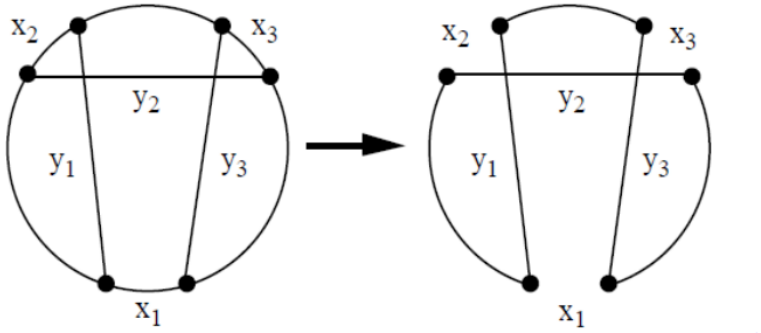

FIGURE 1.1 – Illustration d'une transformation dans un graphe où une arête entre x_2 et x_3 est remplacée pour réduire le croisement avec les arêtes y_1, y_2 et y_3.

Algorithme 3 : Algorithme 3-Opt pour le TSP

Entrées : Une tournée initiale T, une matrice de distances D
Output : Une tournée optimisée T

```
1  amlior ← vrai;
2  tant que amlior faire
3      amlior ← faux;
4      pour tous triplets d'indices (i, j, k) tels que 0 < i < j < k < n faire
5          Supprimer les 3 arêtes (i, i + 1), (j, j + 1) et (k, k + 1) de la tournée;
6          Générer les 7 réorganisations possibles (mouvements 3-Opt);
7          Calculer le coût de chaque réorganisation;
8          si l'une d'elles améliore la tournée alors
9              Appliquer la réorganisation correspondante à la tournée T;
10             amlior ← vrai;
11         fin
12     fin
13 fin
14 retourner T
```

Méthode Lin-Kernighan (LK)

L'algorithme commence avec une tournée admissible donnée, cherche ensuite dans le voisinage de la solution courante défini par l'opération k-opt move toute tournée améliorant la confi-

guration courante. C'est une généralisation simple de principe de k-opt, , dans l'algorithme, k n'est pas fixé à une valeur précise. Mais, il varie de manière croissante si l'échange de k liens produit une tournée plus courte. Jusqu'à une valeur de k qui n'améliore plus une solution déjà trouvée par la précédente. L'opération k-opt move consiste à supprimer k liens (arêtes) et à reconnecter les segments restants par de nouveaux liens, en renversant si possible le sens de parcours d'un ou de plusieurs de ces segments. Plus la valeur de λ est grande, plus la solution finale est proche de l'optimum et plus le temps d'exécution devient élevé. En général on utilise des valeurs entières de $k \in \{2, 3, 4, 5\}$. L'avantage de la méthode est-elle fournit généralement des solutions proches de l'optimum global pour le TSP symétrique. La valeur de k n'est pas fixée à l'avance mais aussi elle a une complexité élevée et une dépendance de la solution initiale [5].

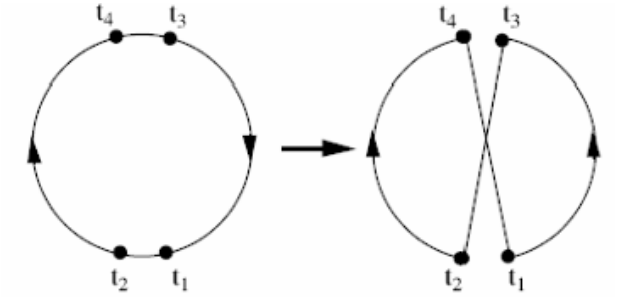

FIGURE 1.2 – Illustration d'un mouvement 2-Opt : l'arête entre t_1 et t_2 ainsi que celle entre t_3 et t_4 sont supprimées, puis remplacées par les arêtes croisées t_1–t_4 et t_2–t_3 pour réduire la longueur totale de la tournée.

Algorithme 4 : Algorithme de Lin-Kernighan (LK) pour le TSP

Entrées : Une tournée initiale T
Output : Une tournée améliorée T'

```
Initialiser T avec une solution admissible;
Définir une liste de liens candidats (ex : k plus proches voisins);
k ← 1;
répéter
jusqu'à Sélectionner une arête (x_i, y_i) à retirer;
Chercher une arête (y_i, x_{i+1}) à ajouter;
Vérifier que la tournée reste valide (pas de sous-tours);
si la nouvelle tournée est plus courte alors
    Accepter la modification et mettre à jour T;
    Augmenter k dynamiquement;
fin
sinon
    Revenir à la meilleure solution trouvée;
fin
;
retourner T'
```

La méthode du plus proche voisin

La méthode du plus proche voisin, c'est l'un des premiers algorithmes heuristiques utilisés pour déterminer une solution au problème du voyageur de commerce. Il donne rapidement la solution, mais généralement pas la solution optimale. Le principe de la methode est de choisé une ville aléatoire et de se diriger vers la ville la plus proche sans revenir a une ville déjà visitée ,jusqu'a avoir visité toutes les villes. Il faut enfin revenir à la première ville choisie, pour obtenir un cycle [22].

Algorithme 5 : Algorithme du Plus Proche Voisin (Nearest Neighbor) pour le TSP

Entrées : Un ensemble de villes V, une matrice de distances D
Output : Une tournée T

Choisir une ville de départ arbitraire v_0;
$T \leftarrow [v_0]$;
`// Initialisation de la tournée`
Marquer v_0 comme visitée;
$v_{\text{courante}} \leftarrow v_0$;
tant que *il existe des villes non visitées* **faire**
    Trouver la ville v_{min} non visitée la plus proche de v_{courante};
    Ajouter v_{min} à T;
    Marquer v_{min} comme visitée;
    $v_{\text{courante}} \leftarrow v_{\text{min}}$;
fin
Ajouter v_0 à la fin de T pour fermer la tournée;
retourner T

1.7.2 Métaheuristiques

L'origine de mots métaheuristique est composée de deux mots grecs : méta et heuristique. Le mot méta est un suffixe signifiant au-delà c'est-à-dire de niveau supérieur. Les métaheuristiques sont des méthodes généralement inspirées de la nature. Contrairement aux heuristiques, elles s'appliquent à plusieurs problèmes de nature différente. Pour cela on peut dire qu'elles sont des heuristiques modernes, de plus haut niveau, dédiées particulièrement à la résolution des problèmes d'optimisation. Leur but est d'atteindre un optimum global tout en échappant aux optimaux locaux[8].

Recherche Tabou

La recherche tabou est une méthode très populaire grâce aux succés qui elle a remportés pour résoudre de nombreux problèmes. C'est une combinaison d'une procédure de recherche locale avec d'autres règles et des mécanismes permettant à celle-ci de surmonter l'obstacle des optima locaux, tout en évitant de cycler. Le principe de la méthode est en cherche a améliorer la meilleure solution courante, et on garde en mémoire la liste des solutions précédents et celle qui se fait avec une methode de deplacement dans l'espace des solutions. En général, la méthode consiste à se déplacer d'une solution vers une autre par une observation du voisinage de la solution de départ, et à définir les transformations tabous que l'on garde en mémoire , et on s'arêtes si on trouve la solution optimale prouvée, si un nombre maximal prédéterminé a été

atteint ou un temps à ne pas dépasser ou bien si la recherche semble stagner sans amélioration de la meilleure solution trouvée [9].

Algorithme 6 : Recherche Tabou pour le Problème du Voyageur de Commerce (TSP)

Entrées : Une solution initiale S_0, taille de la liste tabou T, nombre maximal d'itérations N_{max}
Output : La meilleure solution trouvée S^*

$S^* \leftarrow S_0$;
$L_T \leftarrow \emptyset$;
$k \leftarrow 0$;
tant que $k < N_{max}$ **faire**
  Générer l'ensemble des voisins $V(S_0)$ de S_0;
  Filtrer les voisins interdits par L_T;
  Sélectionner le meilleur voisin S' parmi $V(S_0)$;
  si S' *est meilleur que* S^* **alors**
    $S^* \leftarrow S'$;
  fin
  Ajouter le mouvement de S' dans L_T;
  si *taille de* $L_T > T$ **alors**
    Supprimer le plus ancien élément de L_T;
  fin
  $S_0 \leftarrow S'$;
  $k \leftarrow k + 1$;
fin
retourner S^*

Méthode de colonies de fourmis

L'optimisation par les colonies de fourmis (Ant Colony optimization, ACO) est une approche inspirée par le comportement des colonies de fourmis réelles. Le principe de cet algorithme est que les fourmis artificielles sont définies comme des agents simples. capables de générer des solutions candidates de manière répétée. Chacune de ces solutions est considérée comme l'état du processus de construction, et on passe d'un état i à un autre j par une quantité de phéromone artificielle $\tau(i, j)$. Chaque fourmi construit une solution complète, puis met à jour les traces de phéromone sur les arcs qu'elle a empruntés. La mise à jour des phéromones favorise les bons chemins et oriente la recherche vers des solutions optimales [26].

Nous introduisons les notations et définitions suivantes :

- α : poids de l'information de phéromone, Ce paramètre contrôle l'influence de la quantité de phéromone déposée sur les arcs lors du choix du chemin par une fourmi.
- β poids de l'information heuristique (visibilité) : représente l'influence de la visibilité (souvent l'inverse de la distance) dans le processus de décision.
- ρ Taux d'évaporation des phéromones : contrôle la diminution de la phéromone au fil du temps. $0 < \rho < 1$
- Q constante de dépôt de phéromones : détermine la quantité de phéromone déposée par une fourmi sur son chemin, généralement proportionnelle à la qualité de la solution.

— m nombre de fourmis : correspond au nombre d'agents (fourmis) utilisés à chaque itération pour explorer les solutions.
— $t_{\max}$ Nombre maximal d'itérations : Définit le nombre total d'itérations que l'algorithme va effectuer.

Étapes principales de l'ACO

1. Étape 1 : Initialisation
   — On initialise le phéromone τ_{ij} pour tous les composants ou arcs du graphe de solution. :

   $$\tau_{ij}(0) = \tau_0, \quad \forall (i,j) \in E$$

   avec τ_0 une petite constante positive et E l'ensemble des arêtes du graphe.
   — On initialise l'information de visibilité η_{ij} :

   $$\eta_{ij} = \frac{1}{c_{ij}},$$

2. Étape 2 : construction des solutions (par chaque fourmi) Pour chaque fourmi k, on construit une solution complète selon l'objectif (ville , objets etc) choisi probabilistiquement selon la formule :

   $$P_{ij}^k(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^\alpha \times [\eta_{ij}]^\beta}{\sum\limits_{l \in \mathcal{N}_i^k} [\tau_{il}(t)]^\alpha \times [\eta_{il}]^\beta} & \text{si } j \in \mathcal{N}_i^k \\ 0 & \text{sinon} \end{cases}$$

   où :
   — $\tau_{ij}(t)$: intensité de phéromone.
   — η_{ij} : visibilité heuristique
   — $\mathcal{N}_i^k$ ensemble des villes que la fourmi k n'a pas encore visitées, donc qu'elle peut encore choisir comme prochaine étape depuis la ville i
3. Étape 3 : Évaluation des solutions On va évaluer les solutions fournies par chaque fourmi et on garde en mémoire la meilleure solution.
4. Étape 4 : mise à jour des phéromones

   La quantité de phéromone sur chaque composant (i, j) est réduite selon le taux d'évaporation ρ (avec $0 < \rho < 1$) :

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

   Et pour chaque fourmi k (pour $k = 1, \ldots, m$) dépose une quantité de phéromone sur les composants (i, j) qu'elle a utilisés dans sa solution. La mise à jour est donnée par :

   $$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k$$

où :

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{L_k}, & \text{si la fourmi } k \text{ a utilisé } (i, j) \\ 0, & \text{sinon} \end{cases}$$

avec :
— Q : constante de dépôt de phéromone,
— L_k : coût ou longueur de la solution construite par la fourmi k.
Répéter les étapes 2 à 4 jusqu'à atteindre un nombre maximal d'itérations $t_{\max}$, ou une convergence des solutions (absence d'amélioration du meilleur tour pendant plusieurs itérations) [26].

Le schéma suivant résume ces étapes :

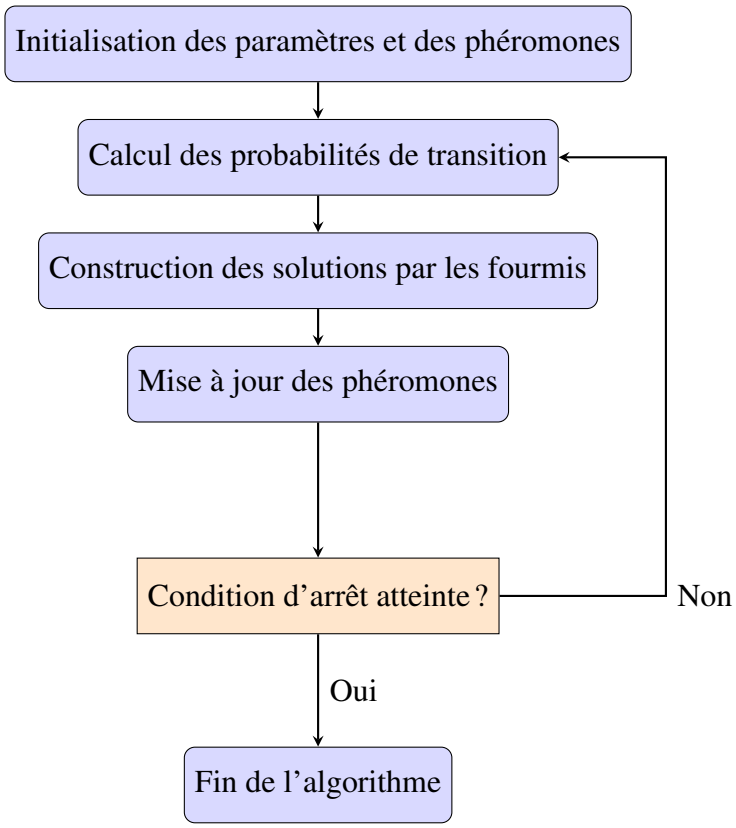

FIGURE 1.3 – Organigramme de l'algorithme des colonies de fourmis (ACO)

Algorithme 7 : Algorithme général ACO

Entrées : Paramètres α, β, ρ, Q, nombre de fourmis m, nombre d'itérations T
Output : La meilleure solution trouvée

Initialiser les niveaux de phéromones $\tau_{ij} \leftarrow \tau_0$;
pour $t \leftarrow 1$ **à** T **faire**
  pour *chaque fourmi* $k \in \{1, \ldots, m\}$ **faire**
    Construire une solution S_k selon :

$$p_{ij}^{(k)} = \frac{\tau_{ij}^{\alpha} \times \eta_{ij}^{\beta}}{\sum_{l \in N_i} \tau_{il}^{\alpha} \times \eta_{il}^{\beta}}$$

    où $\eta_{ij} = \frac{1}{d_{ij}}$ est la visibilité;
  fin
  Évaluer chaque solution S_k;
  Mettre à jour les phéromones :

$$\tau_{ij} \leftarrow (1 - \rho) \times \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{(k)}$$

  avec :

$$\Delta\tau_{ij}^{(k)} = \begin{cases} \frac{Q}{L_k} & \text{si la fourmi } k \text{ a utilisé } (i, j) \\ 0 & \text{sinon} \end{cases}$$

fin
retourner *la meilleure solution trouvée*

L'algorithme ACO est une métaheuristique utilisée pour résoudre des problèmes combinatoires. Parmi ces problèmes, on trouve le TSP. Lorsque l'ACO est appliqué à ce problème, ses composants abstraits prennent une forme concrète adaptée à la structure spécifique de ce problème. L'algorithme suivant résume ces adaptations.

Algorithme 8 : Algorithme ACO appliqué au TSP

Entrées : Graphe $G = (V, E)$, paramètres α, β, ρ, Q, nombre de fourmis m, nombre d'itérations $t_{\max}$

Output : Meilleur circuit trouvé

Initialiser les niveaux de phéromones $\tau_{ij} \leftarrow \tau_0$ pour tout $(i, j) \in E$;
Calculer la visibilité heuristique $\eta_{ij} \leftarrow \frac{1}{d_{ij}}$;
pour $t \leftarrow 1$ **à** $t_{\max}$ **faire**
    pour *chaque fourmi* $k \in \{1, ..., m\}$ **faire**
        Choisir une ville de départ aléatoire;
        tant que *le circuit de la fourmi k n'est pas complet* **faire**
            Sélectionner la prochaine ville j avec :

$$P_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha} \times [\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k} [\tau_{il}]^{\alpha} \times [\eta_{il}]^{\beta}}$$

            Ajouter j au circuit et marquer comme visitée;
        fin
        Calculer la longueur L_k du circuit de la fourmi k;
    fin
    pour *chaque arête* (i, j) **faire**
        Appliquer l'évaporation : $\tau_{ij} \leftarrow (1 - \rho) \times \tau_{ij}$;
    fin
    pour *chaque fourmi k* **faire**
        pour *chaque arête* (i, j) *dans son circuit* **faire**
            Déposer la phéromone : $\tau_{ij} \leftarrow \tau_{ij} + \frac{Q}{L_k}$;
        fin
    fin
    Mettre à jour le meilleur circuit trouvé si nécessaire;
fin
retourner *le meilleur circuit trouvé*

Exemple Illustratif de l'ACO

Considérons une instance TSP où n=5. La matrice des distances est donnée par

$$C = \begin{bmatrix} - & 2.0 & 4.0 & 3.0 & 3.6 \\ 2.0 & - & 2.5 & 3.2 & 3.3 \\ 4.0 & 2.5 & - & 2.8 & 3.0 \\ 3.0 & 3.2 & 2.8 & - & 2.7 \\ 3.6 & 3.3 & 3.0 & 2.7 & - \end{bmatrix}$$

On applique sur ce problème la méthode ACO avec un nombre d'itérations égal à 2.

On définit les paramètres suivants :

— Nombre de fourmis : 3

— $\alpha = 1, \beta = 2$

— Taux d'évaporation : $\rho = 0.5$
— Intensité du dépôt : $Q = 100$
— Phéromone initiale : $\tau_{ij} = 1$ pour tout $i \neq j$

Itération 1 Initialement, on calcule la visibilité $\eta_{ij} = \frac{1}{c_{ij}}$ et on obtient la matrice des visibilités suivante.

$$\eta = \begin{bmatrix} - & 0.5 & 0.25 & 0.333 & 0.278 \\ 0.5 & - & 0.4 & 0.3125 & 0.303 \\ 0.25 & 0.4 & - & 0.357 & 0.333 \\ 0.333 & 0.3125 & 0.357 & - & 0.37 \\ 0.278 & 0.303 & 0.333 & 0.37 & - \end{bmatrix}$$

Chaque fourmi commence par une ville aléatoire : la fourmi 1 commence par la ville 1, la fourmi 2 commence par la ville 2, la fourmi 3 commence par la ville 3. Pour chaque fourmi qui va d'une ville à une autre, on calcule à chaque étape la probabilité de transition :

$$P_{ij}^{1} = \frac{[\tau_{ij}]^{\alpha} \times [\eta_{ij}]^{\beta}}{\sum\limits_{l \in \mathcal{N}_i^k} [\tau_{il}]^{\alpha} \times [\eta_{il}]^{\beta}}$$

La fourmi 1 étant dans la ville 1, elle peut se déplacer vers les villes 2,3,4 ou 5. On calcule le poids associé à chaque ville :

Ville 2 : $\tau_{12}^{\alpha} \times \eta_{12}^{\beta} = 1 \times (0{,}5)^2 = 0{,}25$
Ville 3 : $1 \times (0{,}25)^2 = 0{,}0625$
Ville 4 : $1 \times (0{,}333)^2 \approx 0{,}1109$
Ville 5 : $1 \times (0{,}278)^2 \approx 0{,}0773$

On obtient $\sum\limits_{l \in \mathcal{N}_i^k} [\tau_{il}]^{\alpha} \times [\eta_{il}]^{\beta} = 0{,}25 + 0{,}0625 + 0{,}1109 + 0{,}0773 = 0.5007$
On obtient ainsi les probabilités suivantes :

$p_{12} \approx 0{,}25/0{,}5007 \approx 0{,}499$
$p_{13} \approx 0{,}0625/0{,}5007 \approx 0{,}125$
$p_{14} \approx 0{,}1109/0{,}5007 \approx 0{,}221$
$p_{15} \approx 0{,}0773/0{,}5007 \approx 0{,}154$

On trouve que la probabilité de la ville 2 est plus grande que les autres, alors on choisit et on ajoute la ville 2 au chemin de la fourmi 1 : $1 \rightarrow 2$.

Maintenant, de la ville 2 vers une des villes 3, 4 ou 5 :

Ville 3 : $\tau_{23}^{\alpha} \times \eta_{23}^{\beta} = 1 \times (0{,}4)^2 = 0{,}16$
Ville 4 : $1 \times (0{,}3125)^2 = 0{,}0977$
Ville 5 : $1 \times (0{,}303)^2 \approx 0{,}0918$

On a $\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^\alpha \times [\eta_{il}]^\beta = 0{,}16 + 0{,}0977 + 0{,}0918 = 0{,}3495$, On obtient ainsi les probabilités de transition suivantes :

$p_{23} \approx 0{,}458$
$p_{24} \approx 0{,}28$
$p_{25} \approx 0{,}262$

On remarque que la probabilité de la ville 3 et plus grande, donc on passe de la ville 2 vers la ville 3 et on ajoute 3 au chemin de la fourmi 1 $1 \rightarrow 2 \rightarrow 3$

De la ville 3 vers une des villes 4 ou 5 :

Ville 4 : $\tau_{34}^\alpha \times \eta_{34}^\beta = \quad 1 \times (0{,}357)^2 = 0{,}1275$
Ville 5 : $\quad 1 \times (0{,}333)^2 \approx 0{,}111$

On obtient $\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^\alpha \times [\eta_{il}]^\beta = 0{,}1275 + 0{,}111 = 0{,}2385$.
Les probabilités de transition sont :

$p_{34} \approx 0{,}1275/0{,}2385 \approx 0{,}535$
$p_{35} \approx 0{,}111/0{,}2385 \approx 0{,}465$

On remarque que la probabilité de la ville 4 est plus grande, donc on passe de la ville 3 vers la ville 4 et on ajoute 4 au chemin de la fourmi 1 : $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$. On calcule pour la ville 4 vers la seule ville restante 5.

Ville 5 : $\tau_{45}^\alpha \times \eta_{45}^\beta = \quad 1 \times (0{,}370)^2 = 0{,}137$

On obtient la probabilité : $p_{45} = 1{,}0$
On passe de la ville 4 vers la ville 5 et on ajoute 5 au chemin de la ville 1 : $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$.

On revient à la ville de départ (ville 1). La distance de ville 5 à ville 1 est 3,6.

Le chemin de la fourmi 1 : $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1$ avec un coût total : 2,0 + 2,5 + 2,8 + 2,7 + 3,6 = 13,6

De la même façon, on construit les chemins des fourmis 2,3. On obtient :
Chemin de la fourmi 2 : $2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow 2$ avec la distance tolale 2.5 + 2.8 + 2.7 + 3.6 + 2 = 13.6
Chemin de la fourmi 3 : $3 \rightarrow 2 \rightarrow 1 \rightarrow 4 \rightarrow 5 \rightarrow 3$ avec la distance totale 2.5 + 2.0 + 3.0 + 2.7 + 3.0 = 13.2

Après avoir construit les chemins de chaque fourmi on passe à l'étape de mise à jour des phéromones. On a la quantité de phéromone déposée par chaque fourmi est $\Delta\tau^k_{i,j} = \frac{Q}{L_k}$ si l'arête (i, j) est utilisée. On obtient pour chaque fourmi :

$$\text{Fourmi 1} : \Delta\tau^1_{ij} = \frac{100}{13{,}6} \approx 7{,}353$$
$$\text{Fourmi 2} : \Delta\tau^2_{ij} = \frac{100}{13{,}6} \approx 7{,}353$$
$$\text{Fourmi 3} : \Delta\tau^3_{ij} = \frac{100}{13{,}2} \approx 7{,}576$$

La contribution des fourmis sur chaque arête (i, j) est : $\Delta\tau_{ij} = \sum_k \Delta\tau^k_{ij}$. On a les arêtes utilisées dans chaque chemin :

Fourmi 1 : $(1, 2), (2, 3), (3, 4), (4, 5), (5, 1)$
Fourmi 2 : $(2, 3), (3, 4), (4, 5), (5, 1), (1, 2)$
Fourmi 3 : $(3, 2), (2, 1), (1, 4), (4, 5), (5, 3)$

On calcule $\Delta\tau_{ij}$ de chaque arête utilisée :

Arête (i,j)	Contributions	$\Delta(2)$
$(1, 2)$	Fourmis 1 et 2 : 2×7.35	$14, 7$
$(2, 3)$	Fourmis 1 et 2 : 2×7.35	$14, 7$
$(3, 4)$	Fourmis 1 et 2 : 2×7.35	$14, 7$
$(4, 5)$	Fourmis 1 et 2 et 3 : $2 \times 7.35 + 7, 58$	21.05
$(5, 1)$	Fourmis 1 et 2 : 2×7.35	$14, 7$
$(3, 2)$	Fourmis 3 : 7.58	7.58
$(2, 1)$	Fourmis 3 : 7.58	7.58
$(1, 4)$	Fourmis 3 : 7.58	7.58
$(5, 3)$	Fourmis 3 : 7.58	7.58

On applique l'évaporation : $\tau_{ij}(t + 1) \leftarrow (1 - \rho) \times \tau_{ij}(t) + \Delta\tau_{ij}$, et on met à jour la matrice des phéromones :

$$\tau(2) = \begin{bmatrix} - & 15.2 & - & 8.08 & 15,2 \\ 15.2 & - & 14.7 & - & - \\ - & 14.7 & - & 14,7 & 8.08 \\ 8.08 & - & 14.7 & - & 21.7 \\ 15.2 & - & 8.08 & 21.7 & - \end{bmatrix}$$

A la fin de l'itération 1, on garde en mémoire le chemin de la troisième fourmi :

$3 \to 2 \to 1 \to 4 \to 5 \to 3$ avec un coût égal à $13, 2.$

Itération 2 on place les fourmis : la fourmi 1 dans la ville 4, la fourmi 2 dans la ville 5, la fourmi 3 dans la ville 3

De la même façon que la premère itération. On calcule les probabilités de transition associées à chaque fourmi et on construit les chemins suivants :
Fourmi 1 : le chemin parcouru $4 \to 5 \to 3 \to 2 \to 1 \to 4$ avec la distance tolale $2,7 + 3 + 2,5 + 2 + 3 = 13,2$
Fourmi 2 : le chemin parcouru $5 \to 4 \to 1 \to 2 \to 3 \to 5$ avec la distance totale $2,7 + 3 + 2 + 2,5 + 3 = 13,2$
Fourmi 3 : le chemin parcouru $3 \to 2 \to 1 \to 4 \to 5 \to 3$,avec la distance total $2,5 + 2,0 + 3,0 + 2,7 + 3,0 = 13,2$

Après avoir construit les chemins de chaque fourmi on passe à l'étape de mise a jours des phéromones. Pour chaque arête (i, j), la quantité de phéromone déposée par la fourmi k est donnée par :

$$\Delta\tau_{ij}^{k} = \frac{Q}{L_k} = \frac{100}{13.2} \approx 7.576$$

La mise à jour totale des phéromones sur l'arête (i, j), avec un taux d'évaporation ρ, est donnée par :

$$\tau(3) = \begin{bmatrix} - & 22.76 & - & 26.78 & - \\ 22.76 & - & 22.51 & - & - \\ - & 22.51 & - & 7.35 & 19.20 \\ 26.78 & - & 7.35 & - & 26.01 \\ - & - & 19.20 & 26.01 & - \end{bmatrix}$$

À la fin de cette itération, on ne met pas à jour la solution courante, mais on ajoute ces deux nouvelles solutions, car elles ont la même longueur.

Au bout de 100 000 itérations, l'algorithme renvoie la même solution avec une longueur égale à 13,2

1.8 Complexité du TSP

Les méthodes exactes garantissent la meilleure solution, mais avec un temps d'exécution très élevé, comme l'algorithme de force brute qui a une complexité de O(n !) et l'algorithme de branch and bound, dont la complexité est moins de O(n !), mais reste une complexité exponentielle dans pire des cas.
Les méthodes approchées ne garantissent pas la solution optimale, mais avec un temps d'exécution raisonnable, comme les algorithmes de recherche locale, qui ont une complexité de $O(n^2)$ à $O(n^3)$ Ce tableau illustre un exemple du temps de calcul de l'algorithme de force brute, en supposant l'utilisation d'un logiciel capable de générer plus d'un million de parcours par seconde.

Nombre de villes n	**Nombre de tours possibles $\frac{(n-1)!}{2}$**	**Temps de calcul (converti)**
10	$\frac{9!}{2} = 181440$	0.18 s
15	$\frac{14!}{2} = 43,589,145,600$	12.1 heures
20	$\frac{19!}{2} \approx 6 \times 10^{16}$	1927 ans

Table 1.1 – Complexité temporelle de la recherche exhaustive pour le TSP

Conclusion

Dans ce chapitre on a présenté le problème du voyageur de commerce, un problème d'optimisation combinatoire de la classe NP-difficile bien connu, dont l'objectif est de trouver le plus court cycle passant une seule fois par chaque ville. Et on a présenté plusisur méthodes heuristiques et métaheuristiques. Parmi celles-ci, l'optimisation par colonies de fourmis (ACO) se distingue par son efficacité et sa capacité d'adaptation à différentes configurations de graphe. La résolution efficace du problème du voyageur de commerce (TSP) devient rapidement complexe à mesure que le nombre de villes augmente. Dans ce contexte, les techniques de clustering, ou regroupement de données, offrent une stratégie prometteuse pour réduire la complexité du problème en divisant l'espace de recherche en sous-espaces plus simples à explorer.

2

Techniques de clustering appliquées au TSP

Introduction

L'approche du clustering consiste à regrouper des objets similaires en fonction de critères précis, souvent spatiaux ou statistiques. Dans le contexte de la résolution du TSP, cette technique permet d'exploiter la structure géographique des données afin de faciliter la recherche de solutions efficaces. Nous présentons plusieurs méthodes de clustering : **K-Means**, **clustering hiérarchique**, **DBSCAN**, **GMM** et **OPTICS**, chacune illustrée par une application sur l'instance TSP `att48` de la bibliothèque TSPLIB [24].

L'objectif est d'évaluer la pertinence de ces méthodes en termes de qualité des regroupements (compacité, équilibre, forme) sur les données TSP symétrique considérées. Une brève revue des travaux existants combinant clustering et TSP est également présentés, mettant en lumière les avantages de cette approche hybride.

Sommaire

Introduction **22**
2.1 Définition générale du clustering **23**
2.2 Clustering partitionnel **23**
2.3 Clustering hiérarchique **25**
2.4 Clustering basé sur la densité **29**
2.5 Clustering probabiliste **35**
2.6 Domaines d'application du clustering **39**
2.7 Illustration des techniques de clustering sur une instance de TSP **40**
2.8 Utilisation du clustering dans la résolution du TSP **47**
Conclusion **49**

2.1 Définition générale du clustering

Le clustering regroupe un ensemble de techniques ayant pour objectif de regrouper les enregistrements d'une base de données en des groupes selon leur rapprochement les uns des autres, et ce sans connaissance préalable : il s'agit donc d'une approche d'apprentissage non supervisé. Un système de clustering prend en entrée un ensemble de données et une mesure de similarité entre ces données, et génère en sortie une partition qui reflète la structure générale de l'ensemble de données. Plus formellement, un système de clustering peut être représenté par un couple (D, s) où D représente l'ensemble de données et s la mesure de similarité, et retourne une partition $P = \{G_1, G_2, \ldots, G_m\}$ tel que les $\{G_i \mid i = 1, \ldots, m\}$ ensembles de D qui vérifient :

$$\begin{cases} G_1 \cup G_2 \cup \ldots \cup G_m = D \\ G_i \cap G_j = \emptyset \quad \text{pour } i \neq j \end{cases}$$

Chaque sous-ensemble G_i constitue un cluster, qui représente une ou plusieurs caractéristiques de l'ensemble des données D [28].

Nous décrivons dans ce qui suit les principaux types d'algorithmes de clustering.

2.2 Clustering partitionnel

C'est une méthode de clustering qui consiste à diviser un ensemble de données en k groupes (clusters) où chaque donnée appartient à un seul cluster. Les objets dans le même cluster sont similaires entre eux,les objets dans des clusters différents sont aussi différents que possible.

2.2.1 Clustering K-Means

La méthode k-means est l'une des méthodes de clustering qui permet de créeré des clusters et des centres de cluster. Cet algorithme nécessite de spécifier le nombre de groupes. Le principe général de la méthode est basé sur deux étapes. Pour chaque point de l'ensemble, on l'affecte au centre le plus proche. Et met à jour les centres des clusters. Il s'adapte bien à un grand nombre d'échantillons et a été utilisé dans de nombreux domaines d'application différents [13].

Étapes de la méthode k-means

1. Étape d'initialisation
   On fixe le nombre de clusters k. On initialise aléatoirement k centres de cluster parmi les points de l'ensemble.
2. Étape d'affectation des points
   Pour chaque point, x_i on calcule la distance entre le point x_i et tous les centres C_k des clusters.

   $$C_k^{(t)} = \left\{ x_i \mid \|x_i - \mu_k^{(t)}\|^2 \leq \|x_i - \mu_j^{(t)}\|^2, \ \forall j \in \{1, \ldots, k\} \right\}$$

   On associe chaque point au cluster le plus proche

3. Étape de mise à jour des centres
   Pour chaque cluster trouvé, on recalcule le center de cluster sous forme de la moyenne des points associés par la formule :

$$\mu_k^{(t+1)} = \frac{1}{|C_k^{(t)}|} \sum_{x_i \in C_k^{(t)}} x_i$$

4. Étape du répétition
   Répéter les étapes 2 et 3 jusqu'à ce qu'on obtienne des clusters stables ou bien que le changement soit inférieur à un seuil déjà fixé. [21]

La figure 2.1 résume les étapes principales de la méthode.

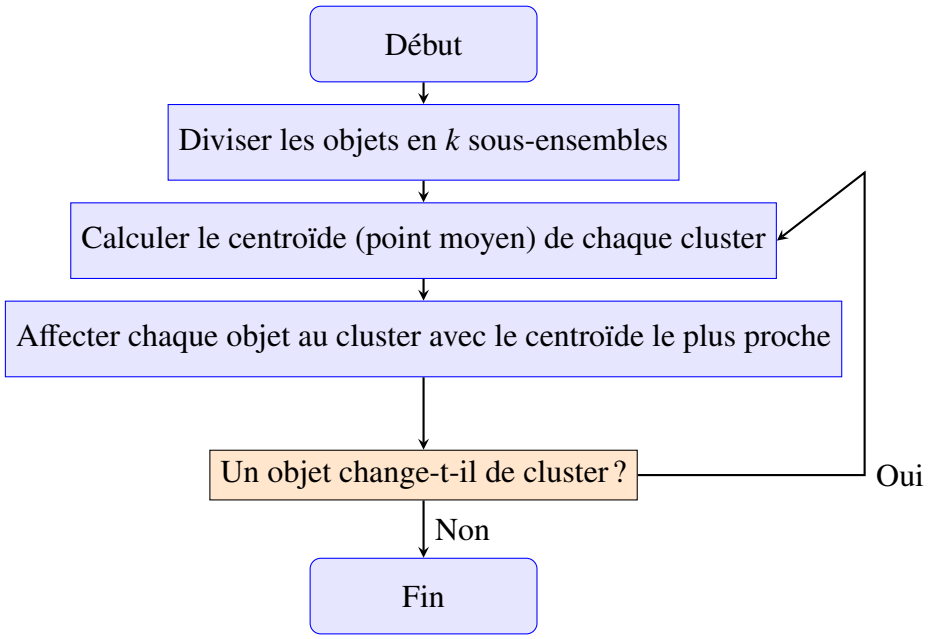

FIGURE 2.1 – Diagramme de flux de l'algorithme de clustering k-Means

L'algorithme suivant définit les étapes de la méthode.

Algorithme 9 : Algorithme K-Means

Entrées : Ensemble de données $D = \{x_1, x_2, \ldots, x_n\}$, nombre de clusters k
Output : Partitions des données en K clusters, centroïdes $\{\mu_1, \ldots, \mu_K\}$

Initialiser aléatoirement les centroïdes $\mu_1, \ldots, \mu_K$;
répéter
  pour *chaque point* $x_i \in D$ **faire**
    Calculer la distance entre x_i et chaque centroïde μ_k;
    Affecter x_i au cluster C_k tel que;

$$k = \arg\min_j \|x_i - \mu_j\|^2$$

  fin
  pour *chaque cluster* C_k **faire**
    Mettre à jour le centroïde selon;

$$\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i$$

  fin
jusqu'à *les centroïdes ne changent plus ou le nombre maximal d'itérations est atteint*;
retourner Les clusters $C_1, \ldots, C_k$ et leurs centroïdes;

2.3 Clustering hiérarchique

Les méthodes de clustering hiérarchique génèrent une succession de partitions emboîtées les unes dans les autres, au lieu de produire une seule division fixe des points.

2.3.1 Clustering hiérarchique d'agglomération

Cette méthode, elle permet de construire une hiérarchie entière des objets sous forme d'un arbre. Son principe, c'est qu'elle commence par considérer tous les points (individu) comme des clusters, et on essaye de fusionner deux ou plusieurs clusters appropriés les plus similaires sont fusionnés. Cette procédure continue jusqu'à ce que tous les points soient identifiés dans des clusters [19].

Étapes du clustering hiérarchique d'agglomération

1. Étape d'initialisation
   Considérer tous les points comme des clusters.

   Nombre initial de clusters = nombre d'individus du problème

2. Étape de calcul de la matrice de distance
   On calcule la matrice de distance avec (exemple : distance euclidienne).

3. Étape 3 : fusion des deux clusters les plus proches
   On identifie les deux clusters C_i et C_j les plus similaires selon une stratégie de liaison : liaison complète

   $$d(A, B) = \max \{d(a, b) \ : \ a \in A, \ b \in B\},$$

   et on fait la mise à jour de la matrice de distance.

   On répète cette étape jusqu'à un seul cluster contenant tous les points, ou bien un nombre de clusters fixé k [16].

Le schéma 2.2 résume ces étapes.

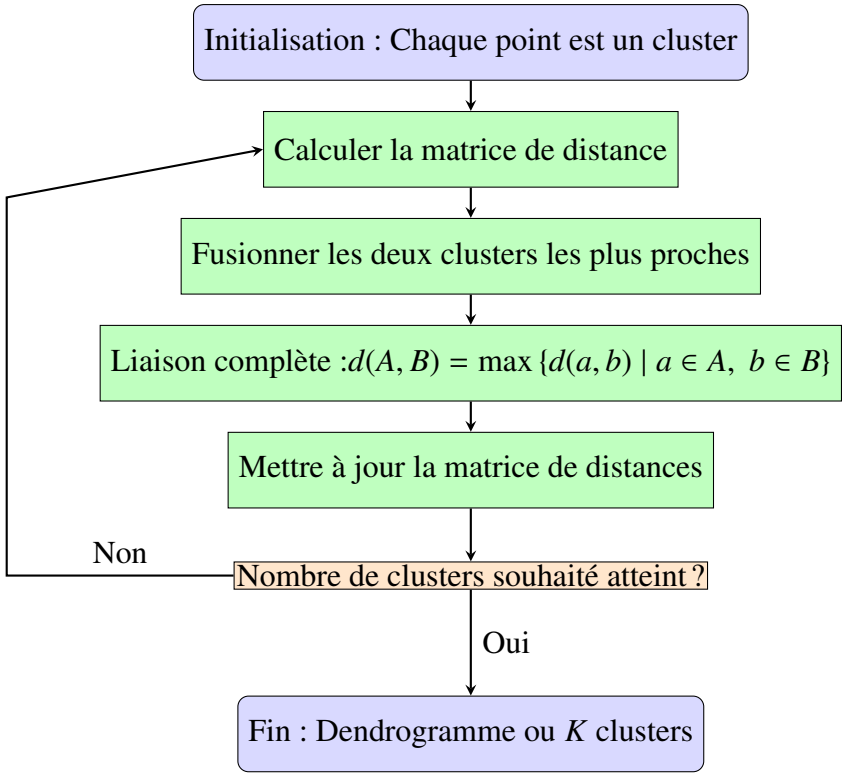

FIGURE 2.2 – Diagramme du lustering agglomératif

Algorithme 10 : Algorithme hiérarchique agglomératif

Entrées : Ensemble de points $D = \{x_1, x_2, \ldots, x_n\}$, distance $d(\cdot, \cdot)$, nombre de clusters k
Output : Partition finale en k clusters

Initialiser chaque point comme un cluster individuel : $C \leftarrow \{\{x_1\}, \{x_2\}, \ldots, \{x_n\}\}$;
Calculer la matrice des distances entre tous les clusters;
tant que $|C| > k$ **faire**
    Trouver les deux clusters $A, B \in C$ les plus proches selon une stratégie de liaison (simple, complète, moyenne, etc.);
    Fusionner A et B : $C \leftarrow (C \setminus \{A, B\}) \cup \{A \cup B\}$;
    Mettre à jour la matrice des distances;
fin
retourner C

2.3.2 Clustering hiérarchique de division

Dans cette méthode, initialement, on considère tous les points (les individus) comme un seul grand cluster, puis on divise successivement les classes en classes k plus raffinées. selon un critère d'optimisation donné, jusqu'à obtenir un ensemble de singletons, c'est-à-dire des groupes ne contenant chacun qu'un seul individu[12].

Étapes du clustrering hiérarchique de division

1. Étape d'initialisation
   On considère tous les points (les individus) comme un seul cluster.
   Le nombre initial de clusters : $s = 1$

2. Étape 2 :
   Sélection le cluster C_i a diviser, on choisit le cluster qui a la plus grande hétérogénéité par rapport à la varianc, etc.
   On applique à ce cluster une méthode de division, généralement la méthode k-means avec $k = 2$, pour qu'on obtienne deux clusters C_{i1} et C_{i2}. Et on met à jour la liste des clusters on remplace le cluster C_i par C_{i1} et C_{i2}.
   On répète cette étape jusqu'a un nombre de clusters k obtenu [31].

Le schéma 2.12 résume ces étapes.

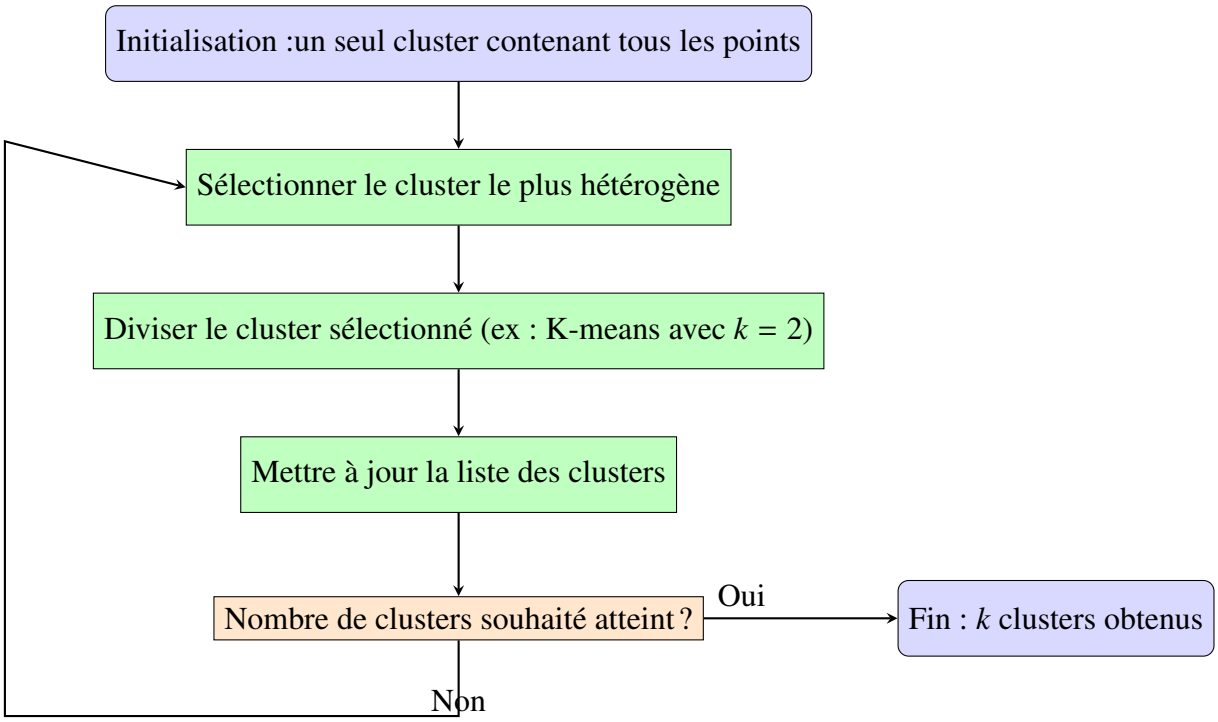

FIGURE 2.3 – Clustering divisif

Algorithme 11 : Clustering hiérarchique divisif

Entrées : Données $D = \{x_1, x_2, \ldots, x_n\}$, nombre de clusters souhaité K
Output : Partition de D en k clusters

Initialiser un seul cluster contenant tous les points : $C \leftarrow \{D\}$;
tant que *le nombre de clusters dans C est inférieur à K* **faire**
    Sélectionner le cluster $C_i \in C$ avec la plus grande hétérogénéité;
    Diviser C_i en deux sous-clusters C_i^1 et C_i^2 (par exemple avec K-means avec $K = 2$);
    Mettre à jour $C \leftarrow (C \setminus \{C_i\}) \cup \{C_i^1, C_i^2\}$;
fin
retourner C

La figure 2.4 montre la différence entre le clustering hiérarchique agglomératif et le clustering hiérarchique divisif.

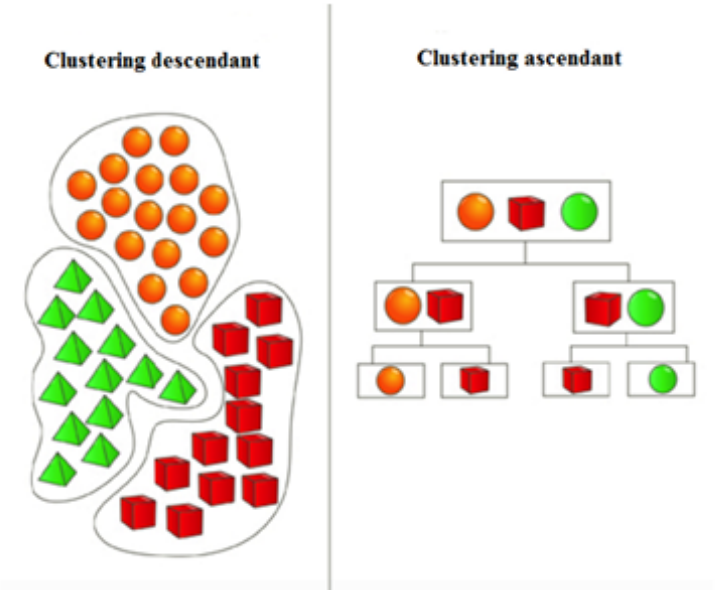

FIGURE 2.4 – Illustration du clustering hiérarchique : à gauche, l'approche **descendante** (divise successivement un cluster global), à droite, l'approche **ascendante** (agglomératif : fusionne progressivement les objets similaires en clusters).

Cette méthode permet de construire une hiérarchie de clusters (sous forme d'arbre) permettant une analyse structurelle des données.

2.4 Clustering basé sur la densité

Dans cette section, on trouve deux méthodes très connues du clustering basé sur la densité à savoir, la méthode DBSCAN et la méthode OPTICS :

2.4.1 DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

La formation d'un groupe se déroule en deux étapes. Tout d'abord, un point dense est choisi aléatoirement, Ensuite, tous les points accessibles depuis ce point, selon le critère de densité, forment le groupe. Deux paramètres spécifiés par l'utilisateur sont utilisés :

— Epsilon (Eps) qui représente le rayon de voisinage, et
— $MinPts$ qui représente le seuil de densité qui correspond au nombre minimal d'objets dans le voisinage d'un point

À ce niveau, on définit 3 types de points :

— Un point est central s'il a au moins MinPts points dans son voisinage, qui est défini par un cercle de rayon Epsilon.
— Un point est considéré comme bordure s'il a moins de MinPts points dans son voisinage, mais qu'il se trouve dans le voisinage d'un point central.
— Un point bruit s'il n'est ni central ni bordure [32].

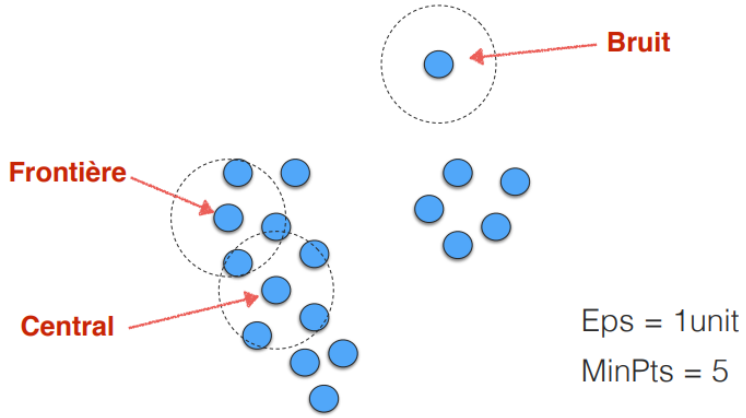

FIGURE 2.5 – Illustration des types de points dans DBSCAN : **Central** (densité suffisante), **Frontière** (proche d'un noyau mais sans assez de voisins), et **Bruit** (isolé). Paramètres : $Eps = 1$, $MinPts = 5$.

Étapes de l'algorithme

1. Initialisation.
   On marque tous les points (les individus) comme des points non encore visités, et on initialise le nombre de clusters à 0.
2. Pour tout point V_i marqué non visité, on détermine ses voisins :

   $$N(x_i) = \left\{ x_j \;\middle|\; \text{distance}(x_i, x_j) \leq \varepsilon \right\}$$

   Si $|N(x_i)| <$ MinPts on considère comme un point bruit, sinon on crée un nouveau cluster C, on ajoute le point x_i au cluster, et on ajoute tous les points voisins $N(x_i)$ du point central à une liste d'expansion.
3. Étape d'expansion du cluster.
   Tant que la liste d'expansion n'est pas vide, on prend un point x_k dans la liste. Si le point x_k est marqué comme un individu non visité On calcule son voisinage $N(x_k)$ et, si $N(x_i)$>MinPts on ajoute les points $N(x_k)$ à la liste d'expansion. Et si le point x_k n'appartient à aucun cluster, on crée un nouveau cluster [20].

Le schéma 2.6 résume les étapes de la méthode.

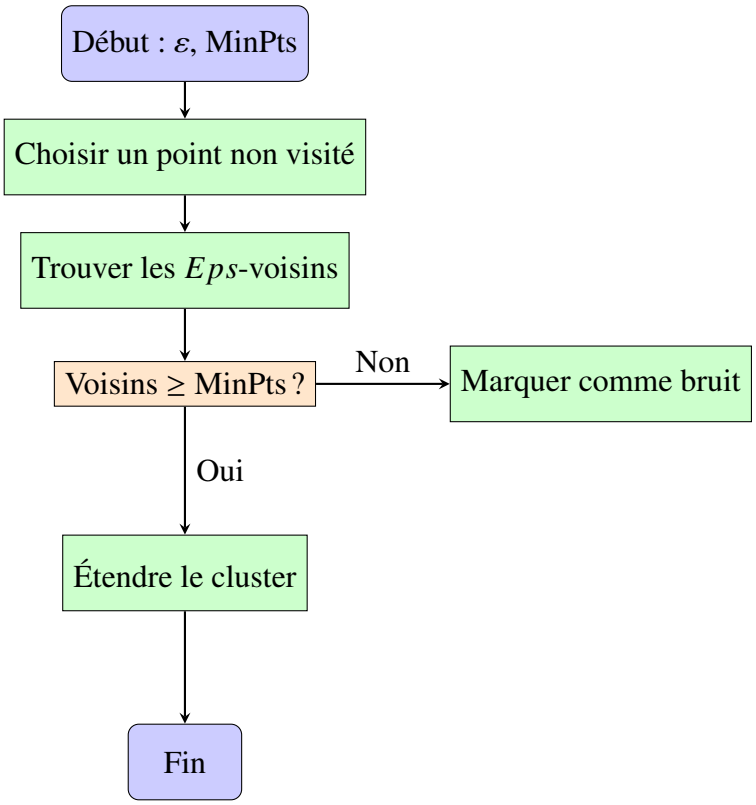

FIGURE 2.6 – Clustering DBSCAN

Algorithme 12 : Algorithme DBSCAN

Entrées : Ensemble de données D, seuil de distance Eps, nombre minimal de voisins $MinPts$

Output : Ensemble de clusters trouvés

```
Marquer tous les points comme non visités;
pour chaque point P dans D faire
    si P est déjà visité alors
        Continuer;
    fin
    Marquer P comme visité;
    Récupérer l'ensemble des voisins N de P dans un rayon Eps;
    si taille de N < MinPts alors
        Marquer P comme bruit;
    fin
    sinon
        Créer un nouveau cluster C et y ajouter P;
        pour chaque q dans N faire
            si q n'est pas visité alors
                Marquer q comme visité;
                Extraire les voisins N' de q;
                si taille de N' ≥ MinPts alors
                    N ← N ∪ N';
                fin
            fin
            si q n'est pas encore membre d'un cluster alors
                Ajouter q à C;
            fin
        fin
    fin
fin
retourner Les clusters formés
```

2.4.2 OPTICS (Ordering Points To Identify the Clustering Structure)

Comme la méthode DBSCAN, la méthode OPTICS utilise deux paramètres : Eps, le rayon maximum à examiner et $MinPts$, le nombre de points (individus) minimum. Avec ces deux paramètres, on définit une distance minimale pour obtenir un groupe de points i. On considère un point j appartenant à un groupe i s'il existe au moins $MinPts$ points dans son Eps-voisinage

Dans cette méthode, l'algorithme définit pour chaque point une distance appelée core-distance qui représente la distance entre la ville i et son $MinPts$ [3] :

$$\text{core-distsance}_{MinPts}(p) = \begin{cases} d(p, o_{MinPts}), & \text{si } |N(p)| \geq MinPts \\ \text{indéfini}, & \text{sinon} \end{cases}$$

— p : le point courant (celui que l'on analyse).

— o_{MinPts} : le $MinPts$-ième point le plus proche de p.
— $d(p, o_{MinPts})$: la distance (euclidienne ou autre) entre p et ce $MinPts$-ième voisin.
— $|N(p)|$: le nombre de points dans le voisinage de p (rayon ε autour de p).
— Si p a au moins $MinPts$ voisins, la core-distance est définie.
— Sinon, la core-distance est *indéfinie* (ce n'est pas un core point).

et une distance appelée reachability-distance qui représente la distance entre le point p et le point o.

$$\text{reachability-distance}_{\varepsilon,MinPts}(o,p) = \begin{cases} \text{Indéfini} & \text{si } |N_\varepsilon(p)| < MinPts \\ \max(\text{core-distance}_{\varepsilon,MinPts}(p), \text{distance}(p,o)) & \text{sinon} \end{cases}$$

— p : point de référence (celui à partir duquel on essaie d'atteindre un autre point).
— o : un voisin potentiel de p (celui dont on mesure l'accessibilité).
— $|N_\varepsilon(p)|$: nombre de points dans le rayon ε autour de p.
— core-distance(p) : densité locale autour de p (distance jusqu'au $MinPts$-ième voisin).
— distance(p, o) : distance directe entre p et o (appelée distance de base).
— Si p n'est pas un core point (pas assez de voisins), la reachability-distance est indéfinie.
— Sinon, c'est le maximum entre la densité locale et la distance réelle jusqu'à o.

Les étapes de la méthode

1. Étape d'initialisation
   On marque toutes les points comme des points non visitées.

2. Étape 2 : calcul du voisinage
   Pour chaque individu i on calcul Eps-voisinage. Si le nombre d'individus dans ε-voisinage est supérieur ou égal à MinPts : on considère x_i comme un point central, et on calcule sa core-distance avec $MinPts$, si non, on définit x_i comme non central. Et pour chaque point central x_i on extrait ses voisins non visités, et on considère chaque individu x_j, comme un individu déjà visité. On calcule la reachability-distance entre x_i et x_j :

   $$\text{reachability-distance}(x_j \mid x_i) = \max\Big(\text{core-distance}(x_i),\ \text{distance}(x_i, x_j)\Big)$$

   on ajoute les voisins dans un fichier de priorité.

3. Étape 3 : construction de l'ordre de traitement
   on insère chaque individu dans une liste ordonnée par rapport à sa reachability-distance. avec cette liste on peut former une carte de densité ordonnée. qui est utilisée ensuite pour extraire les clusters des individus [3].

Le schéma 2.7 résume les étapes de cette méthode.

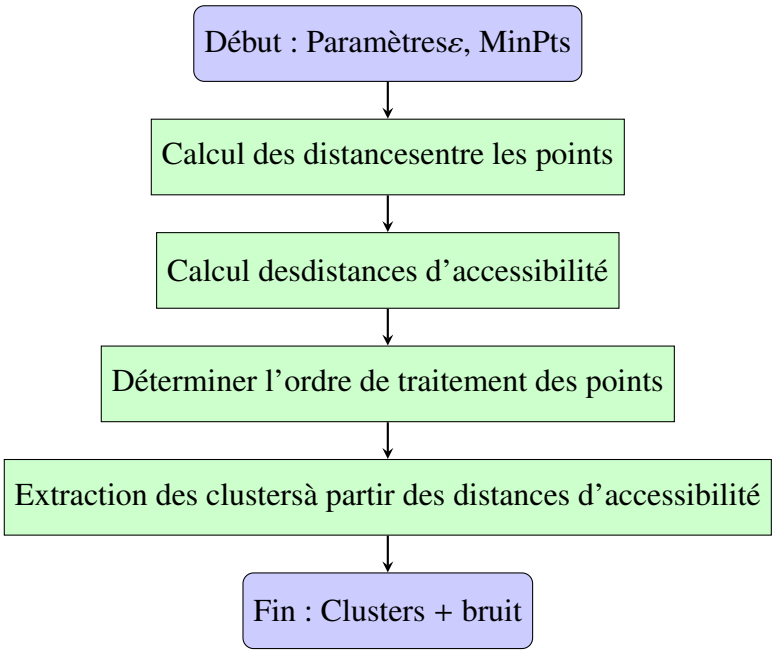

FIGURE 2.7 – clustering OPTICS

Algorithme 13 : OPTICS (Ordering Points To Identify the Clustering Structure)

Entrées : Ensemble de points D, paramètre Eps, `MinPts`
Output : Ordre des points et distances d'accessibilité

```
pour chaque p ∈ D faire
    Marquer p comme non visité;
fin
Initialiser l'ordre des points comme une liste vide;
pour chaque p ∈ D faire
    si p n'est pas visité alors
        Marquer p comme visité;
        N_p ← voisinage(p, Eps);
        Ajouter p à l'ordre des points;
        si |N_p| ≥ MinPts alors
            Calculer la distance cœur de p;
            Initialiser une file de priorité seeds;
            pour chaque q ∈ N_p faire
                si q n'est pas visité alors
                    Calculer la distance d'accessibilité de q;
                    Insérer q dans seeds avec priorité;
                fin
            fin
            tant que seeds n'est pas vide faire
                Extraire q avec la plus petite distance d'accessibilité;
                Marquer q comme visité;
                N_q ← voisinage(q, Eps);
                Ajouter q à l'ordre des points;
                si |N_q| ≥ MinPts alors
                    Calculer la distance cœur de q;
                    pour chaque o ∈ N_q faire
                        si o n'est pas visité alors
                            Calculer la distance d'accessibilité de o;
                            Insérer ou mettre à jour o dans seeds;
                        fin
                    fin
                fin
            fin
        fin
    fin
fin
retourner Ordre des points, distances d'accessibilité;
```

2.5 Clustering probabiliste

Cette méthode de clustering est basée sur les probabilités d'appartenance aux clusters.

2.5.1 Gaussian Mixture Model (GMM)

La méthode de mélange gaussien (en anglais Gaussian Mixture Model , GMM) est une méthode statistique formulée selon une densité de mélange. On utilise ce modèle lorsque on peut pas écrire les données par une seule gaussienne. Donc cela consiste à définir la moyenne, la variance et l'amplitude de chaque gaussienne. Cette méthode permet de classifier les individus en sous-classes ayant des densités spatiales ressemblant pour simplifier la resolution de problème. On optimise les paramètres du modèle par le critère de maximum de vraisemblance pour attacher le plus prossible de la distribution recherchée. La méthode GMM utilise l'algorithme Expectation-Maximization (EM) pour ajuster les paramètres des différentes gaussiennes pour mieux représenter les données. Le nombre de gaussiennes k doit être fixé au début [18].

Algorithme d'Espérance-Maximisation

L'algorithme Espérance-Maximisation (EM) est largement utilisé dans divers domaines. Il repose sur un processus itératif composé de deux étapes principales :

— Étape d'espérance (E-step) : où l'on calcule l'espérance de la vraisemblance en tenant compte des dernières variables observées.
— Une étape de maximisation (M) : cette phase vise à estimer les paramètres du modèle en maximisant la vraisemblance des paramètres trouvée à l'étape E-step.

Notations

— $D = \{x_1, x_2, \ldots, x_n\}$: ensemble des individus observés, représentés par leurs coordonnées spatiales.
— $Z = \{z_1, z_2, \ldots, z_n\}$: variables latentes associées aux individus (ex : appartenance à une composante gaussienne).
— k : nombre de clusters (ou régions) recherchés dans l'espace des individus.
— π_k : proportion de la région k, avec $\sum_{k=1}^{K} \pi_k = 1$.
— μ_k : centre (moyenne) de la région k.
— Σ_k : dispersion spatiale (matrice de covariance) des individus dans la région k.
— $\mathcal{N}(x_n \mid \mu_k, \Sigma_k)$: densité de probabilité de la ville x_n dans la région k selon une loi normale multivariée.
— γ_{nk} : probabilité que l'individu x_n appartienne à la région k (responsabilité).

Étapes de l'algorithme EM

1. Étape 1 :Initialisation Choisir des valeurs initiales pour les paramètres du modèle $\theta^{(0)}$.
2. Étape E (Expectation) :
   Calculer l'espérance de la log-vraisemblance complète, conditionnellement aux observations et aux paramètres actuels :
   $$Q(\theta \mid \theta^{(t)}) = \mathbb{E}_{Z|X,\theta^{(t)}} \left[\log p(X, Z \mid \theta)\right]$$
3. Étape M (Maximization) :
   Maximiser cette espérance pour obtenir les nouveaux paramètres :
   $$\theta^{(t+1)} = \arg\max_{\theta} Q(\theta \mid \theta^{(t)})$$
   repéter l'étape E et M jusqu'a la variation des paramètres ou de la log-vraisemblance devient négligeable.

2.5.2 Étapes de l'algorithme GMM

1. Étape 1 : Initialisation des parametres du GMM :
   On fixe le nombre des cluster k. On initialise :les centres des clusters μ_k . les matrices de covariance Σ_k, les poids du mélange π_k, tels que :

$$\sum_{k=1}^{K} \pi_k = 1 \quad \text{et} \quad \pi_k \geq 0 \quad \forall k.$$

2. Étape E (Expectation)
   Dans cette etape on calcule la responsabilité γ_{ik}, c'est-à-dire la probabilité que la individu x_i appartienne au cluster k, étant donné les paramètres actuels du modèle (π_k, μ_k, Σ_k).

   Pour chaque point x_i et chaque cluster k, on évalue :

$$\gamma_{ik} = \mathbb{P}(Z_i = k \mid x_i) = \frac{\pi_k \times \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \times \mathcal{N}(x_i \mid \mu_j, \Sigma_j)} \tag{2.1}$$

3. Étape M (Maximization)

   Lors de cette étape, on met à jour les paramètres du modèle GMM (π_k, μ_k, Σ_k) à l'aide des responsabilités γ_{ik} calculées lors de l'étape E.

   Pour chaque composante (cluster) $k = 1, \ldots, K$, on procède comme suit :
   — pondéré du cluster k :

$$N_k = \sum_{i=1}^{N} \gamma_{ik} \tag{2.2}$$

   — Mise à jour du poids (mixing coefficient) :

$$\pi_k = \frac{N_k}{N} \tag{2.3}$$

   — Mise à jour de la moyenne (centre du cluster) :

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} \times x_i \tag{2.4}$$

   — Mise à jour de la matrice de covariance :

$$\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} \times (x_i - \mu_k)(x_i - \mu_k)^\top \tag{2.5}$$

   Ces mises à jour garantissent une maximisation de la log-vraisemblance complète du modèle, conditionnellement aux responsabilités γ_{ik}.

4. Critère de convergence
   La log-vraisemblance à maximiser est donnée par :

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} \log\left(\sum_{k=1}^{K} \pi_k \times \mathcal{N}(x_i \mid \mu_k, \Sigma_k)\right) \tag{2.6}$$

   où $\theta = \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^{K}$ est l'ensemble des paramètres du modèle. Le critère d'arrêt est typiquement défini comme suit :

$$\left|\mathcal{L}^{(t)} - \mathcal{L}^{(t-1)}\right| < \varepsilon \tag{2.7}$$

   où :
   — $\mathcal{L}^{(t)}$ est la log-vraisemblance à l'itération t.
   — $\varepsilon > 0$ est un seuil de tolérance (par exemple, $\varepsilon = 10^{-4}$
5. Attribution finale des points aux clusters : Attribution floue : on conserve les responsabilités γ_{ik}. Attribution dure : chaque point est affecté à la composante k ayant la plus grande probabilité γ_{ik}.

Le schéma 2.8 résume les étapes de la méthode.

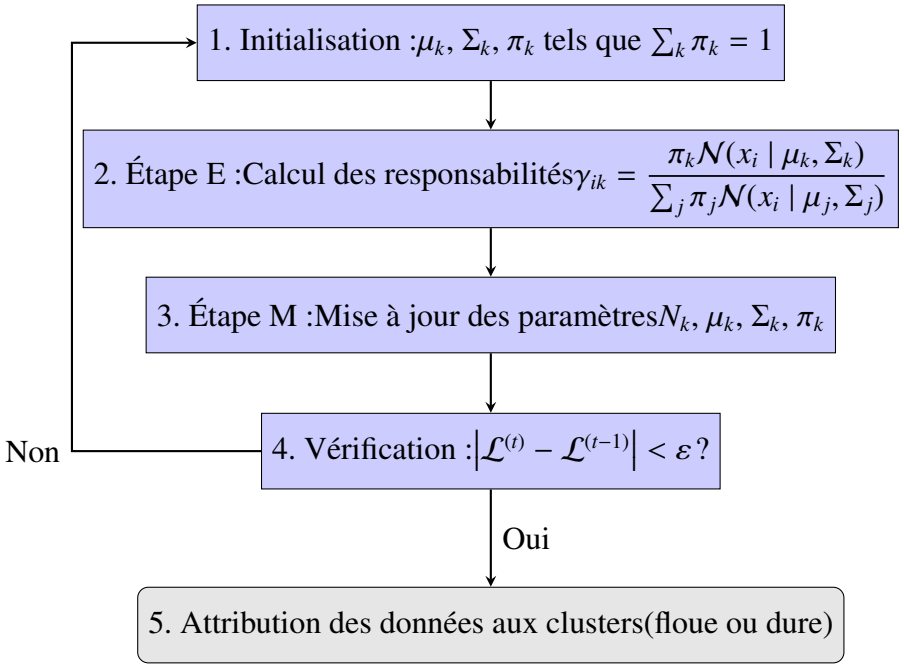

FIGURE 2.8 – Résumé des étapes de la méthode GMM

Algorithme 14 : Méthode des Mélanges de Gaussiennes (GMM) avec EM

Entrées : Données $\mathcal{D} = \{x_1, x_2, \ldots, x_n\}$, nombre de cluster K
Output : Paramètres estimés : π_k, μ_k, Σ_k pour $k = 1, \ldots, K$
/* Initialisation des paramètres */
Fixer K, initialiser aléatoirement :
μ_k (moyennes), Σ_k (matrices de covariance), π_k (poids de mélange) avec :

$$\sum_{k=1}^{K} \pi_k = 1 \quad \text{et} \quad \pi_k \geq 0 \quad \forall k$$

répéter
  /* Étape E : Calcul des responsabilités */
  pour $i = 1$ **à** n **faire**
    pour $k = 1$ **à** K **faire**
      $\gamma_{ik} \leftarrow \dfrac{\pi_k \times \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \times \mathcal{N}(x_i \mid \mu_j, \Sigma_j)}$;
    fin
  fin
  /* Étape M : Mise à jour des paramètres */
  pour $k = 1$ **à** K **faire**
    $N_k \leftarrow \sum_{i=1}^{n} \gamma_{ik}$;
    $\pi_k \leftarrow \dfrac{N_k}{n}$;
    $\mu_k \leftarrow \dfrac{1}{N_k} \sum_{i=1}^{n} \gamma_{ik} \times x_i$;
    $\Sigma_k \leftarrow \dfrac{1}{N_k} \sum_{i=1}^{n} \gamma_{ik} \times (x_i - \mu_k)(x_i - \mu_k)^\top$;
  fin
  /* Calcul de la log-vraisemblance */
  $\mathcal{L} \leftarrow \sum_{i=1}^{n} \log\left(\sum_{k=1}^{K} \pi_k \times \mathcal{N}(x_i \mid \mu_k, \Sigma_k)\right)$;
jusqu'à *convergence de la log-vraisemblance*;
/* Attribution finale des points */
pour $i = 1$ **à** n **faire**
  Affecter x_i au cluster k avec la plus grande γ_{ik} (attribution dure),
  ou bien conserver toutes les γ_{ik} (attribution floue);
fin

2.6 Domaines d'application du clustering

Le clustering est une technique d'apprentissage non supervisé largement utilisée dans de nombreux domaines pour identifier des structures cachées ou regrouper des objets similaires. En

marketing , il permet de segmenter les clients selon leurs comportements d'achat ou préférences. En biologie, il est utilisé pour classer les gènes ou les espèces selon des similarités génétiques. En vision par ordinateur, il sert à regrouper des pixels similaires pour la segmentation d'images. En cybersécurité , il peut aider à détecter des comportements anormaux ou intrusifs. D'autres applications notables incluent le traitement automatique du langage naturel (regroupement de documents ou de textes similaires), la recommandation de contenu , ou encore l' analyse de données sociales (comme la détection de communautés dans les réseaux). Le clustering est ainsi un outil polyvalent, essentiel dans les systèmes d'aide à la décision et l'exploration de données.

2.7 Illustration des techniques de clustering sur une instance de TSP

Pour illustrer les différentes techniques de clustring, nous uilisions l'instance TSP symétrique `att48` de la bibliothèque TSPLIB [24]. Cette instance comporte 48 villes, les coordonnés des villes sont représentées dans la figrure 2.9.

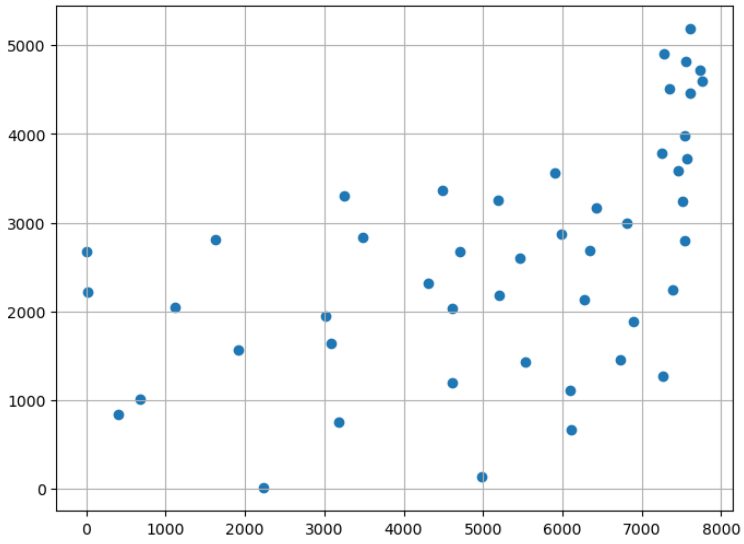

FIGURE 2.9 – Visualisation des villes att48

La matrice des distances est donnée par :

$c_{ij}=$

	1	2	3	4	$\cdots$	45	46	47	48
1	0	4507	1207	6414	$\cdots$	4639	713	1625	3871
2	4507	0	3601	1931	$\cdots$	2275	5390	3643	1196
3	1207	3601	0	5187	$\cdots$	3533	1036	2340	2495
4	6414	1931	5187	0	$\cdots$	2206	7057	5406	2681
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
45	4639	2275	3533	2206	$\cdots$	0	4776	2965	1223
46	713	5390	1036	7057	$\cdots$	4776	0	1520	3570
47	1625	3643	2340	5406	$\cdots$	2965	1520	0	2894
48	3871	1196	2495	2681	$\cdots$	1223	3570	2894	0

Ici, dans le contexte du TSP, le clustering permet de déterminer une partition (k clusters) de l'ensemble des villes. Pour l'implémentation des différentes techniques nous avons utilisé les fonctions prédéfinies de `Python` de la bibliothèque `scikit-learn`.

K-means L'application de k-means avec $k = 3$ clusters produit une partition assez équilibrée des 48 villes. Le cluster 1 regroupe 13 villes, le cluster 2 inclut 13 villes et le cluster 3 regroupe 22 villes. La variance moyenne inter-cluster est 1523523.0985 .

Les coordonnées des centroides, la taille, la variance inter-cluster de chaque cluster sont représentées dans le tableau 2.1.

Cluster	Taille	Coordonnées du centroïde	Variance inter-cluster
Cluster 1	13	(7521.38, 4176.15)	494 043.14
Cluster 2	13	(1847.23, 1819.38)	2 422 969.6450
Cluster 3	22	(5787.27, 2147.95)	1 653 556.5145

TABLE 2.1 – Caractéristiques des clusters obtenus par K-Means sur l'instance `att48`

La figure 2.10 montre le resultat obtenu.

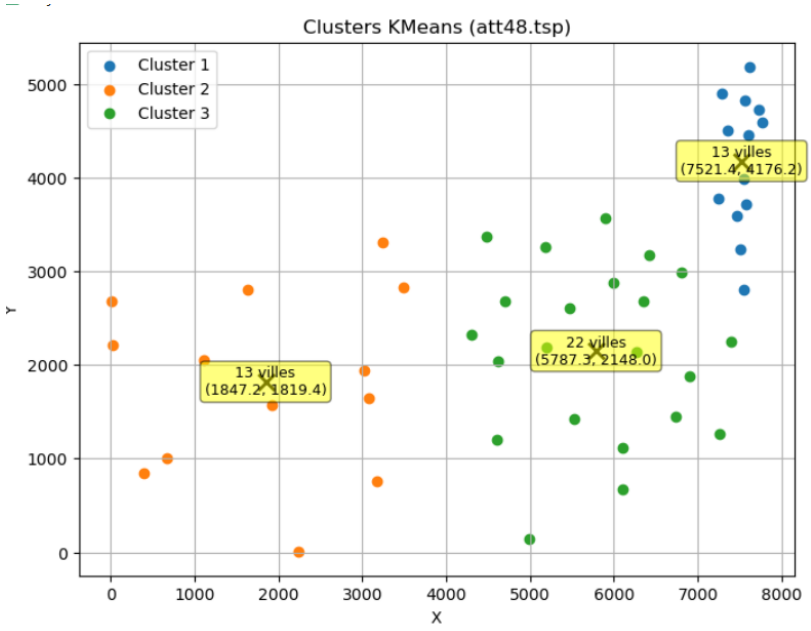

FIGURE 2.10 – Clustering k-means de `att48`

Clustering hiérarchique agglomératif Il permet de construire une structure arborescente de voisinage entre les villes. Pour visualiser les villes, nous avons appliqué k-means avec $k = 4$. On obitent une repartition qui n'est pas parfaitement équilibrée, mais reste globalement raisonnable. Le cluster 1 regroupe 13 villes , le cluster 2 regroupe 12 ville, le cluster 3 7 villes et le cluster 4 attribus 16 villes qui donne la somme des ville est 48 avec la variance inter-cluster moyenne est 119 2207.7964

Cluster	Taille	Coordonnées du centroïde	Variance inter-cluster
Cluster 1	13	(1847.23, 1819.38)	2 422 969.6450
Cluster 2	12	(5107.58, 1915.58)	1 278 326.1528
Cluster 3	7	(7557.14, 4740.57)	83 024.9388
Cluster 4	16	(6931.69, 2835.88)	984 510.4492

TABLE 2.2 – Caractéristiques des clusters obtenus par le clustering agglomératif sur l'instance `att48`

La figure 2.11 montre le resultat obtenu avec la méthode.

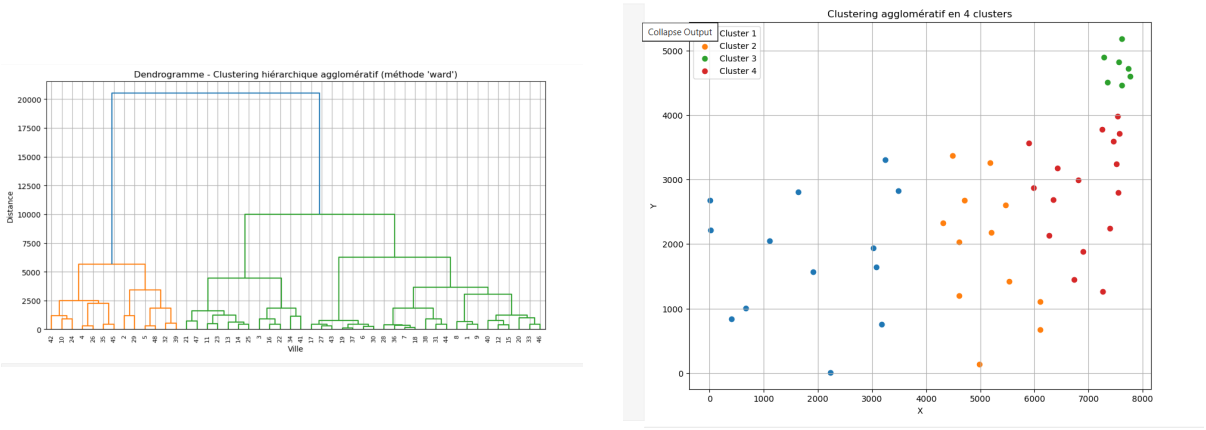

FIGURE 2.11 – Clustering hiérarchique d'agglomération de `att48`

Clustering hiérarchique divisif Pour cette méthode on pas de fonction prédéfinie que l'on peut appliquer directement. Nous avons donc implémenté l'algotithme puis appliqué avec les paramètres $max\_depth = 3$ et $min\_size = 5$ en utilisant k-means avec $k = 2$.

On obtient 8 cluster, la distribution de ces clusters est modérément déséquilibrée : 5 clusters sont proches de la moyenne (6–8 villes), mais 3 sont beaucoup plus petits. Le cluster 1 regroupe 7 villes, cluster 2 regroupe 6 villes, le cluster 3 comporte 8 villes, cluster 4 inclut 10 villes, cluster 5 6 villes, cluster 6 4 villes, cluster 7 3 villes et cluster 8 regroupe 4 villes avec une variance inter-cluster moyenne est 600 376.9870.
Les coordonnées des centroides, la taille, la variance inter-cluster de chaque cluster sont représentées dans le tableau 2.3

Cluster	Taille	Coordonnées du centroïde	Variance inter-cluster
Cluster 1	7	(7557.14, 4740.57)	83024.9388
Cluster 2	6	(7479.67, 3517.67)	165065.1111
Cluster 3	8	(5419.50, 2962.00)	567864.7500
Cluster 4	10	(6545.20, 1786.40)	781601.2000
Cluster 5	6	(3625.50, 2346.17)	694518.7222
Cluster 6	4	(3750.75, 526.00)	1452685.1875
Cluster 7	3	(1553.67, 2142.33)	371505.1111
Cluster 8	4	(277.25, 1684.75)	686750.8750

TABLE 2.3 – Caratéristiques des clusters obtenus par Bisecting K-Means sur l'instance `att48`

La figure 2.12 montre le resultat obtenu avec cette méthode.

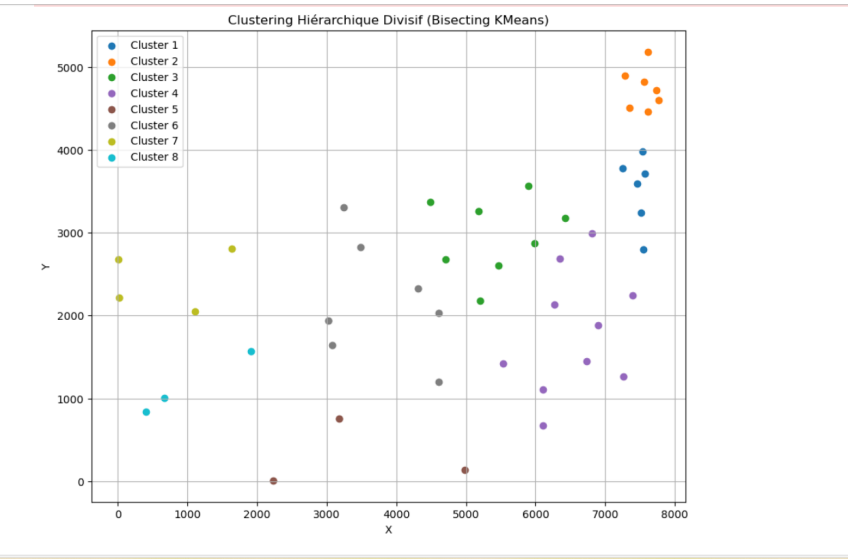

FIGURE 2.12 – Clustering hiérarchique de division de att48

DBSCAN Elle consiste à créer des clusters de forme aléatoire à partir des villes fortement connectée. Nous avons appliqué la méthode DBSCAN avec $Eps = 1000$ et $MinPts = 3$.
On obetint des clusters qui ne sont pas équilibrés en termes de nombre de villes. On a le cluster 1 regroupe 39 villes et le cluster 2 regroupes 3 villes avec 6 villes considiré comme des villes bruit avec une variance inter-cluster moyenne égale à 2 047 632.8008.
Les coordonnées des centroides, la taille, la variance inter-cluster de chaque cluster sont représentées dans le tableau 2.4

Cluster	**Taille**	**Centroïde μ_i**	**Variance inter-cluster**
Cluster 1	39	(6054.46, 2868.74)	3 723 760.4905
Cluster 2	3	(1553.67, 2142.33)	371 505.1111

TABLE 2.4 – Résumé des clusters DBSCAN obtenus sur l'instance `att48.tsp`

La figure 2.13 montre le resultat obtenu avec la méthode.

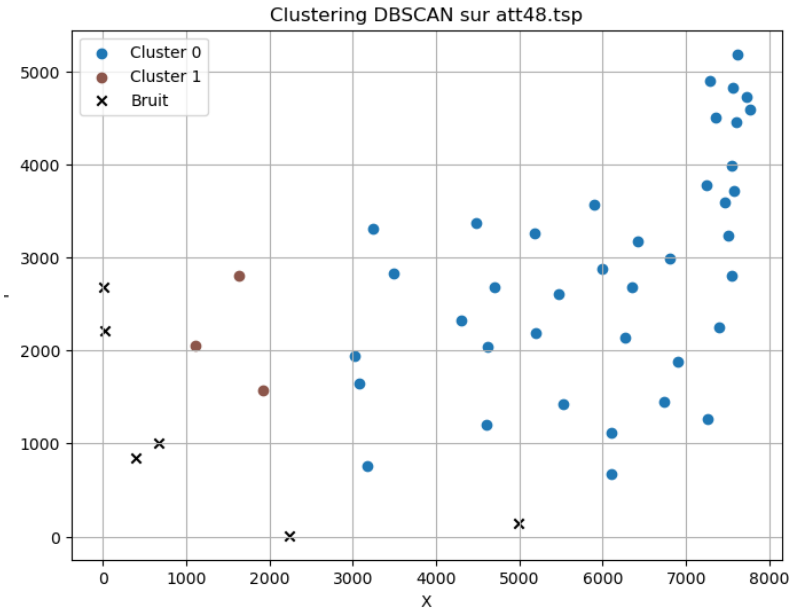

FIGURE 2.13 – Clustering des villes ATT48 avec Clustering DBSCAN

OPTICS Elle consiste à classer les villes selon leur densité locale sans exiger une séparation stricte des clusters. Elle permet d'analyser les structures à densité variable dans le TSP. L'application de OPTICS sur cette instance avec $\epsilon = 0.1$ et $MinPts = 3$ produit une partition acceptable. On a obtenu 5 cluster, le cluster 1 contient 5 villes, le cluster 2 contient 6 villes, le cluster 3 contient 4 villes, le cluster 4 7 villes et le cluster 5 regroupe 6 villes avec 20 villes comme des villes bruit et une variance inter-cluster moyenne egal a 325317.18.
Les coordonnées des centroides, la taille, la variance inter-cluster de chaque cluster sont représentées dans le tableau 2.5.

Cluster	Taille	Coordonnées du centroïde	Variance inter-cluster
Cluster 1	5	(6912.00, 1797.00)	303 548.8000
Cluster 2	6	(6156.17, 2981.50)	285 741.7222
Cluster 3	4	(7456.00, 3766.50)	36 020.7500
Cluster 4	7	(7557.14, 4740.57)	83 024.9388
Cluster 5	6	(642.33, 1932.83)	918 249.6944

TABLE 2.5 – Caractéristiques des clusters obtenus par OPTICS sur `att48`

La figure 2.14 montre le resultat obtenu avec cette méthode.

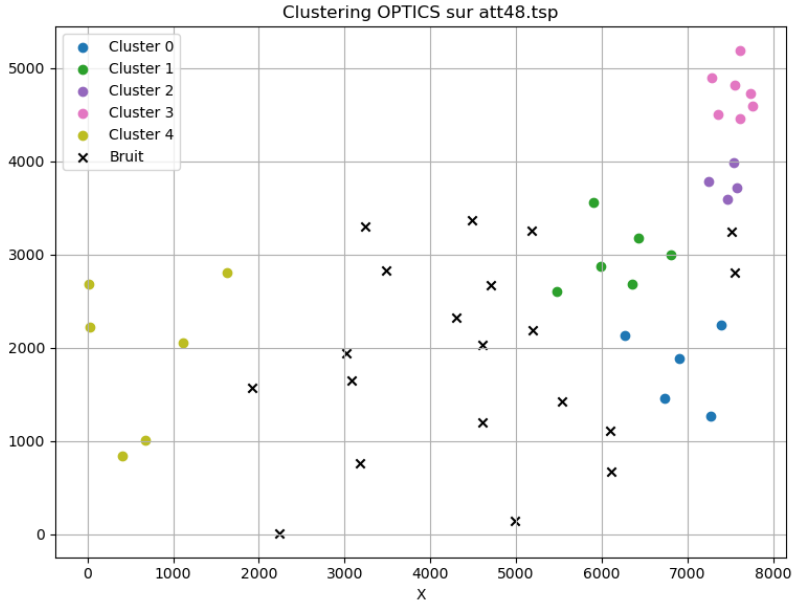

FIGURE 2.14 – Clustering des villes ATT48 avec Clustering optics

GMM Cette méthode consiste à modéliser les villes associées à des clusters probabilistes. Elle peut former des zones de distribution ellipsoïdale autour des centres. On a appliqué la méthode GMM sur cette instance avec $k = 3$. On a obtenu une répartition acceptable, le cluster 1 contient 14 villes, le cluster 2 contient 13 villes, le cluster 3 contient 21 villes, avec une variance inter-cluster moyenne egal a 1 584 630.29.
Les coordonnées des centroides, la taille, la variance inter-cluster de chaque cluster sont représentées dans le tableau 2.6.

Cluster	Taille	Coordonnées du centroïde	Variance inter-cluster
Cluster 1	14	(7515.81, 4096.65)	710 912.8028
Cluster 2	13	(1902.41, 1729.11)	2 434 163.9916
Cluster 3	21	(5670.69, 2205.26)	1 608 814.0689

TABLE 2.6 – Résumé des caractéristiques des clusters obtenus par GMM sur l'instance `att48`

La figure 2.15 montre le resultat obtenu avec cette méthode.

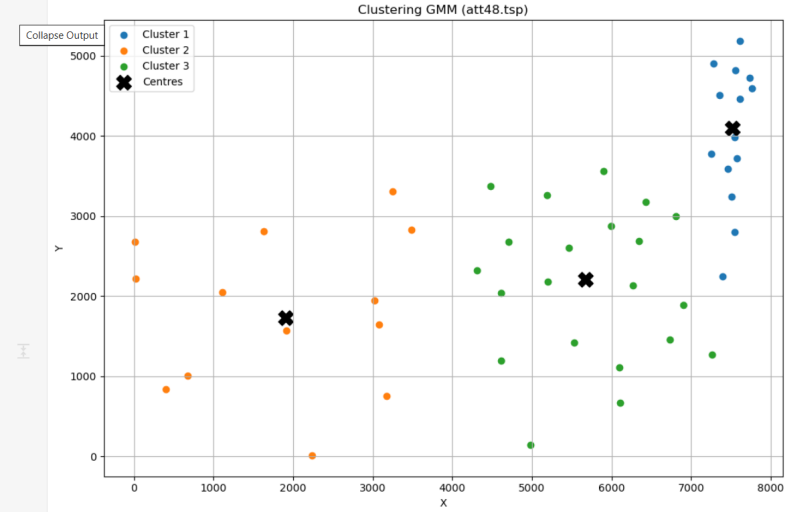

FIGURE 2.15 – Clustering des villes ATT48 avec Clustering GMM

Nous remarquons que OPTICS donne une meilleure compacité avec une variance inter-cluster = 325 317.18, mais avec une perte conséquente d'information (20 bruits). La méthode hiérarchique de division propose également une très bonne compacité (variance inter-cluster= 600 377) sans bruit, mais la taille inégale des clusters (dont certains très petits) peut être un inconvénient. Le meilleur compromis entre compacité, équilibre des tailles et absence de bruit est obtenu avec le clustering hiérarchique agglomératif, qui combine : une bonne variance inter-cluster, une répartition acceptable, aucun bruit.

2.8 Utilisation du clustering dans la résolution du TSP

L'utilisation du clustering dans la résolution du TSP est principalement motivée par le besoin de réduire la complexité algorithmique inhérente à ce problème NP-difficile. En effet, lorsque le nombre de villes devient important, la résolution exacte du TSP devient rapidement inenvisageable en raison du coût exponentiel des calculs. Le clustering permet alors de diviser l'ensemble des villes en sous-groupes géographiquement cohérents, facilitant ainsi une résolution locale du TSP dans chaque cluster. Cette décomposition réduit considérablement le temps de calcul, permet une exécution parallèle et s'adapte naturellement aux structures spatiales des problèmes réels, comme les tournées de livraison ou les réseaux urbains. De plus, cette approche améliore la robustesse des solutions heuristiques en limitant l'impact des optima locaux et facilite les mises à jour en cas d'ajout ou de modification de points. Ainsi, le clustering rend la résolution du TSP plus efficace, évolutive et applicable à grande échelle, tout en conservant une qualité de solution satisfaisante.

Pour résoudre le TSP à l'aide du clustering, une stratégie classique consiste à appliquer l'une des méthodes de clustering sur le TSP permettant de diviser le problème en sous-problèmes TSP, ensuite Plusieurs stratégies classiques sont utilisées pour combiner le clustering avec la résolution du TSP. La plus courante consiste à appliquer un algorithme de regroupement tel que K-Means, DBSCAN, OPTICS ou les modèles de mélanges gaussiens (GMM), puis à résoudre

chaque cluster localement avec une heuristique (2-opt, 3-opt) ou une métaheuristique (colonie de fourmis, recuit simulé, algorithme génétique, etc.). Ensuite, les clusters sont connectés entre eux à l'aide de diverses méthodes : en dépendant des points les plus proches entre clusters, en construisant un TSP réduit sur les centres de clusters, ou en insérant les solutions locales dans un chemin global via des techniques d'insertion ou d'agrégation. D'autres approches plus avancées utilisent des structures hiérarchiques, des partitions spatiales (comme les arbres quadtrees ou les diagrammes de Voronoï), ou encore des méthodes adaptatives où les clusters sont affinés dynamiquement selon la qualité des solutions. Ces stratégies montrent que le choix du clustering et de la méthode de fusion influence fortement sur la performance finale de l'algorithme, tant en termes de temps de calcul que de qualité du circuit obtenu.

Plusieurs travaux ont été proposés dans la littérature pour combiner efficacement le clustering avec la résolution du TSP, notamment pour traiter des instances de grande taille. Ahmed et al.[2] ont introduit une méthode fondée sur un clustering hiérarchique combiné à des métaheuristiques comme la colonie de fourmis et le recuit simulé. Leur approche consiste à diviser récursivement les villes, à résoudre localement les sous-problèmes, puis à fusionner les sous-solutions, avec de bons résultats sur les instances de la bibliothèque TSPLIB. Chen et Wu [6] ont proposé une méthode basée sur le clustering K-Means, suivie d'un algorithme génétique local et d'une stratégie d'insertion pour reconstruire le chemin global, permettant une réduction significative du temps de calcul. De leur côté, Wang et al. [27] ont utilisé OPTICS pour identifier des clusters de densité variable, puis appliqué une approche évolutionnaire pour résoudre localement chaque cluster, ce qui améliore la flexibilité et l'adaptabilité du modèle aux données complexes. Enfin, Bektas et Laporte [4] ont présenté une approche hiérarchique de décomposition spatiale, dans laquelle un TSP est d'abord résolu entre les centres des clusters, puis à l'intérieur de chaque cluster, améliorant ainsi l'efficacité de la résolution tout en maintenant une qualité de solution acceptable. Ces différentes approches montrent que l'intégration du clustering dans la résolution du TSP permet d'exploiter la structure géographique des données, de réduire la complexité algorithmique, et d'obtenir des solutions performantes pour des instances de grande taille.

Voici le schéma 2.16 illustratif du principe général.

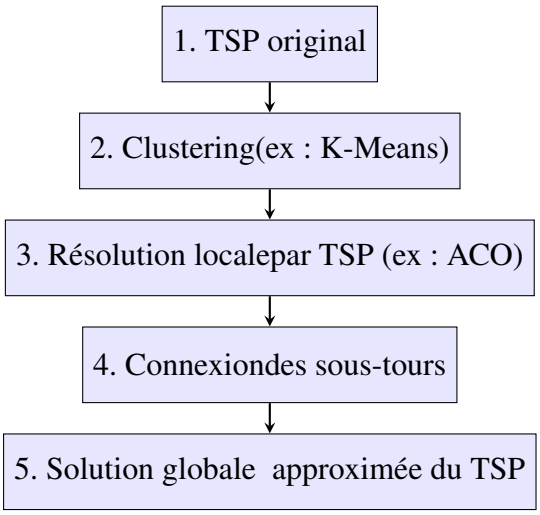

FIGURE 2.16 – Principe générale de clustering + TSP

Nous allons appliquer plusieurs méthodes de clustering (K-Means, GMM, OPTICS, DBSCAN) combinées avec l'algorithme ACO, puis utiliser une heuristique pour connecter les chemins locaux obtenus.

Conclusion

Dans ce chapitre, après avoir présenté les algorithmes classiques de clustering tels que K-Means, DBSCAN, OPTICS ou encore GMM, nous les avons appliqués à une instance réelle du TSP afin d'évaluer leur efficacité en termes de structuration des données. L'analyse a permis de mettre en évidence les forces et les limites de chaque méthode (sur l'instance TSP testée), notamment en termes de compacité, d'équilibre et forme des clusters, critères essentiels pour une bonne décomposition du problème.

3

ACO/Clustering pour la résolution du TSP

Introduction

Ce chapitre propose une méthode hybride associant le regroupement et l'algorithme ACO pour s'attaquer de manière efficace au TSP. Le concept consiste à classifier les villes en groupements grâce à des techniques telles que K-Means, DBSCAN, GMM ou OPTICS, pour ensuite gérer le problème du voyageur de commerce dans chaque segment localement avec l'ACO. On utilise d'abord les méthodes locales ou OPTICS, puis on résout le TSP localement pour chaque groupe en utilisant l'ACO. On compte par la suite les solutions locales pour constituer un circuit global. L'approche est démontrée à l'aide d'un exemple graphique sur une instance du TSP de petite taille.

Sommaire

Introduction 50
3.1 Méthodologie de résolution 50
3.2 Algorithme ACO/Clustering (Pseudocode) 53
3.3 Exemple ilustratif 53
Conclusion 57

3.1 Méthodologie de résolution

3.1.1 Choix de l'algorithme de clustering

Dans le contexte de ce projet, on a appliqué plusieurs méthodes de clustering (déjà définies dans le chapitre 2) afin d'identifier la méthode la plus efficace pour le problème du TSP. L'objectif de cette méthode est de regrouper l'ensemble des villes du TSP en sous-groupes cohérents. Les critères d'évaluation des algorithmes de clustering étaient les suivants :

— la qualité de chaque cluster (compacité, séparation),
— la capacité de détection automatique du nombre de clusters,
— La capacité de traiter des formes complexes.
— Sa correspondance avec la stratégie de résolution locale par ACO.

3.1.2 Intégration de l'ACO

Une fois l'étape de clustering (k-means, GMM, optics, ...) réalisée, on obtient des clusters, et chaque cluster est considéré comme un sous-problème du TSP. Nous appliquons la méthode de résolution des colonies de fourmis (ACO) pour chaque cluster, afin de déterminer un chemin optimal local entre les villes qui le composent.

Extraction des sous-problèmes

Chaque cluster C_i contient un ensemble de villes. Et on extrait de la matrice globale une matrice de distance locale pour chaque cluster.

3.1.3 Application de l'ACO sur chaque cluster

On exécute la méthode ACO sur chaque cluster C_i et on obtient dans chaque cluster une solution locale *Li* qui représente la longueur de chemin parcouru dans chaque cluster.
les paramètres de l'ACO sont adaptés pour la taille du problème.

Traitement des points bruités

Les méthodes DBSCAN et OPTICS produisent des villes bruites, c'est-à-dire que ces villes n'appartiennent à aucun cluster, et à ce niveau, on applique la méthode K-means sur ces bruites pour obtenir de nouveaux clusters et on applique l'ACO directement.

3.1.4 Stratégie de connexion des solutions locales

Après avoir appliqué la méthode ACO sur tous les clusters, on obtient des solutions sous forme d'un chemin , c'est-à-dire que sans revenir à la ville de départ. À ce niveau, on cherche à connecter ou bien a relié tous ces solutions pour obtenir une solution globale du problème de TSP. Alors, la question qui se pose, c'est comment on peut connecter ces solutions ?. Pour répondre à cette question on doit d'abord :

Identification des points de connexion

Pour chaque cluster, C_i on identifie les points de départ et d'arrivée du chemin local obtenus par l'ACO. Avec ces points, on peut faire une combinision entre les clusters.

Initialisation du chemin global :

Au début, le chemin global est un ensemble vide. On choisit aléatoirement un cluster de départ C_0 et on ajoutele chemin associé au chemin global.

Recherche du cluster suivant :

Pour chaque cluster C_i non encore intégré, on calcule quatre distances possibles entre les extrémités du chemin global courant et les extrémités du chemin local C_i.

— $d_{\text{finG, début}C_i}$: distance entre la fin du chemin global et le début du chemin local du cluster C_i
— $d_{\text{finG, fin}C_i}$: distance entre la fin du chemin global et la fin du chemin local du cluster C_i
— $d_{\text{débutG, début}C_i}$: distance entre le début du chemin global et le début du chemin local du cluster C_i
— $d_{\text{débutG, fin}C_i}$: distance entre le début du chemin global et la fin du chemin local du cluster C_i

Pour chaque cluster candidat, on sélectionne la plus petite parmi les 4 distances. Et parmi les clusters non intégrés, on sélectionne le cluster C_j à distance minimale.

Connexion au chemin global

On ajoute le chemin local du cluster sélectionné C_i au début ou à la fin du chemin global selon le point de connexion.
Inversé si nécessaire , pour assurer la continuité du chemin.
On répète le processus jusqu'à ce que tous les clusters soient intégrés dans le chemin global.
Une fois que tous les clusters sont intégrés dans le chemin global, on relève les deux extrémités du chemin global (début et fin). pour former un cycle hamiltonien complet.

Le schéma suivant résume les étapes de l'algorithme de résulistion de problème du TSP avec le clustering et la méthode ACO

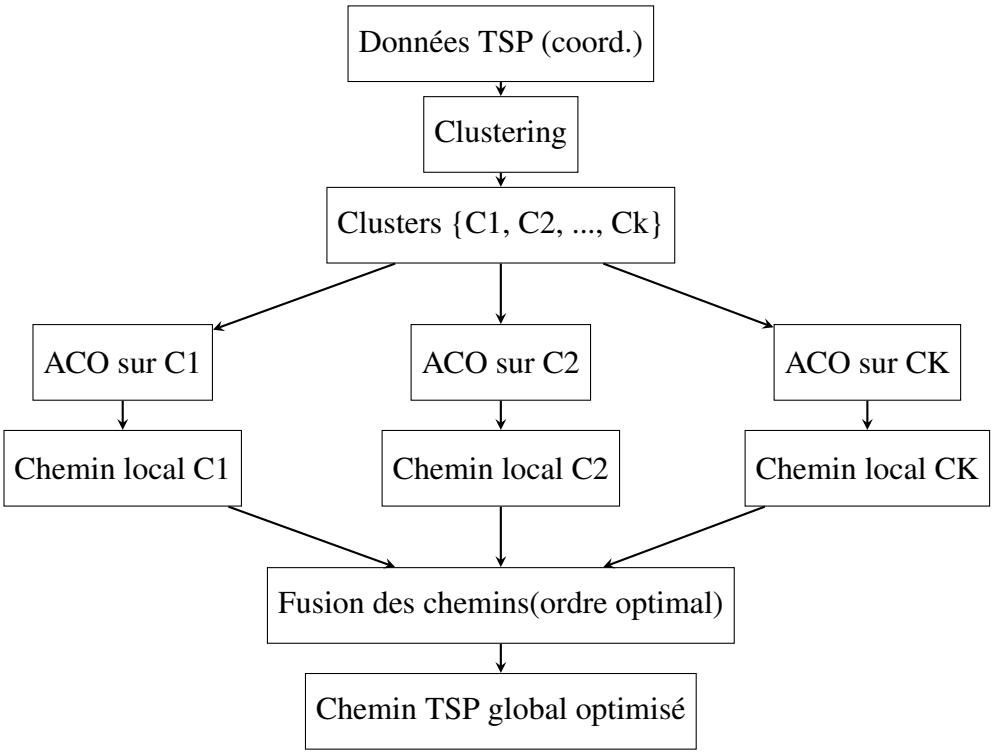

FIGURE 3.1 – Schéma de la méthodologie de résolution du TSP par clustering et ACO

3.2 Algorithme ACO/Clustering (Pseudocode)

Algorithme 15 : Résolution du TSP avec ACO/K-Means clustering

Entrées : Coordonnées des villes (fichier TSP), nombre de clusters n
Output : Chemin optimal approximatif, coût total, temps d'exécution

```
Charger les coordonnées des villes;
Appliquer K-Means avec n clusters;
pour chaque cluster faire
    Récupérer les indices des villes;
    si le cluster contient plus d'une ville alors
        Initialiser les paramètres ACO : α, β, ρ, Q, nombre de fourmis, itérations;
        pour chaque itération faire
            pour chaque fourmi faire
                Construire un chemin selon les probabilités de transition;
                Calculer sa longueur;
            fin
            Mettre à jour les phéromones;
        fin
        Conserver le meilleur chemin local;
    fin
fin
Calculer les distances entre extrémités des clusters;
pour chaque paire (i, j) de clusters distincts faire
    Calculer les quatre distances possibles : s_i → s_j, s_i → e_j, etc.;
    Conserver la distance minimale et son orientation;
fin
Construire l'ordre de visite des clusters (plus proche voisin);
Initialiser le chemin global avec le premier cluster;
pour chaque cluster suivant dans l'ordre faire
    Ajuster le sens selon l'orientation;
    Ajouter les villes dans le bon ordre au chemin global;
fin
Fermer le cycle (retour à la ville de départ);
Calculer la longueur totale;
Afficher la solution, sa validité et le temps de calcul;
```

3.3 Exemple ilustratif

Application de l'ACO avec Gaussian Mixture Model (GMM) sur le TSP symétrique

Pour illustrer les étapes principales de Algorithme 15, nous utilisons l'instance TSP symétrique `att48` décrite dans la section 2.7.

Rappelons la matrice des coûts :

$c_{ij}=$

	1	2	3	4	⋯	45	46	47	48
1	0	4507	1207	6414	⋯	4639	713	1625	3871
2	4507	0	3601	1931	⋯	2275	5390	3643	1196
3	1207	3601	0	5187	⋯	3533	1036	2340	2495
4	6414	1931	5187	0	⋯	2206	7057	5406	2681
⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮
45	4639	2275	3533	2206	⋯	0	4776	2965	1223
46	713	5390	1036	7057	⋯	4776	0	1520	3570
47	1625	3643	2340	5406	⋯	2965	1520	0	2894
48	3871	1196	2495	2681	⋯	1223	3570	2894	0

Les coordonnées des villes :

Coordonnée	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x	6734	2233	5530	401	3082	7608	7573	7265	6898	1112	5468	5989	4706	4612	6347	6107
y	1453	10	1424	841	1644	4458	3716	1268	1885	2049	2606	2873	2674	2035	2683	669

Coordonnée	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
x	7611	7462	7732	5900	4483	6101	5199	1633	4307	675	7555	7541	3177	7352	7545	3245
y	5184	3590	4723	3561	3369	1110	2182	2809	2322	1006	4819	3981	756	4506	2801	3305

Coordonnée	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
x	6426	4608	23	7248	7762	7392	3484	6271	4985	1916	7280	7509	10	6807	5185	3023
y	3173	1198	2216	3779	4595	2244	2829	2135	140	1569	4899	3239	2676	3712	3253	1942

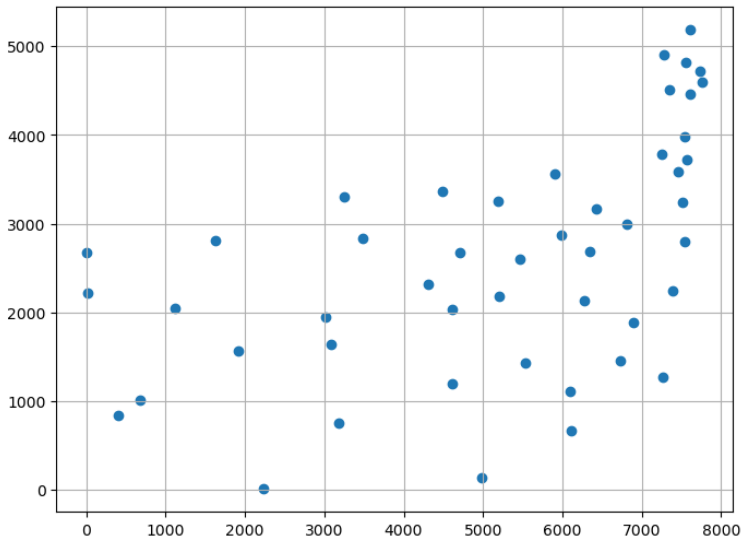

FIGURE 3.2 – Visualisation des villes ATT48

Technique de clustering utilisé La méthode GMM, implémentée via la classe `GaussianMixture` de la bibliothèque `scikit-learn`, modélise les données comme une combinaison de plusieurs

distributions gaussiennes. Chaque cluster correspond à une composante gaussienne caractérisée par sa moyenne, sa covariance et son poids.

Dans notre exemple, le nombre de composantes (clusters) est fixé à $k = 3$. Le modèle est ajusté sur les coordonnées des villes, ce qui permet d'obtenir une partition probabiliste des points, voir paragraphe GMM de la section 2.7.

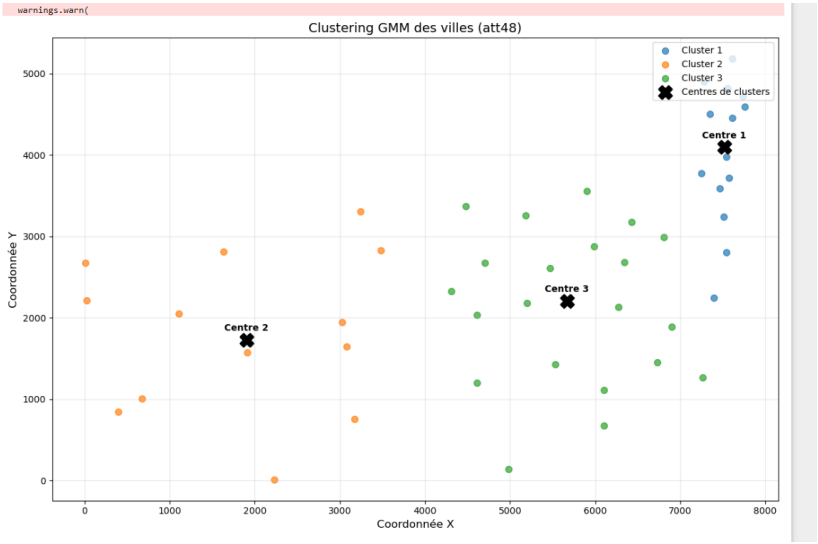

FIGURE 3.3 – Résultat du clustering GMM sur `att48.tsp` avec $k = 3$ clusters.

Les clusters obtenus par le GMM :
Cluster 0 (14 villes) : 5, 6, 16, 17, 18, 26, 27, 29, 30, 35, 36, 37, 42, 43
Cluster 1 (13 villes) : 1, 3, 4, 9, 23, 25, 28, 31, 34, 38, 41, 44, 47
Cluster 2 (21 villes) : 0, 2, 7, 8, 10, 11, 12, 13, 14, 15, 19, 20, 21, 22, 24, 32, 33, 39, 40, 45, 46

Optimisation locale avec ACO Pour chaque cluster C_i, nous appliquons l'ACO pour résoudre le sous-TSP associé ; on obtient des solutions locales. Cela permet de résoudre efficacement chaque sous-TSP.
La solution obtenue par chaque cluster :

- Cluster 0 (14 villes) :17 → 6 → 27 → 5 → 36 → 18 → 26 → 16 → 42 → 29 → 35 → 43 → 30 → 37
- Cluster 1 (13 villes) :28 → 1 → 41 → 25 → 3 → 34 → 44 → 9 → 23 → 31 → 38 → 47 → 4
- Cluster 2 (21 villes) :13 → 24 → 12 → 20 → 46 → 19 → 32 → 45 → 14 → 11 → 10 → 22 → 2 → 39 → 8 → 0 → 7 → 21 → 15 → 40 → 33

```
=== CHEMIN PAR CLUSTER ===
Cluster 0 (14 villes) :
17 -> 6 -> 27 -> 5 -> 36 -> 18 -> 26 -> 16 -> 42 -> 29 -> 35 -> 43 -> 30 -> 37
Cluster 1 (13 villes) :
28 -> 1 -> 41 -> 25 -> 3 -> 34 -> 44 -> 9 -> 23 -> 31 -> 38 -> 47 -> 4
Cluster 2 (21 villes) :
13 -> 24 -> 12 -> 20 -> 46 -> 19 -> 32 -> 45 -> 14 -> 11 -> 10 -> 22 -> 2 -> 39 -> 8 -> 0 -> 7 -> 21 -> 15 -> 40 -> 33
```

FIGURE 3.4 – Chemins locaux optimisés par ACO dans chaque cluster

Connexion des sous-chemins On selectionne le cluster 0 et on ajoute sa distance à la distance du chemin global courant.

On extrait les extermités des chemins

cluster	début	fin
Chemin global	17	37
cluster 1	28	4
cluster 2	13	33

calcule les quatre possibilités de connexion par la distance euclidienne et on trouve les résultats suivants :

Les connexions possibles entre Cluster 0 et Cluster 1

- $17 \rightarrow 28$: distance entre 17 et 28 : $\sqrt{(7462-3177)^2+(3590-756)^2} = 5137,39$
- $17 \rightarrow 4$:distance entre 17 et 4 : $\sqrt{(7462-3082)^2+(3590-4644)^2} = 5015,73$
- $37 \rightarrow 28$:distance entre 37 et 28 : $\sqrt{(7392-3177)^2+(2244-756)^2} = 4469,94$
- $37 \rightarrow 4$:distance entre 37 et 4 : $\sqrt{(7392-3089)^2+(2244-1644)^2} = 4351,56$

On remarque que la distance minimale $37 \rightarrow 4$ $(= 4351,56)$

Les connexions possibles entre cluster 0 et cluster 2

- $17 \rightarrow 13$: distance entre 17 et 13 : $\sqrt{(7462-4612)^2+(3590-2035)^2} = 3232,33$
- $17 \rightarrow 33$:distance entre 17 et 33 $\sqrt{(7462-4608)^2+(3590-1138)^2} = 3723,27$
- $37 \rightarrow 13$:distance entre 37 et 37 $\sqrt{(7392-4612)^2+(2244-2035)^2} = 2787,84$
- $37 \rightarrow 33$:distance entre 37 et 33 $\sqrt{(7392-4608)^2+(2244-1198)^2} = 2974,06$

On remarque que la distance minimale $37 \rightarrow 13$ $(= 2787,84)$

La distance $37 \rightarrow 13$ < la distance $37 \rightarrow 4$, on relie donc la ville 37 à la ville 13 et on ajoute le chemin du cluster 2 au chemin global, et on obtient :
$17 \rightarrow 6 \rightarrow 27 \rightarrow 5 \rightarrow 36 \rightarrow 18 \rightarrow 26 \rightarrow 16 \rightarrow 42 \rightarrow 29 \rightarrow 35 \rightarrow 43 \rightarrow 30 \rightarrow 37 \rightarrow 13 \rightarrow 24 \rightarrow 12 \rightarrow 20 \rightarrow 46 \rightarrow 19 \rightarrow 32 \rightarrow 45 \rightarrow 14 \rightarrow 11 \rightarrow 10 \rightarrow 22 \rightarrow 2 \rightarrow 39 \rightarrow 8 \rightarrow 0 \rightarrow 7 \rightarrow 21 \rightarrow 15 \rightarrow 40 \rightarrow 33$

On met à jours les extermités des chemins :

cluster	début	fin
Chemin global	17	33
Cluster 1	28	4

On passe à la 2^{eme} étape pour ajouter le cluster 1 au chemin global. On calcule les 4 distances entre les extermités :

Les connexions possibles entre le chemin global et le cluster 1

— $17 \rightarrow 28$: distance entre 17 et 28 : $\sqrt{(7462-4612)^2+(3590-2035)^2} = 5137,39$
— $17 \rightarrow 4$:distance entre 17 et 4 : $\sqrt{(7462-3082)^2+(3590-1644)^2} = 5015,73$
— $33 \rightarrow 4$:distance entre 33 et 4 : $\sqrt{(4608-3082)^2+(1198-1644)^2} = 1589,84$
— $33 \rightarrow 28$:distance entre 33 et 28 : $\sqrt{(4608-3177)^2+(1198-756)^2} = 1497,71$

La distance minimale $33 \rightarrow 28$, donc on relie la ville 33 à la ville 28 et on ajoute le chemin du cluster 1 au chemin global. On obtient :

$17 \rightarrow 6 \rightarrow 27 \rightarrow 5 \rightarrow 36 \rightarrow 18 \rightarrow 26 \rightarrow 16 \rightarrow 42 \rightarrow 29 \rightarrow 35 \rightarrow 43 \rightarrow 30 \rightarrow 37 \rightarrow 13 \rightarrow 24 \rightarrow 12 \rightarrow 20 \rightarrow 46 \rightarrow 19 \rightarrow 32 \rightarrow 45 \rightarrow 14 \rightarrow 11 \rightarrow 10 \rightarrow 22 \rightarrow 2 \rightarrow 39 \rightarrow 8 \rightarrow 0 \rightarrow 7 \rightarrow 21 \rightarrow 15 \rightarrow 40 \rightarrow 33 \rightarrow 28 \rightarrow 1 \rightarrow 41 \rightarrow 25 \rightarrow 3 \rightarrow 34 \rightarrow 44 \rightarrow 9 \rightarrow 23 \rightarrow 31 \rightarrow 38 \rightarrow 47 \rightarrow 4$

On reli les deux villes isolées pour fermer le circuit $4 \rightarrow 17$et on obteint le chemin global final :

$17 \rightarrow 6 \rightarrow 27 \rightarrow 5 \rightarrow 36 \rightarrow 18 \rightarrow 26 \rightarrow 16 \rightarrow 42 \rightarrow 29 \rightarrow 35 \rightarrow 43 \rightarrow 30 \rightarrow 37 \rightarrow 13 \rightarrow 24 \rightarrow 12 \rightarrow 20 \rightarrow 46 \rightarrow 19 \rightarrow 32 \rightarrow 45 \rightarrow 14 \rightarrow 11 \rightarrow 10 \rightarrow 22 \rightarrow 2 \rightarrow 39 \rightarrow 8 \rightarrow 0 \rightarrow 7 \rightarrow 21 \rightarrow 15 \rightarrow 40 \rightarrow 33 \rightarrow 28 \rightarrow 1 \rightarrow 41 \rightarrow 25 \rightarrow 3 \rightarrow 34 \rightarrow 44 \rightarrow 9 \rightarrow 23 \rightarrow 31 \rightarrow 38 \rightarrow 47 \rightarrow 4 \rightarrow 17$

avec la distance totale parcouroue

$$L = 39988.96$$

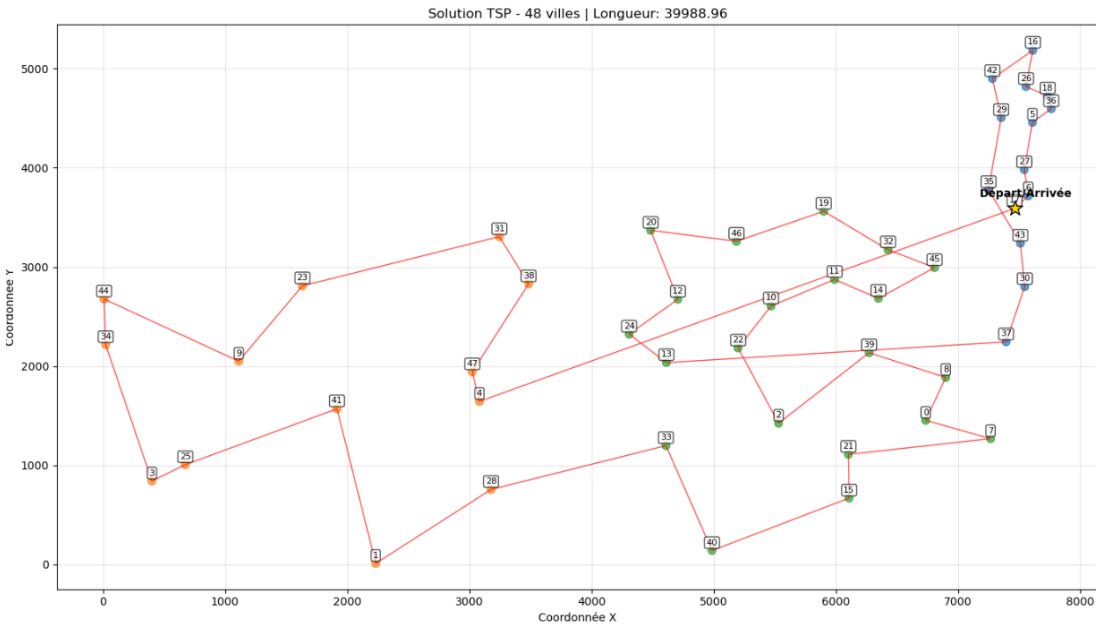

FIGURE 3.5 – Chemin global

Conclusion

Dans ce chapitre, nous avons présenté une approche hybride pour résoudre le problème du voyageur de commerce (TSP), combinant le clustering pour diviser le problème en sous-problèmes, et l'algorithme ACO pour optimiser les trajets au sein de chaque cluster.

Les clusters sont formés selon la répartition géographique des villes, puis traités individuellement par ACO, avant d'être connectés via une stratégie minimisant les distances inter-clusters. Une expérimentation sur une petite instance a illustrée le fonctionnement de chaque étape. Les résultats montrent des solutions proches de l'optimal avec un gain de temps significatif, notamment pour les grandes instances, c'est ce que nous tentons de mettre en évidence dans le chapitre suivant.

4

Implémentation et expérimentations

Introduction

Dans ce chapitre, nous présentons une étude expérimentale visant à évaluer l'impact du **clustering** combiné à l'algorithme **ACO** pour résoudre le problème du voyageur de commerce (**TSP**). Plusieurs instances issues de la bibliothèque **TSPLIB** ont été utilisées pour garantir la diversité des tests. Quatre méthodes de clustering sont considérées : **K-means**, **GMM**, **DBSCAN** et **OPTICS**. Nous précisons ensuite les **paramètres choisis** pour chaque méthode, tant pour ACO que pour les algorithmes de regroupement. Les performances sont comparées selon trois critères : **qualité de solution**, **temps de calcul**, et **écart relatif**. L'objectif est d'identifier la combinaison la plus efficace pour améliorer la résolution du TSP.

Sommaire

Introduction **59**
4.1 Instances testées (TSPLIB) **59**
4.2 Environnement de travail **60**
4.3 Choix des paramètres **61**
4.4 Comparaison des performances **62**
Conclusion **69**

4.1 Instances testées (TSPLIB)

TSPLIB (en anglais Traveling Salesman Problem Library) [23] est une bibliothèque développée par Gerhard Reinelt en 1991 dans le but de fournir un ensemble de références communes et reproductibles aux chercheurs travaillant dans le domaine de l'optimisation. Les instances de TSPLIB sont exprimées sous un format texte standardisé, incluant les coordonnées des villes

2D, les distances entre les nœuds, ou des matrices de coûts explicites. Elle inclut des insctances TSP systémériques avec un nombre de villes allant de 14 villes à 85 900 villes.

4.2 Environnement de travail

Pour la mise en œuvre de notre méthodologie, on a choisi d'utiliser le langage de programmation Python, en raison de ses nombreux atouts qui en font un outil de référence dans les domaines de l'optimisation, du machine learning et de la recherche opérationnelle. En effet, Python offre une courbe d'apprentissage rapide , une syntaxe claire , ainsi qu'un écosystème extrêmement riche en bibliothèques scientifiques, ce qui la mise en œuvre et l'expérimentation d'algorithmes complexes.

De plus, Python permet une page de prototypage rapide , indispensable pour tester plusieurs approches dans un temps limité. Son interopérabilité , sa portabilité sur toutes les plateformes et le soutien d'une communauté active renforce encore son intérêt dans les projets académiques comme industriels.
Python est un langage de programmation interprété, libre et multi-paradigme, créé par Guido van Rossum en 1991. Il est aujourd'hui l'un des langages les plus utilisés dans les domaines scientifiques et techniques[25].

Les bibliothèques utilisées dans ce projet :

numpy elle est utilisé pour Manipulation de données numériques (vecteurs, matrices, calculs mathématiques). Dans notre code est utilisé pour stocker les coordonnées des villes, Calculs vectoriels, distances, Manipulation des chemins.

matplotlib.pyplot Elle est utilisé pour Affichage et visualisation des résultats.

from sklearn.mixture import GaussianMixture Elle est utilisé pour Algorithme de clustering probabiliste basé sur les mélanges gaussiens (GMM).

scipy.spatial.distance import cdist Utilisé pour calcul rapide de la matrice des distances euclidiennes entre tous les couples de villes.

gzip Utilité pour lire des fichiers .tsp.gzcompressés (format GZIP).

random utilisé pour Générer des nombres ou sélections aléatoires.

sklearn.cluster import DBSCAN, KMeans, OPTICS Utilisé pour import les les algorithme DBSCAN, KMeans, OPTICS.

time Utilité pour mesurer le temps d'exécution.

L'ensemble des développements et expérimentations ont été réalisés sur un ordinateur portable personnel Lenovo ThinkPad X280 , équipé d'un processeur Intel Core i3 8^{eme} génération et de 8 Go de mémoire RAM . Le système d'exploitation utilisé est Windows 11 , avec un environnement Python installé via la distribution Anaconda . Le travail a été principalement effectué dans des notebooks Jupyter , ce qui a permis une intégration fluide entre le code, les résultats et les visualisations graphiques.

FIGURE 4.1 – Le langage python et Jupyter

4.3 Choix des paramètres

4.3.1 Le choix des paramètres de la méthode ACO

Les paramètres de l'algorithme ACO utilisés dans ce chapitre sont représentés dans le tableau 4.1.

Paramètre	Valeur	Description
m	30	Nombre de fourmis utilisées à chaque itération.
$t_{\max}$	100	Nombre total d'itérations (cycles de colonie).
α	1.0	Influence de la trace de phéromone dans la probabilité de transition.
β	3.0	Influence de la visibilité (inverse de la distance) dans la probabilité de transition.
ρ	0.5	Taux d'évaporation des phéromones après chaque itération.
Q	100	Quantité de phéromone déposée (coefficient de contribution).
τ_0	$\frac{1}{n}$	Valeur initiale des phéromones sur tous les arcs (n étant le nombre de villes).

TABLE 4.1 – Paramètres de l'algorithme ACO utilisés pour la résolution du TSP

4.3.2 Choix des paramètres de clustering

Le clustering joue un rôle essentiel dans la résolution TSP, il donne la possibilité de diviser le problème en sous-problèmes plus petits. Cette décomposition permet de réduire la complexité globale du TSP.
Le choix des paramètres de clustering est donc relié a leur impact sur la qualité des clusters générés et aussi sur la qualité de la solution obtenue. Les paramètres suivants sont choisis en considérant qu'il s'agisse du nombre de clusters k pour K-Means et GMM, ou des paramètres eps et $minPts$ pour OPTICS et DBSCAN. Pour chaque méthode, plusieurs combinaisons de valeurs ont été testées en les faisant varier selon des intervalles adaptés aux caractéristiques du jeu de données. L'objectif de cette phase expérimentale était d'identifier les configurations les plus pertinentes, en évaluant la qualité des regroupements produits dans le cadre de leur utilisation sur le problème du TSP. Les paramètres retenus sont ceux ayant permis d'obtenir les meilleurs résultats observés lors des expérimentations.

4.4 Comparaison des performances

Dans cette section, nous avons appliqué une approche combinant le clustering et l'algorithme ACO pour résoudre plusieurs instances du TSP.

Les jeux de données utilisés dans cette exécution 4.2 :

N°	Jeu de données	Nb. villes	Longueur optimale	N°	Jeu de données	Nb. villes	Longueur optimale
1	eil51	51	426	9	rat783	783	8806
2	eil76	76	538	10	a280	280	2579
3	pr76	76	108159	11	dsj1000	1000	18660188
4	kroE100	100	22068	12	pcb1173	1173	56892
5	kroB200	200	29437	13	fl1400	1400	20127
6	gil262	262	2378	14	fnl4461	4461	182566
7	lin318	318	42029	15	brd14051	14051	469385
8	pcb442	442	50778				

TABLE 4.2 – Caractéristiques des jeux de données TSP utilisés

Dans un premier temps, nous comparons les performances des différentes méthodes de clustering combinées avec l'ACO selon trois critères : la longueur totale des circuits obtenus (Tableau 4.3), le temps d'exécution en secondes (Tableau 4.5) et l'écart relatif exprimé en pourcentage (Tableau 4.4).

Comparaison en termes de qualité Les tableaux 4.3 et 4.4, et la figeur 4.2 comparent les performances de différentes combinaisons de méthodes de clustering (GMM, K-means, DBSCAN et OPTICS) avec l'algorithme ACO sur plusieurs instances du TSP. Le premier tableau donne les longueurs absolues des circuits obtenus, tandis que le second montre les écarts relatifs (en %) par rapport aux longueurs optimales.

N°	Jeu de données	GMM+ACO	K-means+ACO	DBSCAN+ACO	OPTICS+ACO
1	eil51	487.08	472.72	555.83	549.28
2	eil76	622.48	616.72	732.29	689.54
3	pr76	138713.19	133729.21	149221.33	142471.82
4	kroE100	26974.15	25454.59	27567.70	27318.04
5	kroB200	35538.48	35823.43	40086.66	41202.22
6	gil262	2869.11	2912.57	3346.61	3423.00
7	lin318	50351.14	50641.85	55121.60	58032.95
8	pcb442	63159.08	63198.38	71521.25	72593.22
9	rat783	11017.24	10930.15	13991.13	12783.65
10	a280	3311.18	3284.38	3777.17	3620.51
11	dsj1000	23650482.77	22879090.60	25993194.99	25498167.10
12	pcb1173	74704.06	72498.26	81025.08	81946.87
13	fl1400	26116.86	26773.11	28072.72	28442.01
14	fnl4461	228712.39	234285.99	257509.49	249946.27
15	brd14051	595639.25	593678.39	-	-

TABLE 4.3 – Comparaison des performances des méthodes de clustering (GMM, K-means, DBSCAN, OPTICS) combinées à l'algorithme ACO sur différentes instances du TSP. Les valeurs indiquent la longueur totale des circuits obtenus.

L'écart relatif ε entre une solution obtenue L_{sol} et la solution optimale L_{opt} est donné par la formule suivante :

$$\varepsilon = \left(\frac{L_{\text{sol}} - L_{\text{opt}}}{L_{\text{opt}}} \right) \times 100\%$$

N°	Instance	GMM+ACO	K-means+ACO	DBSCAN+ACO	OPTICS+ACO
1	eil51	14.34%	10.97%	30.48%	28.94%
2	eil76	15.70%	14.63%	36.11%	28.17%
3	pr76	28.25%	23.64%	37.96%	31.72%
4	kroE100	22.23%	15.35%	24.92%	23.79%
5	kroB200	20.73%	21.70%	36.18%	39.97%
6	gil262	20.65%	22.48%	40.73%	43.94%
7	lin318	19.80%	20.49%	31.15%	38.08%
8	pcb442	24.38%	24.46%	40.85%	42.96%
9	rat783	25.11%	24.12%	58.88%	45.17%
10	a280	28.39%	27.35%	46.46%	40.38%
11	dsj1000	26.74%	22.61%	39.30%	36.64%
12	pcb1173	31.31%	27.43%	42.42%	44.04%
13	fl1400	29.76%	33.02%	39.48%	41.31%
14	fnl4461	25.28%	28.33%	41.05%	36.91%
15	brd14051	26.90%	26.48%	—	—

TABLE 4.4 – Écarts relatifs (%) entre les longueurs obtenues par les méthodes de clustering combinées à ACO et les longueurs optimales pour chaque instance TSP

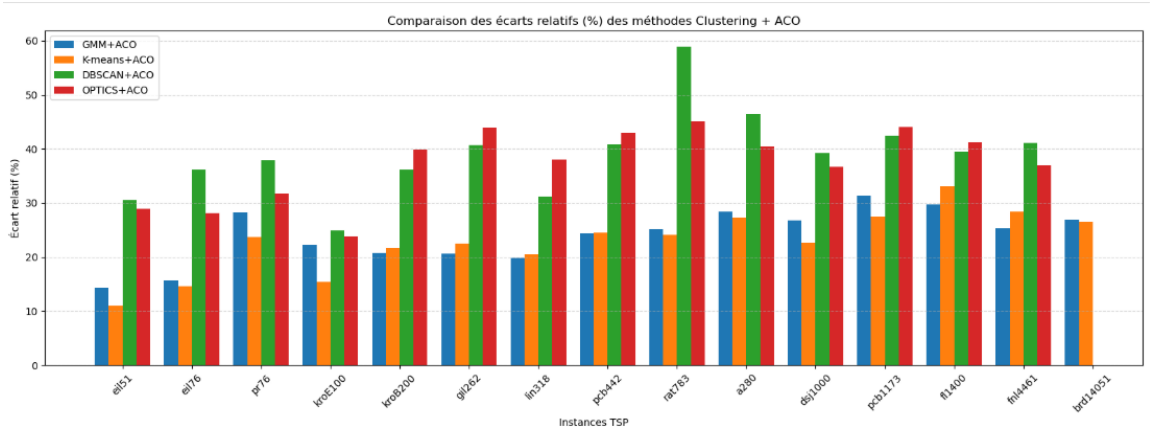

FIGURE 4.2 – Comparaison des écarts relatifs (%) des méthodes Clustering + ACO sur différentes instances TSP

On remarque que l'approche K-means+ACO se distingue par sa régularité et sa robustesse : elle fournit les meilleures longueurs ou des valeurs très proches des meilleures pour la majorité des instances, notamment sur eil51, eil76, pr76, kroE100, a280 et dsj1000.

L'approche GMM+ACO donne également des résultats compétitifs, souvent proches de ceux de K-means+ACO, et parfois meilleurs, comme sur gil262 et fl1400.

En revanche, les méthodes DBSCAN+ACO et OPTICS+ACO produisent généralement des circuits plus longs, avec des écarts relatifs plus élevés, ce qui indique un découpage moins efficace des clusters dans le cadre du TSP. Cela se reflète particulièrement dans les instances de grande taille comme pcb1173, fnl4461 et pcb442, où les écarts dépassent souvent les 40 %. On remarque également que pour certaines très grandes instances, comme brd14051, les résultats ne sont pas disponibles pour DBSCAN et OPTICS, probablement en raison de leur faible efficacité ou de problèmes liés au bruit ou à la densité variable.

Globalement, K-means+ACO semble offrir le meilleur compromis entre performance et stabilité, suivi de près par GMM+ACO, tandis que DBSCAN et OPTICS affichent une qualité de solution plus variable et souvent inférieure.

Comparaison en termes de temps d'exécution La figure 4.3 et le tableau 4.5 présentent une comparaison des temps d'exécution (en secondes) des différentes méthodes de clustering (GMM, K-means, DBSCAN et OPTICS) combinées à l'algorithme ACO pour la résolution des instances considérées.

N°	Jeu de données	GMM+ACO	K-means+ACO	DBSCAN+ACO	OPTICS+ACO
1	eil51	1.21	1.15	1.51	1.01
2	eil76	1.99	1.86	1.75	1.2
3	pr76	2.14	1.93	2 1.71	1.3
4	kroE100	3.27	2.75	5.31	3.36
5	kroB200	11.40	9.68	12.3	9.37
6	a280	14.13	12.19	15.98	12.3
7	gil262	13.50	12.01	14.52	12.1
8	lin318	19.18	18.64	19.49	18.5
9	pcb442	37.14	34.40	38.25	28.74
10	rat783	69.98	61.80	284.30	60.2
11	dsj1000	66.35	62.84	280.91	60.63
12	pcb1173	65.99	63.10	300	75
13	fl1400	99.90	88.62	600	140.4
14	fnl4461	491.01	188.46	1435.65	1016.30
15	brd14051	718.47	576.30	-	-

TABLE 4.5 – Comparaison des temps d'exécution (en secondes) des méthodes de clustering (GMM, K-means, DBSCAN et OPTICS) combinées à l'algorithme ACO pour la résolution de plusieurs instances TSP.

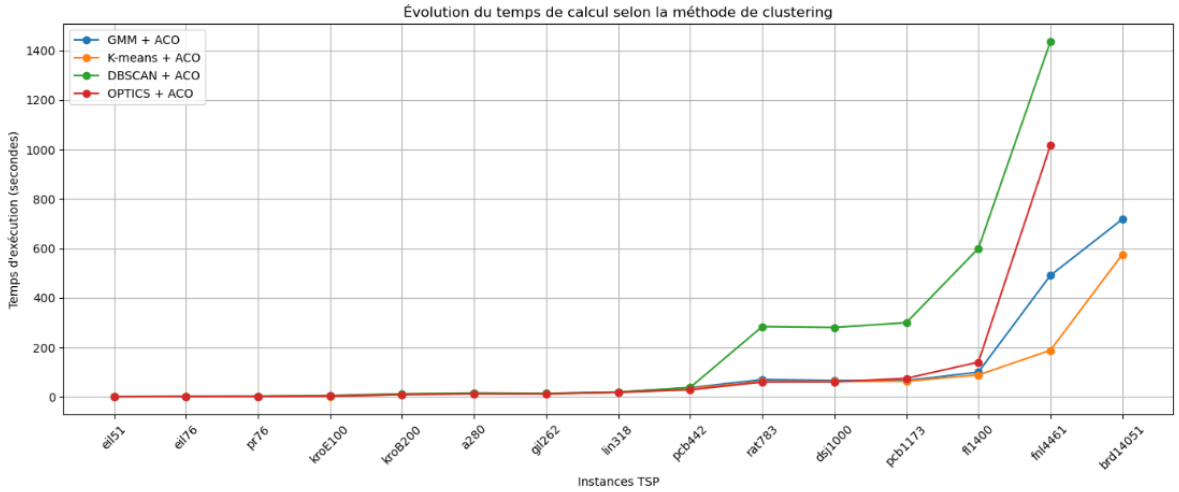

FIGURE 4.3 – Évolution du temps de calcul selon la méthode de clustering utilisée sur l'ensemble des instances testées

On remarque que pour les problèmes de petite taille ($n \leq 100$), toutes les méthodes $GMM + ACO$, $K - means + ACO$, $DBSCAN + ACO$ et $OPTICS + ACO$ présentent des temps d'exécution très faibles, ne dépassant pas 5 secondes, tout en fournissant des solutions de qualité comparable. Cela indique que, dans ce cas, le choix de la méthode de clustering a peu d'impact sur le coût temporel.

En revanche, lorsqu'on passe à des instances de taille moyenne, on observe que les méthodes GMM+ACO, K-means+ACO et OPTICS+ACO restent très efficaces, maintenant des temps d'exécution raisonnablement faibles. À l'opposé, DBSCAN+ACO devient nettement plus coûteuse en temps, avec des hausses importantes à partir de $n > 442$ (voir instances rat783, dsj1000), traduisant une scalabilité plus limitée.

Enfin, pour les instances de très grande taille, comme fl1400, fnl4461 ou brd14051, seules K-means+ACO et GMM+ACO parviennent à conserver une exécution acceptable, tout en maintenant une bonne qualité de solution. Toutefois, K-means+ACO présente un net avantage sur GMM+ACO, en particulier en termes de temps de calcul, ce qui en fait la méthode la plus favorable pour le traitement d'instances de grande dimension. Cette tendance est clairement visible dans la figure, où la courbe associée à K-means croît plus lentement que celles des autres méthodes, en particulier DBSCAN et OPTICS, qui deviennent inapplicables au-delà d'un certain seuil de complexité.

On conclut que l'approche K-means+ACO est la solution la plus performante parmi toutes les méthodes hybrides étudiées. Elle se distingue par sa robustesse, sa régularité et sa scalabilité, offrant des résultats de très bonne qualité (faibles écarts relatifs par rapport aux optima) tout en maintenant des temps d'exécution réduits, même sur des instances de très grande taille. Contrairement aux méthodes DBSCAN+ACO et OPTICS+ACO, qui deviennent rapidement inefficaces ou inapplicables à mesure que la taille des problèmes augmente, K-means+ACO reste stable et rapide. Bien que GMM+ACO produise aussi des résultats de qualité, ses temps de calcul plus élevés sur les grandes instances le rendent moins avantageux. Ainsi, K-means+ACO représente le meilleur compromis entre efficacité, précision et temps de calcul, ce qui en fait

la méthode recommandée pour résoudre efficacement les problèmes du voyageur de commerce sur un large éventail d'instances.

Comparaison ACO et ACO/Clustering Le tableau 4.6 et la figure 4.4 comparent l'algorithme ACO seul à l'ACO combiné au clustering (notamment K-means), avec en évidence l'intérêt majeur d'intégrer une phase de décomposition du problème, en particulier pour les instances de grande taille.

N°	Jeu de données	ACO	Temps (s)	Écart relatif	ACO+clustering	Temps (s)	Écart relatif
1	eil51	451.84	14.58	6.07%	472.72	1.15	10.97%
2	eil76	565.51	17.03	5.11%	616.72	1.86	14.63 %
3	pr76	117570.61	18.39	8.70%	133729.21	1.93	23.64%
4	kroE100	24031.68	132.6	8.9%	25454.59	2.75	15.35%
5	kroB200	32793.86	494.47	11.4%	35538.48	11.4	20.73%
6	gil262	2730.03	647.79	14.8%	2869.11	13.50	20.65%
7	lin318	48149.28	1017.07	14.56%	50351.14	19.18	19.8%
8	pcb442	63687.65	1316.33	25.8%	63169.08	37.14	24.38%
9	rat783	11592.59	3118.41	25.50%	10930.15	61.8	24.12%
10	a280	3168.81	1718.72	22.87%	3284.38	12.19	27.35%
11	dsj1000	-	>1000000	-	22879090.60	62.84	22.61%
12	pcb1173	-	>1000000	-	72498.26	63.10	27.43%
13	fl1400	-	>1000000	-	26116.86	99.90	29.76%
14	fnl4461	-	>1000000	-	228712.39	491.01	25.28%
15	brd14051	-	>1000000	-	593678.39	576.30	26.48%

TABLE 4.6 – Comparaison des performances entre ACO sans cluster et ACO avec clustering sur plusieurs instances TSP

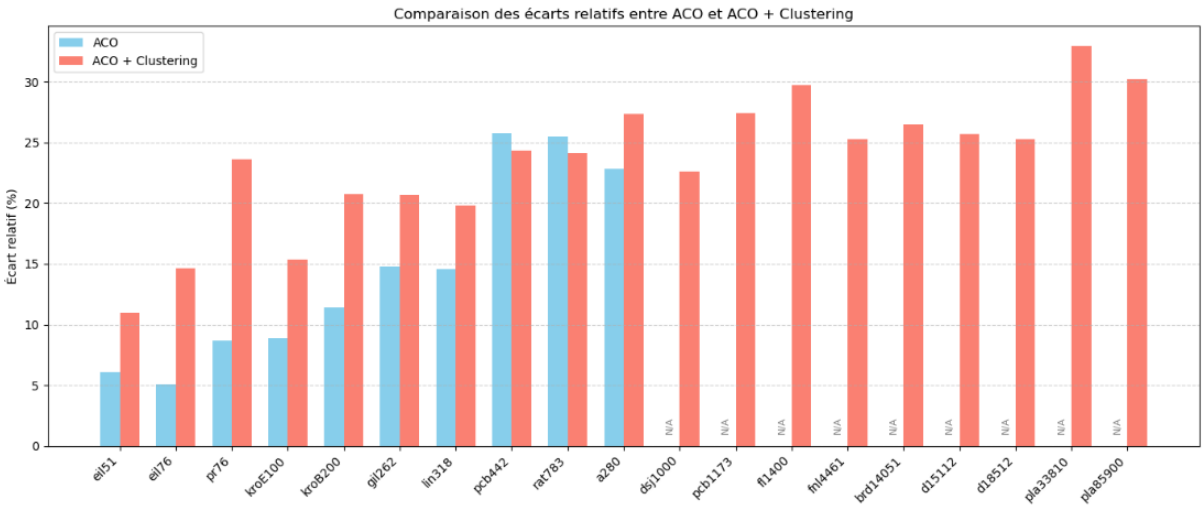

FIGURE 4.4 – Comparaison des écarts relatifs entre ACO et ACO + Clustering sur plusieurs instances du TSP

On remarque sur les petites et moyennes instances (telles que eil51 , eil76 , pr76), l'ACO sans clustering fournit généralement des solutions de meilleure qualité, avec des écarts relatifs

plus faibles par rapport aux optima connues. Par exemple, sur l'instance eil76 , l'ACO seul atteint un écart de seulement 5,11 %, contre 14,63 % pour la méthode avec clustering. Toutefois, cette performance a un coût important en temps de calcul : les temps d'exécution explosent rapidement avec la taille du problème. Sur les grandes instances (pcb1173, fl1400, fnl4461, brd14051), l'ACO sans clustering devient inopérante, ne produisant aucun résultat exploitable dans un délai raisonnable (temps d'exécution dépassant 1 000 000 secondes).

En revanche, l'ACO couplé au clustering permet d'obtenir des solutions fiables en quelques secondes ou minutes seulement, tout en maintenant des écarts relatifs acceptables (compris entre 22 % et 30 % dans la plupart des cas). Cela montre que le clustering réduit considérablement la complexité du problème en le décomposant en sous-tours plus simples à optimiser, ce qui rend l'approche ACO scalable. Bien qu'on observe une légère dégradation de la qualité sur les petites instances, cette perte est largement compensée par les gains en efficacité sur les grandes tailles.

Application de ACO/K-means sur de grandes instances de TSP Le tableau 4.7 présente les résultats de l'application de l'approche hybride ACO couplée au clustering K-means sur quatre grandes instances du TSP.

N°	Jeu de données	ACO+clustering	Temps (s)	Écart relatif
1	d15112	1 977 305.56	1 341.44	25.70%
2	d18512	808 140.21	2 049.73	25.25%
3	pla33810	87 819 369.93	2 330.34	32.96%
4	pla85900	185 422 527.26	8 482.48	30.23%

TABLE 4.7 – Application de ACO/K-means sur de grandes instances de TSP

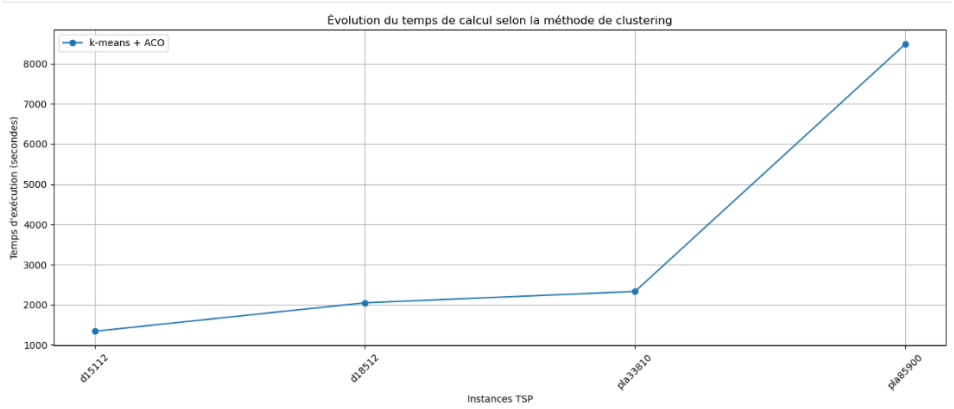

FIGURE 4.5 – Évolution du temps de calcul (en secondes) pour l'approche K-means + ACO sur de grandes instances TSP

On observe tout d'abord une augmentation significative du temps d'exécution avec la taille de l'instance : l'instance d15112 est résolue en 1341,44 secondes, tandis que pla85900 nécessite 8482,48 secondes, ce qui reflète la complexité croissante du problème avec le nombre de villes. En termes de qualité de solution, mesurée par l'écart relatif, les résultats restent relativement constants autour de 25 à 33 %, ce qui montre que, malgré la montée en taille, la

méthode ACO+K-means maintient une performance stable en qualité. L'instance pla33810 présente l'écart relatif le plus élevé (32,96 %), ce qui peut être lié à une structure plus complexe du graphe ou à une difficulté plus grande pour le partitionnement initial par K-means.

Globalement, cette approche hybride offre un compromis intéressant entre temps de calcul et qualité de solution , en permettant de résoudre efficacement des instances de très grande taille qui seraient inaccessibles par une ACO pure sans clustering préalable.

En conclusion, le clustering joue un rôle fondamental dans la résolution du TSP avec ACO : il améliore drastiquement les temps de calcul et permet de traiter des instances autrement inaccessibles, tout en préservant une qualité de solution raisonnable. C'est donc un levier essentiel pour garantir un bon compromis entre performance et précision dans les approches heuristiques basées sur les colonies de fourmis.

Conclusion

L'étude expérimentale a mis en évidence l'intérêt d'associer l'algorithme ACO à une phase de clustering pour améliorer l'efficacité de résolution du problème TSP. D'un point de vue du temps de calcul, les méthodes K-means + ACO et OPTICS + ACO se sont révélées les plus rapides, permettant une réduction significative du temps par rapport à ACO seul, même sur des instances de grande taille. Du côté de la qualité des solutions , K-means + ACO et GMM + ACO se démarquent en obtenant les circuits les plus courts, avec une légère supériorité de K-means. À l'inverse, DBSCAN + ACO a montré des performances irrégulières, notamment sur les grandes instances, avec des temps de calcul très élevés et une difficulté à bien gérer la structure des données. En somme, l'approche K-means + ACO ressort comme la plus équilibrée, alliant précision des résultats et rapidité d'exécution.

Conclusion générale

Dans ce mémoire, nous avons introduit le TSP en définissant les fondements théoriques, avant de présenter de manière synthétique les principales méthodes de résolution disponibles dans la littérature. En commençant par une présentation des méthodes exactes. Les méthodes exactes assurent des solutions optimales mais restent limitées à de petites instances en raison de leur complexité exponentielle . Pour faire face à cette limite, nous avons exploré dans ce mémoire les méthodes approchées , notamment les métaheuristiques, et plus précisément l'algorithme à colonies de fourmis (ACO) , reconnu pour son efficacité à fournir de bonnes solutions dans des temps raisonnables.

Afin d'améliorer encore les performances et la scalabilité de l'ACO sur de grandes instances, une stratégie couramment adoptée consiste à combiner cet algorithme avec des techniques de clustering. Le clustering permet de diviser l'espace de recherche en sous-clusters, une connexion intelligente visant à reconstituer un chemin global cohérent. Cette stratégie permet d'exploiter les forces respectives du partitionnement spatial et de l'exploration stochastique.

Les résultats expérimentaux, obtenus sur des instances variées de la bibliothèque TSPLIB , ont démontré l'efficacité de cette approche hybride : elle permet de réduire significativement les temps de calcul , tout en maintenant une qualité de solution compétitive . L'impact du choix de la méthode de clustering (K-Means, OPTICS, GMM) sur la qualité finale a également été mis en évidence, soulignant l'importance de la structuration préalable du problème.

En conclusion, cette approche constitue une base solide et évolutive pour la résolution du TSP à grande échelle. Elle ouvre la voie à plusieurs pistes de recherche, telles que l'intégration de stratégies multi-niveaux ou adaptatives, l'hybridation avec d'autres métaheuristiques (recherche tabou, recuit simulé), l'application à des variantes du TSP (multi-voyageurs, contraintes de capacité, etc.), ou encore l'optimisation parallèle et distribuée sur les architectures modernes.

Ainsi, notre travail s'inscrit dans une dynamique d'amélioration continue des techniques de résolution du TSP, en combinant intelligence algorithmique et structuration des données pour produire des solutions efficaces, robustes et applicables à des problèmes réels complexes.

Bibliographie

[1] Ahmed, A., Kacem, I., and Chu, C. Combining hierarchical clustering with metaheuristics for solving large-scale tsp instances. *International Journal of Operational Research 21*, 3 (2014), 307–325.

[2] Ahmed, A., Kacem, I., and Chu, C. Combining hierarchical clustering with metaheuristics for solving large-scale tsp instances. *International Journal of Operational Research 21*, 3 (2014), 307–325.

[3] Ankerst, M., Breunig, M. M., Kriegel, H.-P., and Sander, J. Optics : Ordering points to identify the clustering structure. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data* (Philadelphia, PA, USA, 1999), ACM, pp. 49–60.

[4] Bektas, T., and Laporte, G. The pollution-routing problem. *Transportation Research Part B : Methodological 41*, 7 (2007), 761–778.

[5] Cai, X., Wang, K., Mei, Y., Li, Z., Zhao, J., and Zhang, Q. Decomposition-based lin-kernighan heuristic with neighborhood structure transfer for multi/many-objective traveling salesman problem. *IEEE Transactions on Evolutionary Computation* (2022), 1–1. Early access.

[6] Chen, R., and Wu, J. Solving traveling salesman problem by combining clustering and genetic algorithm. In *Proceedings of the International Conference on Machine Learning and Cybernetics* (2003), vol. 3, pp. 1539–1543.

[7] Croes, G. A. A method for solving traveling-salesman problems. *Operations Research 6*, 6 (1958), 791–812.

[8] Gherboudj, A. *Méthodes de résolution de problèmes difficiles académiques*. Thèse de doctorat, 3^{e} cycle lmd, Université Constantine 2, Algérie, 2013. Thèse soutenue devant un jury universitaire le 2013 à la Faculté des Technologies de l'Information et de la Communication.

[9] Glover, F. Tabu search – part i. *ORSA Journal on Computing 1*, 3 (1989), 190–206.

[10] Hertono, G. F., and Handari, B. D. The modification of hybrid method of ant colony optimization, particle swarm optimization and 3-opt algorithm in traveling salesman problem. In *Journal of Physics : Conference Series* (2018), vol. 974, IOP Publishing, p. 012032.

[11] Ismail, A., Sulaiman, N., and Ghazali, R. Solving traveling salesman problem using k-means clustering and genetic algorithm. In *2016 6th International Conference on Computer and Communication Engineering (ICCCE)* (2016), IEEE, pp. 206–210.

[12] Jayaprada, S., Aswani, A., and Gayathri, G. Hierarchical divisive clustering with multi view-point based similarity measure. In *Proceedings of the International Conference on Soft Computing and Pattern Recognition (SoCPaR)*. Springer, Cham, 2014, pp. 483–491.

[13] Kumar, R., Pati, P. B., and Deepa, K. Clustering the various categorical data : An exploration of algorithms and performance analysis. In *2023 IEEE International Conference on Next-Generation Computing (INCET)* (2023), pp. 1–6.

[14] Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., and Shmoys, D. B. *The Traveling Salesman Problem*. John Wiley & Sons, Chichester, UK, 1985. Chapitres sur les méthodes exactes pour la résolution du TSP.

[15] Mhenni, R. B., Bourguignon, S., Mongeau, M., Ninin, J., and Carfantan, H. Algorithme branch-and-bound pour l'optimisation exacte en norme 0. *RAIRO - Operations Research 54*, 1 (2020), 49–71. Disponible en ligne : https://doi.org/10.1051/ro/2019037.

[16] Monath, N., Dubey, A., Guruganesh, G., Zaheer, M., Ahmed, A., McCallum, A., Mergen, G., Najork, M., Terzihan, M., Tjanaka, B., Wang, Y., and Wu, Y. Scalable hierarchical agglomerative clustering. https://doi.org/10.1145/3447548.3467404, 2020. DOI : 10.1145/3447548.3467404.

[17] Nemhauser, G. L., and Wolsey, L. A. *Integer and Combinatorial Optimization*. Wiley, New York, 1988.

[18] Oloo, J. M. Examining gaussian mixture models using clustering algorithms. Master of science in statistical sciences, Strathmore University, Nairobi, Kenya, 2023. Thèse de Master, Institut de Mathématiques, disponible en libre accès à la bibliothèque de Strathmore University.

[19] Oti, E. U., and Olusola, M. O. Overview of agglomerative hierarchical clustering methods. *British Journal of Computer, Networking and Information Technology 7*, 2 (2024), 14–23.

[20] Oti, E. U., and Olusola, M. O. Overview of agglomerative hierarchical clustering methods. *British Journal of Computer, Networking and Information Technology 7*, 2 (2024), 14–23.

[21] Patel, S. S., Kumar, N., Aswathy, J., Vaddadi, S. K., Akbar, S. A., and Pancharіya, P. C. K-means algorithm : An unsupervised clustering approach using various similarity/dissimilarity measures. In *Proceedings of the International Conference on Intelligent Computing and Smart Communication (ICICSC 2021)*. Springer, Singapore, 2022, pp. 805–813.

[22] Rahman, M. S., Hasan, M. M., and Rokonuzzaman, M. Improvement of the nearest neighbor heuristic search algorithm for traveling salesman problem. *Indonesian Journal of Electrical Engineering and Computer Science 33*, 1 (2024), 505–512.

[23] Reinelt, G. Tsplib—a traveling salesman problem library. *INFORMS Journal on Computing 3*, 4 (1991), 376–384.

[24] Reinelt, G. Tsplib - a library of sample instances for the tsp (and related problems). http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/, 1995. Consulté en juin 2025.

[25] Rossum, G. *Python Reference Manual*, 2000. http://web.mit.edu/18.417/doc/pydocs/ref.pdf.

[26] Soh, M., Tsofack, B. N., and Djamegni, C. T. Approche heuristique multi colonie des fourmis pour la résolution du problème de voyageur de commerce. In *Colloque Africain sur*

la Recherche en Informatique et en Mathématiques Appliquées (CARI) (Thiès, Sénégal, 2020). Document disponible sur HAL : https://hal.science/hal-02958748v1.

[27] Wang, Y., Huang, J., and Liu, Y. An improved clustering-based genetic algorithm for the traveling salesman problem. *Journal of Computers 5*, 8 (2010), 1233–1240.

[28] Watanabe, H. Clustering as average entropy minimization and its application to structure analysis of complex systems. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC)* (2001), vol. 4, IEEE, pp. 2408–2414.

[29] Weber, V. *Caractérisation des instances difficiles de problèmes d'optimisation NP-difficiles*. Thèse de doctorat, Université de Grenoble, France, 2013. Thèse dirigée par Nadia Brauner et codirigée par Yann Kieffer, soutenue le 8 juillet 2013 au laboratoire G-SCOP.

[30] Wikipedia contributors. Problème du voyageur de commerce — wikipédia. https://fr.wikipedia.org/wiki/Probl%C3%A8me_du_voyageur_de_commerce, 2024. Page consultée le 16 juin 2025.

[31] Yamada, Y., Masuyama, N., Amako, N., Nojima, Y., Loo, C. K., and Ishibuchi, H. Divisive hierarchical clustering based on adaptive resonance theory. In *Proceedings of the IEEE Conference on Computer and Communications Security (CCS)* (2020), pp. 1–6.

[32] Zhao, W. Block-diagonal guided dbscan clustering, 2024. arXiv preprint.

Résumé

Le problème du voyageur de commerce (TSP) est un classique de l' optimisation combinatoire , consistant à déterminer le plus court chemin passant par un ensemble de villes une seule fois avant de revenir au point de départ. Sa simplicité apparente contraste avec sa complexité algorithmique, ce qui en fait un problème de référence dans de nombreuses applications industrielles et scientifiques, telles que la logistique, la planification de tournées, ou les circuits imprimés.

Ce travail s'inscrit dans une approche visant à dépasser les limites des méthodes exactes, qui deviennent inapplicables pour de grandes instances à cause de leur complexité exponentielle. Pour cela, nous avons adopté une méthode hybride combinant clustering et l'algorithme à colonies de fourmis (ACO) , une métaheuristique bio-inspirée reconnue pour ses performances sur les problèmes NP-difficiles.

L'idée principale repose sur le paradigme "diviser pour régner" : les villes sont d'abord regroupées en clusters homogènes à l'aide d'algorithmes de clustering (K-Means, OPTICS, GMM), afin de faciliter la résolution locale de sous-TSP. Ensuite, l'algorithme ACO est appliqué sur chaque cluster, générant des sous-tours de bonne qualité. Une stratégie de reconnexion intelligente permet ensuite de reconstruire une solution globale cohérente.

Les expérimentations réalisées sur différentes instances de la bibliothèque TSPLIB , y compris des instances de grande taille, ont permis de valider l'efficacité de cette approche. Elle permet de réduire considérablement les temps de calcul tout en produisant des solutions de qualité compétitive , souvent proches de celles obtenues par ACO seul, mais avec un gain de scalabilité significatif.

En somme, cette approche hybride constitue une solution efficace et évolutive pour le TSP. Elle ouvre également des perspectives intéressantes en matière de résolution parallèle, d'adaptation dynamique des paramètres, ou d'extension à d'autres variantes du TSP.

Mots-clés

Problème du Voyageur de Commerce, Clustering, Optimisation par Colonies de Fourmis.

Abstract

The Travelling Salesman Problem (TSP) is a well-known problem in combinatorial optimization, which consists of finding the shortest possible route that visits a set of cities exactly once and returns to the starting point. Despite its simple formulation, TSP is computationally challenging and serves as a benchmark for numerous industrial and scientific applications, such as logistics, route planning, and circuit design.

This work addresses the scalability limitations of exact methods, which become intractable for large instances due to their exponential complexity. To overcome this, we propose a hybrid approach that combines clustering techniques with the Ant Colony Optimization (ACO) algorithm—a bio-inspired metaheuristic known for its efficiency on NP-hard problems.

The core idea is based on the principle of "divide and conquer" : the cities are first partitioned into homogeneous clusters using algorithms like K-Means, OPTICS, or Gaussian Mixture Models (GMM), which simplifies the global problem by reducing it to several smaller subproblems. ACO is then applied independently to each cluster to find high-quality sub-tours. A smart reconnection strategy is employed to rebuild a coherent global tour.

Experimental results on various instances from the TSPLIB benchmark—including large-scale ones—demonstrate that this hybrid approach significantly reduces computation time while delivering competitive-quality tours, often comparable to those produced by ACO applied to the entire problem.

In conclusion, this method offers a scalable and efficient solution for solving large TSP instances. It also lays the groundwork for future research directions, such as parallel solving strategies, adaptive parameter tuning, or extension to TSP variants.

keywords

Travelling Salesman Problem (TSP), Clustering, Ant Colony Optimization (ACO).