REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE YAHIA FARES DE MEDEA

Faculté des Sciences

Département Mathématiques et de l'Informatique



Mémoire de fin d'études de Master

Spécialité : Ingénierie des systèmes et technologie web

Etude comparative entre des différents algorithmes d'optimisation sur plusieurs bases des données uni-critère et multicritère.

Présenté par :

M^elle^. Bouchra ABDENNEBI.

M^elle^. Manel BENOUA.

Proposé et Dirigé par :

Dr. Mohamed Yassine HAYI.

Président de jury : Mohamed Nabil ABDOUN.

Examinateur : Azzeddine Rachid BENAISSA.

Promotion 2023 /2024

Remerciements

Nous remercions d'abord Dieu Tout-Puissant de nous avoir permis et Il nous a donné la volonté, la patience et le courage de faire ce travail.

Nous tenons à remercier sincèrement le ***Dr. Mohamed Yassine HAYI*** *pour son encadrement exceptionnel et son soutien indéfectible tout au long de notre projet.*

Tout d'abord, nous le remercions pour sa présence et son dévouement. Malgré son emploi du temps chargé, il a toujours trouvé le temps de répondre à nos questions, de relire notre travail et de nous prodiguer de précieux conseils. Son expérience et sa rigueur académique ont grandement contribué à la qualité de nos recherches.

Nous tenons également à exprimer notre gratitude pour son attitude encourageante et positive. Il a réussi à créer un environnement de travail stimulant et bienveillant, où nous nous sentions à l'aise pour partager nos idées et explorer de nouvelles approches. Sa capacité à nous motiver et à nous inspirer a été une source d'énergie et de détermination tout au long de ce projet.

Enfin, nous remercions pour sa patience et sa compréhension. Les défis et les obstacles rencontrés au cours de notre travail auraient été bien plus difficiles à surmonter sans son soutien et ses encouragements constants. Son approche pédagogique et son sens aigu de l'écoute ont été des éléments clés dans la réussite de notre projet.

Nous lui sommes infiniment reconnaissants pour tout ce qu'il a fait pour nous.

Nous aussi remercions ***Mr. Alla Tchoketch kebir***

Nous remercions nos chers parents pour tout ce qu'ils ont donné pour nous.

Nous remercions les membres de jury qui nous feront l'honneur d'estimer et juger ce travail. *Merci à tous.*

Je dédie ce projet :

À mon cher père

A mon cher père **AHMED** : aucune dédicace ne saurait exprimer l'amour, l'estime, le dévouement et le respect que j'ai toujours eu pour toi.

Ce travail est le fruit de tes sacrifices que tu as consentis pour mon éducation et ma formation.

Je te dédie ce travail en témoignage de mon profond amour, Puisse Dieu, le tout puissant, te préserver et t'accorder santé, longue vie et bonheur.

À ma chère mère

À ma chère mère, honorable. Aimable **FATIHA :** tu représentes pour moi le symbole de la bonté par excellence la source de tendresse et l'exemple du dévouement qui n'a pas cessé de m'encourager ta prière et ta bénédiction m'a été un grand secours pour mener à bien mes études. Aucune dédicace ne saurait être assez éloquente pour exprimer ce que tu mérites pour tous les sacrifices depuis ma naissance. Durant mon enfance et même à l'âge adulte.

À ma sœur **SIRINE** et mon frère **RAYANE**

À MA binôme **BOUCHRA**

Je souhaite exprimer ma profonde gratitude à mon binôme, **Bouchra**, pour son soutien inestimable et sa collaboration exemplaire tout au long de notre projet.

Merci infiniment, pour tout ce que tu as apporté à ce projet et pour l'amitié que nous avons développée au fil du temps.

À mon homme, **Oussama**, est ma belle-famille **Belkhirate.**

À mes chers amis.

Manel Benoua

Dédicace

Je commence par exprimer ma gratitude envers Dieu, source de toute grâce et de toute bénédiction

Avec un excès d'ambition, j'ai embrassé ce défi, Le voyage n'était pas court, le rêve pas proche, et la route, dépourvue de facilités. Mais je l'ai entreprise, je l'ai conquise. Je dédie ce rêve, longtemps attendu,

À mon père, dont la force, la sagesse et le soutien indéfectible ont été mes piliers dans cette entreprise. J'ai été inspiré par votre confiance en moi et vos conseils éclairés pour donner le meilleur de moi-même. Merci pour tout ce que vous avez fait pour moi.

À ma mère, la joie de ma vie, mon pilier solide et ma sécurité dans les jours incertains. Tu as été la bougie qui éclairait mes nuits sombres, la lumière qui guidait mon chemin. Merci pour ton amour infini et ta présence réconfortante.

À mes chères sœurs et frères, vous avez été mes compagnons de route, mes confidents et mes soutiens inébranlables. Votre présence et votre amour m'ont donné la force de persévérer et de réussir.

À mon binôme dans ce projet, dont la collaboration, l'engagement et le soutien ont été d'une valeur inestimable tout au long de cette aventure. Merci d'avoir partagé ce voyage avec moi.

À mon professeur, dont l'expertise, les conseils avisés et l'encouragement ont été d'une valeur inestimable dans l'accomplissement de ce travail. Ta passion pour l'enseignement et ton dévouement envers tes étudiants sont une source d'inspiration pour moi. Merci pour ton mentorat précieux.

À mes amis, dont l'amitié, le soutien et les encouragements ont illuminé mon chemin.

Et enfin, je me remercie moi-même pour ma détermination, ma persévérance et mon travail acharné tout au long de ce parcours. C'est grâce à ma volonté et à mon engagement que j'ai pu atteindre cet objectif. Je suis fière de ce que j'ai accompli et reconnaissante pour cette opportunité.

À tous ceux qui ont contribué, de près ou de loin, à la réalisation de ce mémoire, je vous adresse mes plus sincères remerciements.

BOUCHRA ABDENNEBI

Résumé

L'intelligence artificielle (IA) vise à imiter l'intelligence humaine en utilisant des algorithmes exécutés dans un environnement informatique. Avec l'évolution technologique rapide, l'analyse du Big Data est devenue essentielle pour extraire des informations pertinentes. Les techniques d'optimisation et de recommandation jouent un rôle crucial en adaptant les résultats aux besoins spécifiques des utilisateurs. Notre objectif avec ce projet est d'appliquer et de comparer différents algorithmes d'optimisation dans le contexte de l'intelligence artificielle et du Big Data. L'optimisation joue un rôle clé dans le traitement des grands volumes de données et l'amélioration des systèmes complexes. Ce mémoire se concentre sur la comparaison et l'optimisation de trois algorithmes d'optimisation : l'algorithme de Dijkstra, l'algorithme génétique et l'algorithme A*. Il étudie comment ces algorithmes peuvent résoudre des problèmes d'optimisation complexes et compare leurs performances en termes de rapidité et d'efficacité. L'objectif principal était d'analyser, d'évaluer et de comparer les performances de ces algorithmes sur quatre bases de données différentes, dont deux bases de données uni-critères et deux bases de données multicritères. Les paramètres de l'algorithme génétique, tels que la taille de la population, le nombre d'itérations et les différents opérateurs génétiques, sont détaillés. L'évaluation montre que les algorithmes A* et Dijkstra sont généralement plus rapides que l'algorithme génétique pour les bases de données uni-critères et multicritères. Les performances des algorithmes ont été mesurées en termes de temps d'exécution et de capacité à atteindre les objectifs d'optimisation minimaux ou maximaux selon les configurations de graphes et critères spécifiques.

Mots Clés : Optimisation, Plus court chemins, Résolution de problèmes complexes, Big Data, Uni-critères, Multicritères, Algorithme Dijkstra, Algorithme génétique.

Abstract

Artificial intelligence (AI) aims to mimic human intelligence by using algorithms executed in a computer environment. With the rapid evolution of technology, big data analysis has become essential to extract relevant information. Optimization and recommendation techniques play a crucial role in adapting the results to the specific needs of users. Our goal with this project is to apply and compare different optimization algorithms in the context of artificial intelligence and Big Data. Optimization plays a key role in processing large volumes of data and improving complex systems. This thesis focuses on the comparison and optimization of three optimization algorithms: the Dijkstra algorithm, the genetic algorithm and the A* algorithm. It studies how these algorithms can solve complex optimization problems and compares their performance in terms of speed and efficiency. The main objective was to analyze, evaluate and compare the performance of these algorithms on four different databases, including two single-criteria databases and two multi-criteria databases. The parameters of the genetic algorithm, such as population size, number of iterations and different genetic operators, are detailed. The evaluation shows that the A* and Dijkstra algorithms are generally faster than the genetic algorithm for single-criteria and multi-criteria databases. The performance of the algorithms was measured in terms of execution time and ability to achieve the minimum or maximum optimization objectives according to specific graph configurations and criteria.

Keywords: Optimization, Shorter paths, Algorithm comparison, Solving complex problems, Big Data, Uni-criteria, Multicriteria, Dijkstra algorithm, Genetic algorithm.

ملخص

يهدف الذكاء الاصطناعي (AI) إلى تقليد الذكاء البشري باستخدام الخوارزميات المنفذة في بيئة الكمبيوتر. مع التطور السريع للتكنولوجيا، أصبح تحليل البيانات الضخمة ضروريًا لاستخراج المعلومات ذات الصلة. وتؤدي تقنيات التحسين والتوصية دورا حاسما في تكييف النتائج مع الاحتياجات المحددة للمستعملين. هدفنا من هذا المشروع هو تطبيق ومقارنة خوارزميات التحسين المختلفة في سياق الذكاء الاصطناعي والبيانات الضخمة. يلعب التحسين دورًا رئيسيًا في معالجة كميات كبيرة من البيانات وتحسين الأنظمة المعقدة. تركز هذه الأطروحة على مقارنة وتحسين ثلاث خوارزميات تحسين: خوارزمية Dijkstraوالخوارزمية الجينية وخوارزمية .* A يدرس كيف يمكن لهذه الخوارزميات حل مشاكل التحسين المعقدة ومقارنة أدائها من حيث السرعة والكفاءة. وكان الهدف الرئيسي هو تحليل وتقييم ومقارنة أداء هذه الخوارزميات في أربع قواعد بيانات مختلفة، بما في ذلك قاعدتا بيانات أحديتا المعايير وقاعدتي بيانات متعددتي المعايير.

يتم تفصيل معايير الخوارزمية الجينية، مثل حجم السكان وعدد التكرارات ومختلف المشغلين الجينيين. يوضح التقييم أن خوارزميات *A و Dijkstra أسرع بشكل عام من الخوارزمية الجينية لقواعد البيانات أحادية المعايير ومتعددة المعايير. تم قياس أداء الخوارزميات من حيث وقت التنفيذ والقدرة على تحقيق الحد الأدنى أو أقصى أهداف التحسين وفقًا لتكوينات ومعايير الرسم البياني المحددة.

الكلمات المفتاحية: التحسين، المسارات الأقصر، مقارنة الخوارزميات، حل المشكلات المعقدة، البيانات الضخمة، أحادية المعايير، متعددة المعايير ، الخوارزمية جيكسترا، الخوارزمية الجينية .

Table des matières

Liste des figures I

Liste des tableaux V

Liste des abréviations VI

Introduction Générale

Introduction générale 1

1. Problématiques 2

2. Les objectifs de cette étude 2

3. Organisation du mémoire 2

Chapitre 1 État de l'art

1.1. Introduction 4

1.2. Qu'est-ce qu'un problème d'optimisation ? 5

1.2.1. Définition de l'optimisation 5

1.2.2. Concepts principaux en optimisation 5

1.2.3. Les étapes clés de l'optimisation 6

1.2.3.1. Modélisation 6

1.2.3.2. Analyse 6

1.2.3.3. Résolution des problèmes 7

1.2.4. Historique d'optimisation 7

1.3. Optimisation Uni-critère 8

1.4. Optimisation Multicritère 8

1.5. Théorie des graphes et Optimisation 9

1.5.1. Graphes 9

1.6. Quelques Algorithmes d'optimisation 10

1.6.1. Algorithme de Dijkstra 10

1.6.2. Algorithme Floyd-Warshall 11

1.6.3. Algorithmes Méta Heuristiques 12

1.6.3.1. Algorithme Génétique 13

1.6.3.2. Algorithme A* 14

1.6.3.3. Algorithme Gloutons 14

1.6.3.4. Recuit Simulé 15

1.6.3.5. Algorithme de Recherche Tabou 16

1.6.3.6. Algorithmes de Colonies de Fourmis (ACO) 18

1.6.3.7. Algorithme des Essaims de particules (PSO) 20

1.6.3.8. Algorithme des Lucioles (Firefly Algorithm) 20

1.7. Conclusion 22

Chapitre 2 Travaux connexes

2.1. Introduction 23

2.2. Quelques Travail connexe 23

2.2.1. Recherche de plus court chemin multimodal de point à point dépendant du temps 23

2.2.2. Extraction optimisée de règles d'association 26

2.2.3. Amélioration de l'algorithme de chauve-souris par modification de règles d'évolution et introduction de mécanisme de croisement 27

2.2.4. Chimp optimization algorithm 27

2.2.5. Dynamic Programming for Shortest Path Problem in a Multimodal Transportation Network Comprising Intermediate Sinks 29

2.2.6. Optimization design of forebody shape of an underwater axisymmetric body for transition delay using a genetic algorithm 31

2.2.7. Inner-outer layer co-optimization of sizing and energy management for renewable energy microgrid with storage 35

2.2.8. Un algorithme hybride pour la résolution du problème de partitionnement matériel/logiciel 38

2.2.9. Deep reinforcement learning for intelligent risk optimization of buildings under hazard... 39

2.3. Travaux connexes d'optimisation 40

2.4. Conclusion 42

Chapitre 3 Approche proposée

3.1. Introduction 43

3.2. Approche proposée 43

3.2.1. Architecture globale de l'approche proposée 44

3.2.2. Comparaison Algorithme 44

3.2.3. Environnement et outils de développement 45

3.2.3.1 Python 45

3.2.3.2. Kaggle 46

3.2.3.3. Google Colab 46

3.2.3.4. Les bibliothèques utilisées 46

3.3. Description et pré-traitement des bases de données 48

3.3.1. Google Web Graph 48

3.3.1.1. Traitement des données 49

3.3.2. Melbourne Restaurant Food Delivery data 54

3.3.2.1. Traitement des données 56

3.3.3. Matrices adjacentes 58

3.3.4. Chess Game Dataset (Lichess) 61

3.3.4.1. Les valeurs de chaque pièce 64

3.3.4.2. Encodage des éléments d'un jeu d'échecs avec des lettres. 64

3.3.4.3. Traitement des données 65

3.4. Tableaux de comparaison des bases de données 66

3.5. Les algorithmes utilisés 66

3.5.1. Algorithme génétique 67

3.5.2. Algorithme Dijkstra 68

3.5.2. Algorithme A* 69

3.6. Conclusion 70

Chapitre 4 Résultats et discussion

4.1. Introduction 72

4.2. Les paramètres de l'algorithme génétique 72

4.3. Analyse et évaluation des performances des algorithmes appliqués aux bases de données Uni-critère 73

4.3.1. Résultats de base de données 1(Google Web Graph) 73

4.3.2. Résultats de base de données 2 (Melbourne Restaurant Food Delivery data) 76

4.4. Analyse et évaluation des performances des algorithmes appliqués aux bases de données Multicritère 79

4.4.1. Résultats de base de données 3 (Matrices adjacentes) 80

4.4.2. Résultats de base de données 4 (Chess Game Dataset (Lichess)) 90

4.5. Tableaux de comparaison des résultats pour les bases de données 99

4.6. Complexité 100

4.7. Différentes versions des BDDs utilisées 101

4.8. Résultat et discussion 102

4.9. Conclusion 103

Conclusion Générale

Conclusion Générale 105

Les références Bibliographie

Bibliographie 106

Liste des figures

Chapitre 1 État de l'art

Figure 1.1. Explication du premier problème d'optimisation. 7

Figure 1.2. Graphe orienté. 9

Figure 1.3. Graphe non-orienté. 10

Figure 1.4. Algorithme de Dijkstra. 11

Figure 1.5. Exemple sur l'algorithme de Floyd-Warshall. 11

Figure 1.6. L'algorithme de Floyd-Warshall. 12

Figure 1.7. Classification des métaheuristiques. 13

Figure 1.8. Algorithme Génétique. 13

Figure 1.9. Organigramme de Recuit simulé. 15

Figure 1.10. Organigramme de Recherche Tabou. 17

Figure 1.11. Détermination du plus court chemin par une colonie de fourmis. 19

Figure 1.12. Organigramme de colonies de fourmis. 19

Figure 1.13. Optimisation par Essaim Particulaire (Particle Swarm Optimization) PSO. 20

Figure 1.14. Organigramme des Lucioles (Firefly Algorithm). 21

Chapitre 2 Travaux connexes

Figure 2.1. Temps de réponse en fonction du nombre de zones pour le réseau de train de la Suisse. 24

Figure 2.2. Temps de réponse en fonction du rang géographique pour le réseau de train de la Suisse. 24

Figure 2.3. Temps de réponse du YPT en fonction du temps de réponse du PYPT pour le réseau de trains de la Suisse. 25

Figure 2.4. Temps de réponse du YPT en fonction du temps de réponse du PYPT pour la ville de Stockholm. 25

Figure 2.5. Evaluation du temps de réponse des algorithmes de DM. 26

Figure 2.6. Comparaison de pertinence des RAs générée. 27

Figure 2.7. Phylogénie des hominoïdes de la superfamille. 28

Figure 2.8. ADN humain et chimpanzé. 28

Figure 2.9. Comparaison sur la base de la distance la plus courte. 30

Figure 2.10. Comparaison sur la base du coût au kg. 31

Figure 2.11. Deux courbes NURBS et les points de contrôle correspondants. 33

Figure 2.12. Forme originale de l'avant-corps de SUBOFF et forme de la méthode de paramétrage NURBS. Les symboles carrés sont les 16 points de données. 33

Figure 2.13. Organigramme de la méthode d'optimisation. 34

Figure 2.14. Schéma séquentiel conventionnel de conception de micro-réseau (révisé). 35

Figure 2.15. Micro-réseau AC proposé. 36

Figure 2.16. Charger des profils. Trois profils sont fournis pour représenter les petits, grands et charges fluctuantes, avec une résolution temporelle de 15 minutes sur une période de 7 jours. ..37

Figure 2.17. Résumé graphique. 38

Figure 2.18. Convergence des 3 algorithmes vers la même configuration. 39

Figure 2.19. Comparaison des 3 algorithmes. 39

Figure 2.20. Illustration of the proposed DRL architecture for intelligent risk optimization of building structures. 40

Chapitre 3 Approche proposée

Figure 3.1. Architecture globale de l'approche proposée. 44

Figure 3.2. Capteur de Code Python pour convertir la bdd de txt vers excel. 50

Figure 3.3. Graphe de base de données 1. 51

Figure 3.4. Capture de la BDD1 Web-Google. 51

Figure 3.5. Capture de fichier Excel qui contient les Nœuds vides. 52

Figure 3.6. Capteur Code Python pour Remplacé les Noueds vides. 52

Figure 3.7. Capture de fichier excel qui contient les Nœuds vides Remplacé. 53

Figure 3.8. Capture de la BDD1 Web-Google Finale. 54

Figure 3.9. Capture de la BDD2 edges. 55

Figure 3.10. Capture de la BDD2 Nodes. 55

Figure 3.11. Capteur de code Python pour modifier la base de données. 57

Figure 3.12. Capture de la BDD2 nodes_updated1.xls. 57

Figure 3.13. Capture de la BDD2 edges_updated1.xls. 58

Figure 3.14. Graphe de base de données 2. 58

Figure 3.15. Capture de la matrice symétrique de temps non orientée. 59

Figure 3.16. Capture de la matrice symétrique de distance non orientée. 59

Figure 3.17. Capteur de code Python de Création base de données. 60

Figure 3.18. Graphe de base de données 3. 60

Figure 3.19. État initial de Jeu d'échecs. 62

Figure 3.20. Graphe de base de données 4. 65

Chapitre 4 Résultats et discussion

Figure 4.1. Comparaison de la distance entre trois algorithmes BDD1. 74

Figure 4.2. Temps d'exécution par couple pour chaque algorithme (30 premiers couples) BDD1. 75

Figure 4.3. Temps d'exécution par couple pour chaque algorithme (30 premiers couples) BDD2. 77

Figure 4.4. Comparaison de la distance entre trois algorithmes BDD2. 78

Figure 4.5. Temps d'exécution par couple pour chaque algorithme (30 paires aléatoires) en utilisant la méthode (A+B) BDD3. 81

Figure 4.6. Comparaison de là (distance + temps) entre trois algorithmes BDD3. 82

Figure 4.7. Temps d'exécution par couple pour chaque algorithme (30 premiers couples) en utilisant la méthode (A*B) BDD3. 84

Figure 4.8. Comparaison de la (distance * temps) entre trois algorithmes BDD3. 85

Figure 4.9. Temps d'exécution par couple pour chaque algorithme (30 premiers couples) en utilisant la méthode (Ax + By) BDD3. 87

Figure 4.10. Comparaison de la (x*distance + y*temps) entre trois algorithmes BDD3. 88

Figure 4.11. Temps d'exécution par couple pour chaque algorithme (30 premier couples) en utilisant la méthode (A+B) BDD4. 91

Figure 4.12. Comparaison de la (arc1+arc2) entre trois algorithmes BDD4. 92

Figure 4.13. Temps d'exécution par couple pour chaque algorithme (30 premier couples) en utilisant la méthode (A*B) BDD4 94

Figure 4.14. Comparaison de la (arc1*arc2) entre trois algorithmes BDD4. 95

Figure 4.15. Temps d'exécution par couple pour chaque algorithme (30 premiers couples) en utilisant la méthode (Ax + By) BDD4. 97

Figure 4.16. Comparaison des résultats (Ax + By) entre trois algorithmes BDD4. 98

Figure 4.17. Capture de la BDD 4.xls. 101

Figure 4.18. Couple des Nœuds de BDD2(couple_noeuds.txt). 101

Figure 4.19. results_genetic_algorithm (A+B).csv. 102

Figure 4.20. Couple des Nœuds de BDD3(couple_noeuds.xml). 102

Liste des tableaux

Chapitre 2 Travaux connexes

Tableau 2.1. Travaux connexes. 41

Chapitre 3 Approche proposée

Tableau 3.1. Comparaison des bases de données. 66

Tableau 3.2. Matrice des couples Source / Destination. 66

Chapitre 4 Résultats et discussion

Tableau 4.1. Comparaison des 3 algorithmes sur les couples choisis au hasard BDD1. 74

Tableau 4.2. Comparaison des 3 algorithmes sur les couples choisis au hasard BDD2. 77

Tableau 4.3. Les résultats après ajustement des paramètres génétiques BDD2. 79

Tableau 4.4. Comparez les trois algorithmes sur des paires sélectionnées aléatoirement en utilisant la méthode (A+B) BDD3. 81

Tableau 4.5. Comparaison des 3 algorithmes sur les couples choisis au hasard en utilisant la méthode (A*B) BDD3. 83

Tableau 4.6. Comparaison des 3 algorithmes sur les couples choisis au hasard en utilisant la méthode (Ax + By) BDD3. 86

Tableau 4.7. Les résultats après ajustement des paramètres génétiques en utilisant la méthode (A + B) BDD3. 89

Tableau 4.8. Comparaison des 3 algorithmes sur les couples choisis en utilisant la méthode (A+B) BDD4. 90

Tableau 4.9. Comparaison des 3 algorithmes sur les couples choisis en utilisant la méthode (A*B) BDD4. 93

Tableau 4.10. Comparaison des 3 algorithmes sur les couples choisis en utilisant la méthode (Ax + By) BDD4. 96

Tableau 4.11. Tableaux de comparaison des résultats pour les bases de données. 99

Tableau 4.12. Complexité de trois algorithmes. 100

Liste des abréviations

ACO : Algorithmes de Colonies de Fourmis

AG : Algorithme génétique

BDD : Base De Données

ChOA : Chimp Optimization Algorithm

CNY : le yuan chinois

CSA: Cyclic Shift Algorithm

DGA: Dijkstra Genetic Algorithm

DM: data mining

DQN: Deep Q-Network

GDCSA: Gradient Descent with Cyclic Shift Algorithm

IA : L'intelligence artificielle

IOGA : Improved Optimization Genetic Algorithm

LAHC: Algorithme Late Acceptance Hill Climbing

PCSA: Probabilistic Counting with Stochastic Averaging

POC : problèmes d'optimisation combinatoire

PSO : Algorithme des Essaims de particules

TE : Temps d'Exécution

Introduction Générale

Introduction générale

L'intelligence artificielle (IA) est un processus d'imitation de l'intelligence humaine qui repose sur la création et l'application d'algorithmes exécutés dans un environnement informatique dynamique. Son but est de permettre à des ordinateurs de penser et d'agir comme des êtres humains.

Avec l'avancé technologique spectaculaire de l'Intelligence Artificielle (IA) dans plusieurs domaines, explorer et analyser le big data et devenu une tâche essentielle. Le développement du Web a été caractérisé par une croissance significative des services offerts, entraînant une augmentation importante du nombre d'utilisateurs aux contraintes et préférences diverses. L'analyse de cette abondance d'informations revêt une importance cruciale pour en extraire des données pertinentes. Le principal défi est donc de trouver le service qui répond le mieux aux besoins et aux préférences spécifiques de chaque utilisateur.

De nombreuses techniques d'IA ont été développées pour simplifier la recherche et l'extraction d'informations pertinentes Les techniques d'optimisation et de recommandation ont connu un grand intérêt raison de leur bonne performance sur les grands volumes de données et leur capacité d'analyse et d'extraction d'informations pertinentes afin de l'adapter aux attentes des utilisateurs.

Les techniques d'optimisation consistent en un ensemble de méthodes de résolution de problèmes d'optimisation combinatoire (POC) qui se subdivisent en trois groupes : les méthodes exactes, les méthodes heuristiques et les méthodes hybrides. Les algorithmes exacts garantissent de trouver la solution optimale et de prouver son optimalité pour toutes les instances d'un problème. Par contre, le temps nécessaire pour trouver la solution optimale d'un POC augmente en général fortement avec la taille du problème. De plus, pour certains problèmes, la consommation de mémoire de ces algorithmes peut-être très grande et peut parfois entraîner l'arrêt prématuré de l'application informatique.

Les méta-heuristiques gagnent maintenant en popularité, car les meilleurs résultats trouvés pour plusieurs POC ont étés obtenus avec des algorithmes hybrides qui impliquent une combinaison de plusieurs méta-heuristiques ou une combinaison d'une méthode exacte Et d'une méta-heuristique. L'utilisation des méthodes heuristiques, Permet d'obtenir des solutions de bonnes qualités en un temps de résolution raisonnable. (HAYI, 2023).

1. Problématiques

L'optimisation des grandes bases de données se heurte à des défis de performance, surtout lorsqu'il s'agit de résoudre des problèmes d'optimisation combinatoire, tels que celui du plus court chemin dans un graphe. Ces défis sont multifacettes, allant de la gestion efficace des ressources informatiques à la garantie de résultats optimaux. Comprendre les avantages et les inconvénients des différents types d'algorithmes d'optimisation est crucial pour relever ces défis. Dans le contexte des problèmes uni-critères ou multicritères, les algorithmes doivent être évalués en fonction de leur capacité à produire des solutions efficaces et adaptées à la situation donnée. Comprendre ces nuances est essentiel pour choisir l'algorithme le plus approprié pour un contexte spécifique et pour garantir des performances optimales dans la résolution de problèmes d'optimisation combinatoire. Les questions spécifiques à résoudre incluent :

- Quels sont les avantages et les inconvénients de chaque type d'algorithme en termes de performance ?
- Comment ces algorithmes se comparent-ils lorsqu'ils sont appliqués à des problèmes d'optimisation uni-critère par rapport à des problèmes multicritères ?
- Quelles sont les meilleures pratiques pour sélectionner un algorithme d'optimisation en fonction du type de problème et de la nature des données ?

2. Les objectifs de cette étude

Les objectifs de cette étude sont d'analyser et de comparer les performances de différents algorithmes d'optimisation sur des bases de données uni-critère et multicritère. Il s'agit également d'identifier les avantages, les limitations et les conditions d'utilisation optimale pour chaque type d'algorithme en fonction des caractéristiques des données et des problèmes d'optimisation. Ces objectifs permettront de mieux comprendre comment choisir et appliquer les algorithmes d'optimisation les plus appropriés pour améliorer les performances des bases de données dans différents contextes.

3. Organisation du mémoire

Ce mémoire est organisé en quatre chapitres présentés successivement :

- **Le premier chapitre État de l'art :** nous présentons un aperçu général sur l'optimisation, la théorie des graphes et nous introduisons ensuite les graphes orientés et non orientés, ainsi que nous présentons les différents algorithmes connus d'optimisation.

- **Le deuxième chapitre Travaux connexes :** Le deuxième chapitre contient les travaux connexes récents qui nous permet de faire une comparaison générale entre les différents travaux, d'après les bases de données utilisées ou les techniques et les algorithmes et même pour voir les résultats obtenus avec leurs analyses.
- **Le troisième chapitre Approche proposée** : Ce chapitre contient les détails de l'approche proposée et les Environnement et outils de développement, ainsi que nous présentons Description et prétraitement des bases de données et les algorithmes utilisés.
- **Chapitre quatre Résultats et discussion :** Ce chapitre présente les paramètres de l'algorithme génétique, l'analyse et l'évaluation des résultats obtenus, ainsi que la comparaison et la discussion.

Nous terminerons notre mémoire par une conclusion générale qui résume notre contribution et quelques perspectives à notre travail.

Chapitre 1

État de l'art

1.1. Introduction

L'optimisation dans le Big Data et devenue un défi majeur dans la recherche et est largement utilisée dans divers domaines. Les réseaux complexes tels que les réseaux de transport, de communication, de distribution et Internet suscitent de plus en plus d'attention pour diverses applications en science et en ingénierie, notamment dans le domaine de l'optimisation.

Le problème d'optimisation combinatoire est caractérisé par une fonction d'évaluation de la qualité ou du degré de chaque solution. Avec la croissance massive de l'information, il est crucial de développer des approches efficaces pour résoudre ces problèmes d'optimisation et tirer le meilleur parti des données disponibles.

Face à de nouveaux problèmes complexes d'optimisation, il est primordial d'utiliser des techniques méta-heuristiques (Nasr, 2020).La recherche en méta-heuristique est bien connue dans le domaine de l'intelligence artificielle et à également été très active au cours des dernières décennies. Combiner une méta-heuristique avec d'autres techniques d'optimisation, peut fournir une flexibilité avancée et un comportement plus efficace face à des problèmes à grande échelle et réels (Nasr, 2020).

De nombreux algorithmes d'optimisation ont été proposés dans la littérature, et différentes applications telles que celles liées aux transports exigent l'utilisation d'une méta-heuristique du plus court chemin plutôt que l'un des algorithmes standards existants. Le problème du plus court chemin est un problème d'optimisation combinatoire qui a fait l'objet de nombreuses études.

Réseaux informatiques où les sommets représentent des ordinateurs et les arêtes les connexions physiques, Graphe du web, où les sommets correspondent aux pages web et les arcs représentent les liens hypertextes d'une page à une autre, Réseaux de transports de données (téléphonie, wifi, réseaux informatiques, etc.), Et bien d'autres encore.

La théorie des graphes est l'un des paradigmes utilisés pour modéliser ce type de problème. Le graphe, en tant que réseau complexe, permet de décrire les relations Entre un ensemble de nœuds interconnectés (entités). La taille du graphe étudiée dans notre travail est significative,

Rendant ainsi les algorithmes classiques inappropriés pour des graphes de plus grande envergure. L'algorithme heuristique visant à trouver le chemin le plus court doit être optimal et présenter une efficacité notable en termes de rapidité et de pertinence des chemins identifiés.

Dans ce chapitre, nous présentons tout d'abord l'optimisation et son historique ainsi que ses buts.

1.2. Qu'est-ce qu'un problème d'optimisation ?

Il existe deux types de méthodes d'optimisation, la première est l'optimisation mono-objectif, qui se base sur la minimisation (ou la maximisation) d'une seule fonction objective où le but est de trouver la meilleure solution appelée solution optimale qui est facilement définie suivant une seule performance du problème étudié d'une part. d'autre part, l'optimisation multi-objectif optimise simultanément plusieurs fonctions objectives qui sont souvent contradictoires, on cherche de trouver la meilleure solution suivant un ensemble de performance du problème (Fadel, 2019).

1.2.1. Définition de l'optimisation

L'optimisation est une technique mathématique utilisée pour trouver un minimum ou un maximum global permettant d'optimiser une solution à un problème donné. Optimiser, c'est-à-dire chercher rapidement la meilleure solution avec un minimum de couts. (Ketfi, 2016). Dans son sens général, vise à trouver la meilleure solution possible parmi un ensemble de solutions alternatives. Cela peut signifier maximiser un bénéfice, minimiser un coût, ou atteindre un objectif spécifique en respectant certaines contraintes. On la retrouve dans de nombreux domaines :

- **Mathématiques**

L'optimisation mathématique est une branche des mathématiques qui s'intéresse à la formalisation et la résolution de problèmes d'optimisation. Elle propose des algorithmes et des techniques pour trouver des solutions optimales à des problèmes dits "combinatoires" (avec un nombre fini de solutions) ou "continus" (avec un nombre infini de solutions).

- **Informatique**

En informatique, l'optimisation est utilisée pour concevoir des algorithmes efficaces qui trouvent des solutions optimales à des problèmes complexes, comme la planification de tâches, la gestion de ressources ou l'allocation de mémoire.

- **Ingénierie**

Dans le domaine de l'ingénierie, l'optimisation est utilisée pour concevoir des systèmes efficaces et performants, en minimisant les coûts et en maximisant les performances.

1.2.2. Concepts principaux en optimisation

A. Objectif Il s'agit du critère que l'on souhaite optimiser (maximiser ou minimiser).

B. Fonction objective C'est la fonction qui modélise le but à atteindre dans le problème d'optimisation sur l'ensemble des critères, elle est appelée critère d'optimisation ou fonction d'adaptation. Il s'agit de la fonction qui doit être optimisée.

C. Variables Ce sont les paramètres du problème que l'on peut modifier pour influencer l'objectif.

D. Contraintes Ce sont les limitations qui doivent être respectées lors de la recherche de la solution optimale.

E. Vecteur de décision C'est le vecteur qui correspond à l'ensemble des variables du problème.

F. Critère de décision est une variable du problème ou une combinaison de variables qui est utilisée pour évaluer les solutions potentielles et déterminer la meilleure solution parmi celles-ci.

G. Solution optimale C'est la solution qui satisfait toutes les contraintes et qui atteint la meilleure valeur possible pour l'objectif.

H. Espace de recherche C'est l'ensemble des valeurs pouvant être prise par les variables.

I. Espace réalisable Correspond à l'ensemble des valeurs des variables satisfaisants les contraintes. (Smairi, 2013).

1.2.3. Les étapes clés de l'optimisation

1.2.3.1. Modélisation

La modélisation d'optimisation consiste à décrire formellement un problème sous forme mathématique, de manière à ce qu'il puisse être résolu à l'aide d'algorithmes d'optimisation. Cette modélisation est cruciale pour formuler de manière claire et précise le problème à résoudre, en identifiant les variables de décision, la fonction objective à optimiser et les contraintes associées.

1.2.3.2. Analyse

L'analyse des données (aussi appelée analyse exploratoire des données) est une partie très importante dans un processus d'optimisation. La phase d'analyse a pour objectif de supprimer des mots vides et des données non importantes du data-set, afin de bien l'organiser et le faire passer à

la prochaine étape. Cette partie n´nécessite l'utilisation de quelques techniques et quelques scripts de programmation, c'est une partie compliquée qui n´nécessite beaucoup de temps surtout dans le cas des bases de données volumineuses. (HAYI, 2023).

1.2.3.3. Résolution des problèmes

La résolution des problèmes d'optimisation est un domaine passionnant et actif, avec une multitude de techniques et d'approches. Le choix de la méthode la plus adaptée dépend du type de problème, des ressources disponibles et de la précision requise. N'hésitez pas à me poser des questions plus spécifiques sur des techniques particulières ou des applications dans des domaines concrets.

1.2.4. Historique d'optimisation

Vers 300 ans avant notre ère, le mathématicien de la Grèce antique Euclide d'Alexandrie a formulé les premiers problèmes d'optimisation. Il a proposé une méthode permettant de trouver le plus court chemin entre deux points A et C en passant par un point de la droite, ce chemin est obtenu lorsque l'angle d'incidence est égalé l'angle réfléchi. La figure (1.1) suivante représente une explication de cette formulation. (Pape, 1974).

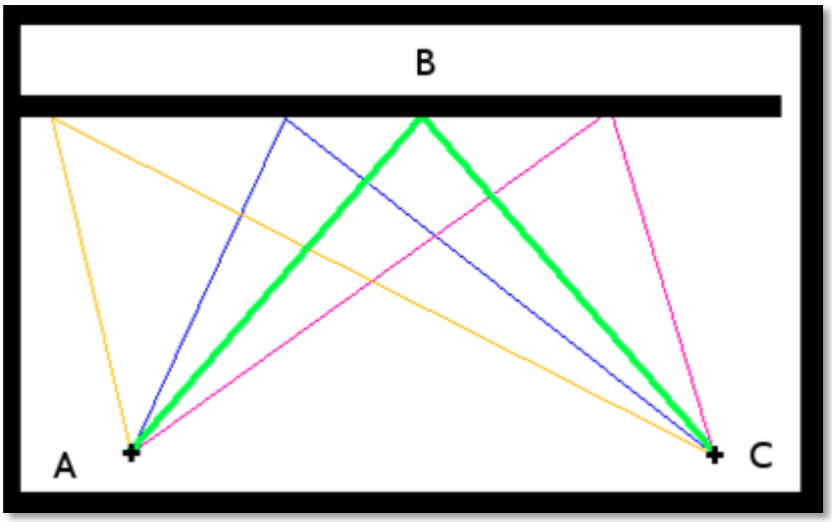


Figure 1.1. Explication du premier problème d'optimisation.

En deuxième siècle avant notre ère, le mathématicien Euclide venait de formuler le premier problème d'optimisation dans son ouvrage historique Eléments. Des ´ années plus tard, Héron d'Alexandrie (ingénieur, mécanicien et mathématicien grec) établissait ses premiers énoncés sur le principe du plus court chemin**.** (Maguire S. a., 2006).

Au XVIIe siècle, Newton prouvait le besoin d'établir des techniques d'optimisation pour l'amélioration du calcul différentiel, et après plusieurs tentatives, il développa une méthode itérative afin d'augmenter la probabilité d'avoir la meilleure valeur approximative du calcul différentiel. Cette méthode itérative a permis d'améliorer les résultats et les fonctionnements. (Bellman, 1958).

Depuis, l'optimisation a connu un d´développement fulgurant et a commencé à être utilisée dans plusieurs domaines. (Polyak, 2007). L'optimisation est l'outil privilégié pour identifier les conditions d'opérations les plus souhaitables ou le programme des opérations le plus rentable menant à la meilleure performance, et ce dans le respect des contraintes. Des exemples de problèmes fréquents sont : mélange de produits à coût minimal, confection d'horaire de production, maximisation de rendement, minimisation des pertes et augmentation de débit dans un réseau. La science de l'optimisation est généralement comprise et pratiquée par les mathématiciens, alors que les problèmes d'optimisation sont issus des activités des ingénieurs.

1.3. Optimisation Uni-critère

L'optimisation uni-critère, c'est-dire celle qui ne prends en compte qu'un seul critère, n'est en fait, en général, qu'une simplification des problèmes qui passent dans la vie courante. En effet, la plupart des problèmes de décision auxquels nous sommes confrontés sont multicritères. Dans les problèmes de choix de fournisseurs, par exemple, à moins que de travailler dans le cadre d'une adjudication où le marché serait attribué au moins-disant et ce, sans autre forme de procès, le problème auquel est confronté le décideur comporte plusieurs critères. Certains étant à maximiser, d'autres à minimaliser. Le présent article traite de ce qui est de la problématique multicritère pour le choix d'investissements. Ce qui a fait le succès d'une entreprise jusqu'à présent peut ne plus suffire pour répondre aux défis d'aujourd'hui. L'adaptation des entreprises aux changements environnementaux reposent sur plusieurs axes dont la théorie de la décision.

1.4. Optimisation Multicritère

L'optimisation multicritère consiste à choisir, en présence de critères multiples, une des alternative(s) parmi un nombre infini d'alternatives qui varient généralement dans un domaine continu. Depuis une trentaine d'années, le domaine de l'optimisation multicritère connaît une évolution importante. Cette évolution s'est traduite par le développement d'un grand nombre de méthodes. La multitude des méthodes d'optimisation multicritère est perçue comme une richesse incontestable de ce domaine. D'ailleurs, certains la justifient par la diversité des problèmes ainsi que par l'existence de différentes approches de résolution possibles et légitimes de ces problèmes.

Cependant, ce phénomène révèle aussi des faiblesses certaines. En effet, la plupart de ces méthodes manquent de fondements axiomatisés, et il est difficile de choisir la méthode à appliquer face à une situation donnée. Le travail présenté dans ce mémoire propose une approche axiomatisée d'optimisation multicritère. Cette approche est fondée sur des concepts tels que l'efficacité partielle qui sont motivés et justifiés par des interprétations intelligibles. Elle est Robuste par rapport aux paramètres utilisés, opérationnelle, et évolutive. Elle peut être utilisée dans la résolution de différentes situations multicritères tels que les problèmes comportant des critères nombreux et incommensurables et les problèmes de décisions publiques. (Othmani, 1998).

1.5. Théorie des graphes et Optimisation

La théorie des graphes est un domaine des mathématiques et de l'informatique qui étudie les structures appelées graphes. Voici quelques éléments clés de cette théorie :

1.5.1. Graphes

Un graphe est en effet un ensemble de composants connectés entre eux. Ces composants sont représentés par des sommets ou nœuds, qui sont des points géométriques. Les sommets sont reliés par des arêtes, également appelées liens ou courbes. Les arêtes peuvent être asymétriques (orientées) ou non orientées. Il existe deux types de graphes :

A. Graphe orienté

Un graphe orienté est un modèle mathématique qui permet de représenter les relations directionnelles entre des éléments (nœuds) à l'aide d'arcs orientés.

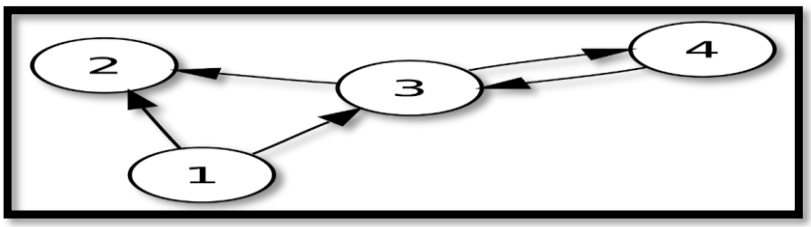


Figure 1.2. Graphe orienté

B. Graphe non-orienté

Une représentation de graphe non orienté consiste en une liste de nœuds interconnectés par des arêtes, mais une seule arête indique la relation réciproque entre deux nœuds, ce qui exclut la présence de deux arêtes entre deux nœuds.

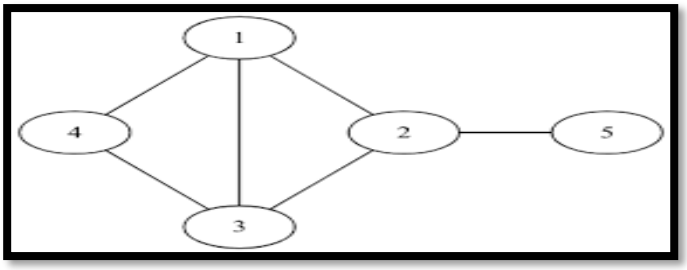


Figure 1.3. Graphe non-orienté.

1.6. Quelques Algorithmes d'optimisation

Ils existent plusieurs algorithmes exacts, hybrides, heuristiques ou même métaheuristiques utilises dans le domaine de l'optimisation. Les différents algorithmes existants ont les mêmes objectifs, à savoir optimiser rapidement les résultats d'un problème donnée. Le concept de ces algorithmes change selon le domaine de l'utilisation ainsi que la taille des choix qu'offre un problème donné. Chaque algorithme a ses propres méthodes et techniques et cette diversité permet de créer une concurrence de qualité entre les algorithmes. Nous présentons ici quelques algorithmes connus et très utilisés dans le domaine d'optimisation :

1.6.1. Algorithme de Dijkstra

L'algorithme de Dijkstra est l'algorithme le plus populaire dans le problème du plus court chemin. Il est souvent utilisé pour orienter des graphes pondérés par des réels positifs. Très ancien mais très fonctionnel, il a été mis en place en 1959 par l'informaticien néerlandais Edsger Dijkstra. Cet algorithme tente toujours à trouver le chemin le plus court d'un nœud à tous les autres nœuds. Il prend un nœud initial, et cherche le chemin le plus court vers tous les autres nœuds. Dans une seconde itération, en fonction des nœuds visités par le premier nœud, il choisit le chemin le plus court du nœud précédent et répète la première étape, et ainsi de suite jusqu' à ce qu'il active tous les nœuds du graphe (Singh, 2018), (Grabener, 2010) . L'algorithme 1 représente l'algorithme de Dijkstra en détail.

Algorithm 1 Dijkstra

$x \leftarrow \emptyset$
$y \leftarrow \{v\}$
$d[v] \leftarrow 0$
while $y \neq \emptyset$ **do**
  choisir u tel que : $u \in y \wedge (\forall z : z \in y : d[u] \leq d[z])$
  $x \leftarrow x \cup \{u\}$
  $y \leftarrow y \backslash \{u\}$
  for $n : n \in s(u)$ **do**
    if $n \in x$ **then**
      skip
    else if $n \in y$ **then**
      $d[n] \leftarrow min(d[n], d[u] + len(u, n))$
    else
      $y \leftarrow y \cup \{n\}$
      $d[n] \leftarrow d[u] + len(u, n)$
    end if
  end for
end while

Figure 1.4. Algorithme de Dijkstra.

1.6.2. Algorithme Floyd-Warshall

Floyd-Warshall ou aussi Roy-Floyd-Warshall est un algorithme qui détermine

Les distances des plus courts chemins entre n'importe quelles paires de sommets dans un graphe orienté et pondéré. Il a été décrit par Bernard Roy en 1959 (Bernard, 1959). Ce dernier est l'un des algorithmes connus dans le domaine de l'optimisation, son fonctionnement est facile à comprendre. Nous prenons tous les arcs du graphe, chaque arc représente une relation entre deux nœuds de type k=0, et nous ajoutons de nouvelles relations par la combinaison des arcs entre eux, et ainsi de suite (Xing, 2019). La figure (1.6) explique mieux le fonctionnement de l'algorithme Floyd-Warshall.

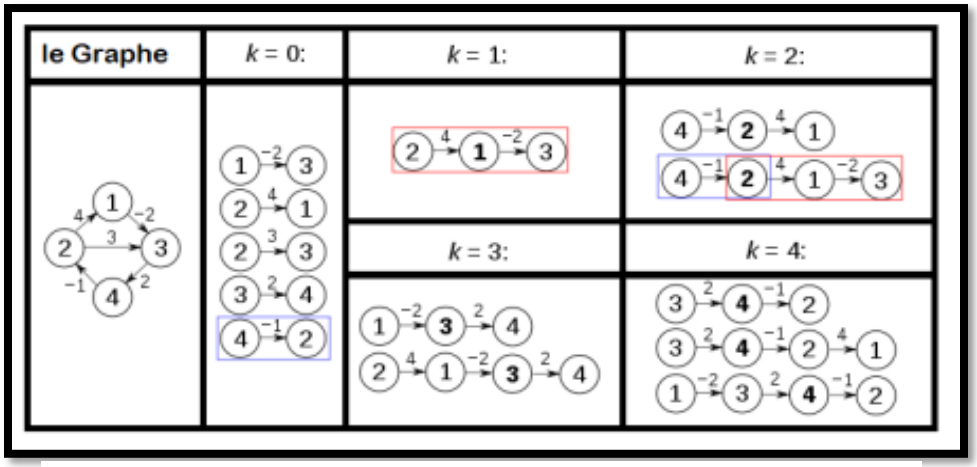


Figure 1.5. Exemple sur l'algorithme de Floyd-Warshall.

Algorithm 1: Pseudocode of Floyd-Warshall Algorithm

```
Data: A directed weighted graph G(V, E)
Result: Shortest path between each pair of vertices in G
for each d ∈ V do
    distance[d][d] ← 0;
end
for each edge (s, p) ∈ E do
    distance[s][p] ← weight(s, p);
end
n = cardinality(V);
for k = 1 to n do
    for i = 1 to n do
        for j = 1 to n do
            if distance[i][j] > distance[i][k] + distance[k][j] then
                distance[i][j] ← distance[i][k] + distance[k][j];
            end
        end
    end
end
```

Figure 1.6. L'algorithme de Floyd-Warshall.

1.6.3. Algorithmes Méta Heuristiques

Une méta-heuristique est une méthode générique pour la résolution des problèmes combinatoires. La résolution de ces problèmes nécessite l'examen d'un très grand nombre (exponentiel) de combinaisons. Plusieurs chercheurs ont déjà été confrontés à ce phénomène d'explosion combinatoire qui transforme un problème apparemment très simple en un véritable casse-tête d`es lors que nous augmentons la taille du problème à résoudre. C'est le cas par exemple quand on cherche à concevoir un emploi du temps. S'il y a peu de cours à planifier, le nombre de combinaisons à explorer est faible et le problème est très rapidement Résolu.

Cependant, l'ajout de quelques cours seulement peut augmenter Considérablement le nombre de combinaisons à explorer de sorte que le temps de résolution devient excessivement long (Hussain, 2019). Le but d'une méta-heuristique, est de réussir à trouver un optimum global. Pour cela, l'idée est à la fois de parcourir l'espace de recherche, et d'explorer les zones qui paraissent prometteuses, par contre les heuristiques sont des méthodes de résolution purement algorithmiques qui permettent d'obtenir des solutions à n'importe quel problème décisionnel rapidement. Les méta-heuristiques contiennent plusieurs algorithmes, nous présentons ci-après les plus connus :

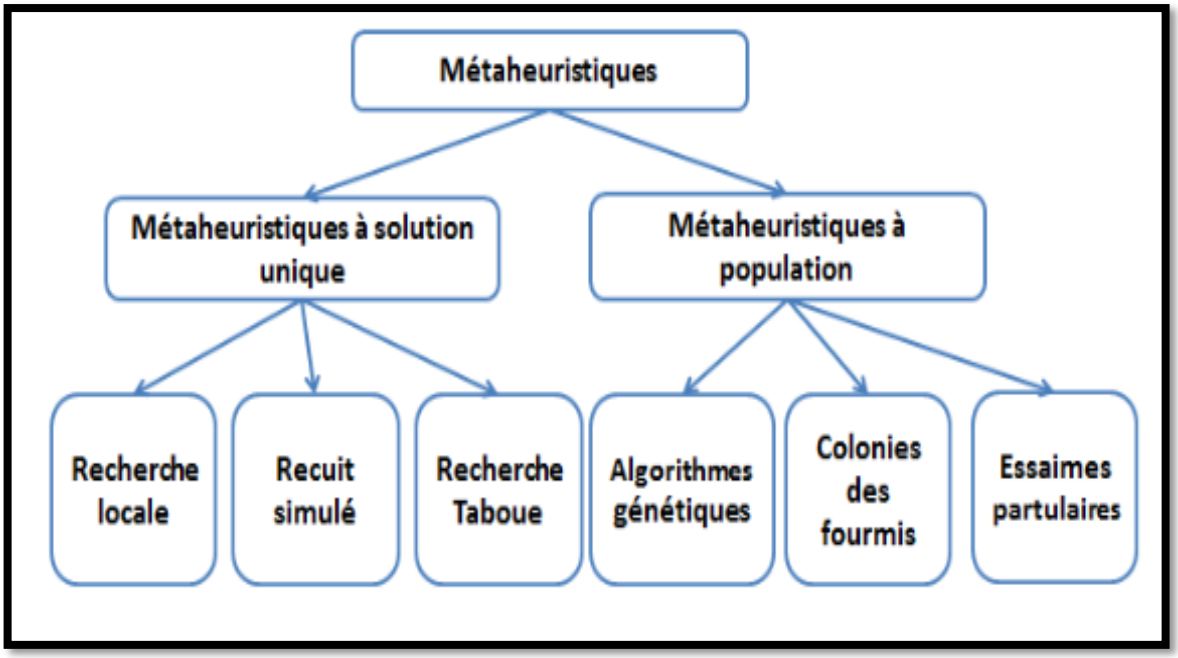


Figure 1.7. Classification des métaheuristiques.

1.6.3.1. Algorithme Génétique

Un algorithme génétique est un algorithme de recherche et d'optimisation aléatoire et évolutif qui imite l'évolution biologique naturelle. Il a été appliqué pour résoudre divers problèmes dans différents domaines d'activité humaine et utilisé pour obtenir une solution optimale dans un délai raisonnable. Il utilise une sélection naturelle (aléatoire) pour avoir une population. Ce dernier évoluera après chaque itération, et lorsque l'algorithme atteint une condition d'arrêt, il retournera la meilleure solution qui existe dans la population comme solution finale au problème (Hamed, 2010), (Rahmani Hosseinabadi, 2019).

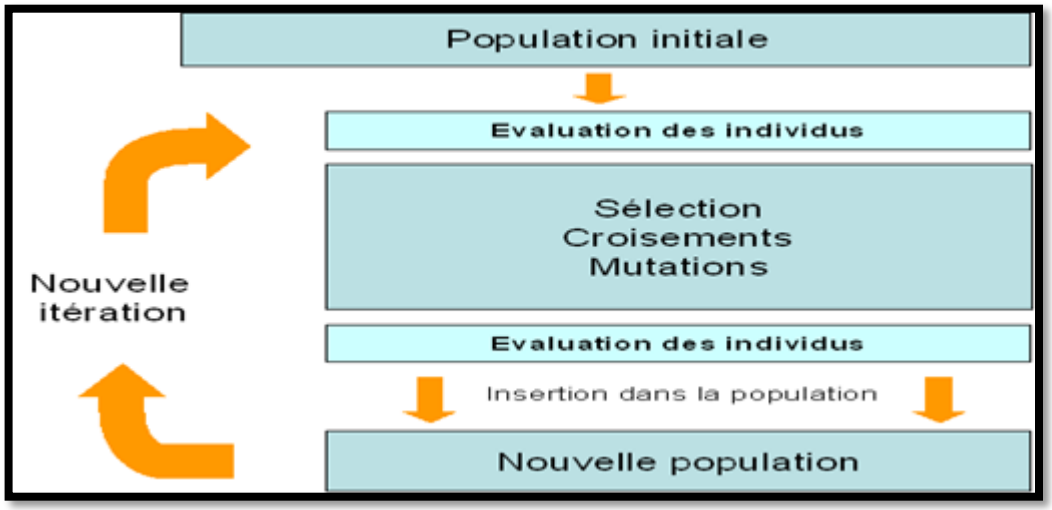


Figure 1.8. Algorithme Génétique.

1.6.3.2. Algorithme A*

L'algorithme de recherche **A*** (qui se prononce **A étoile**, ou ***A star*** à l'anglaise) est un algorithme de recherche de chemin dans un graphe entre un nœud initial et un nœud final tous deux donnés. De par sa simplicité il est souvent exhibé comme un exemple typique d'algorithme de planification, un domaine de l'intelligence artificielle. L'algorithme A* a été créé pour que la première solution trouvée soit l'une des meilleures, c'est pourquoi il est célèbre dans des applications comme les jeux vidéo privilégiant le temps de calcul à l'exactitude des résultats. L'algorithme A* est considéré comme l'un des heuristiques les plus populaires en raison de sa complétude, de son optimalité et de son efficacité optimale (H. Yusnita, 2020), (Kacem, 2012).

À partir de cette origine : (HAYI, 2023).Se constitue par un ensemble de procédés :

- Lire le graphe.
- Récupérer les chemins existants.
- Vérifier si un emplacement valide existe.
- Ignorer la position du nœud si elle est invalide.
- Ajouter à la liste des nœuds enfants valides pour le parent sélectionné.
- Obtenir les meilleurs résultats.

1.6.3.3.Algorithme Gloutons

Un algorithme glouton (en anglais greedy algorithmes) est un algorithme qui effectue le meilleur choix possible à chaque instant. Il fonctionne donc étape par étape sans avoir à revenir en arrière ou à prédire l'étape suivante. L'algorithme glouton cherche une solution optimale locale à chaque étape, dans le but de trouver la meilleure solution optimale. Les algorithmes gloutons sont parfois appel´es algorithmes gourmands. Ils sont très utilisés dans les problèmes de voyageur de commerce, l'arbre couvrant minimal, coloration de graphe et même dans le problème du rendu de monnaie. Par exemple, si nous souhaitons résoudre Le problème du rendu de monnaies par l'algorithme glouton, ce dernier doit répéter le choix de la pièce de plus grande valeur qui ne d´passe pas la somme restante pour donner une somme avec le moins possible de pièces. Il est appel´e une heuristique gloutonne dans les cas où l'algorithme ne fournit pas systématiquement la solution optimale (Cerrone, 2017).

1.6.3.4.Recuit Simulé

Le recuit simulé est un algorithme proposé par kirkpatrick en 1983 (Kirkpatrick, 1983) .

- La méthode du Recuit Simulé repose sur une analogie avec la thermodynamique où la fonction à minimiser est l'énergie du système.
- Le recuit est un processus physique de chauffage. Un système physique porté à une température assez élevée devient liquide, et dans ce cas le degré de liberté des atomes qui le composent augmente.
- Inversement lorsque l'on baisse la température le degré de liberté diminue jusqu'à obtenir un solide. Suivant la façon dont on diminue la température on obtient des configurations d'énergie différentes :

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

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

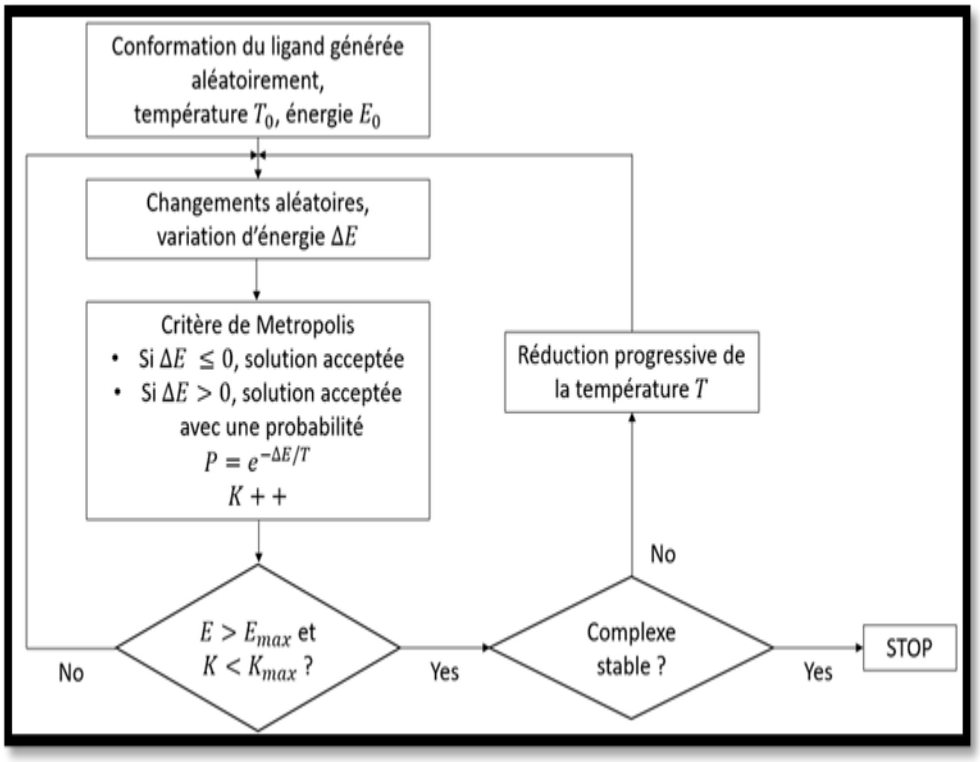


Figure 1.9. Organigramme de Recuit simulé.

1.6.3.5.Algorithme de Recherche Tabou

L'algorithme de recherche tabou a été proposé la première fois en 1986 par Glover (F. Glover, 1986). L'idée est simple, l'algorithme ajoute chaque solution déjà visitée `a une liste noire pour qu'il ne visite pas cette solution une autre fois et pour minimiser le temps de réponse. Dans les années 90, l'algorithme de recherche tabou est devenu populaire. A la manière d'une méthode par descente, le voisinage est exploré de manière complète et d`es qu'une meilleure solution est trouvée, elle remplace la solution courante. Le risque dans ce type d'approche est l'apparition de cycles. C'est pour cela que la recherche Tabou possède une liste noire ou taboue afin de mémoriser les solutions déjà effectuées. Mais plus cette liste est longue, plus elle prend de place en mémoire, et de même, un temps important pour vérifier un nouveau voisin s'il existe dans la liste taboue ou non.

Pour cela, la taille de la liste taboue est devenue fixe et elle garde seulement les dernières solutions visitées. Ainsi, une solution est acceptée si le mouvement qui l'a générée n'est pas tabou (Boisson, 2008), (Lebbah, 2015)

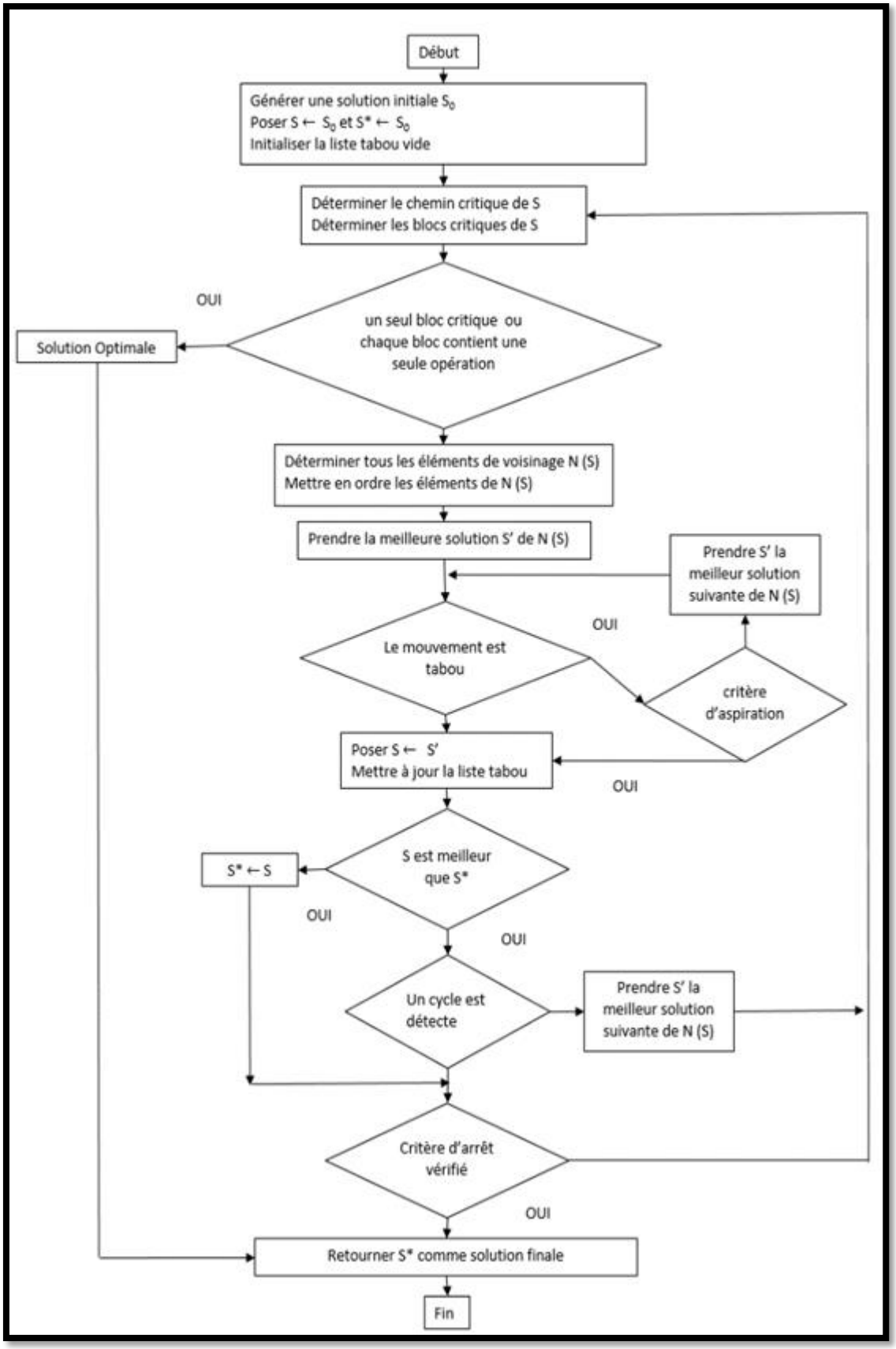


Figure 1.10. Organigramme de Recherche Tabou.

1.6.3.6.Algorithmes de Colonies de Fourmis (ACO)

L'algorithme de colonies de fourmis fait partie des algorithmes d'intelligence en essaim, il est inspiré du comportement des fourmis lorsqu'elles cherchent de la nourriture. (Deneubourg, 1990).

Initialement proposé par Marco Dorigo et al. Dans les années 1990, pour la recherche de chemins optimaux dans un graphe, basant sur travaux de Deneubourg sur les fourmis (1983).

S'inspire du comportement des fourmis recherchant un chemin entre leur colonie et une source de nourriture.

Cette méta-heuristique a été utilisée pour résoudre le problème du voyageur de commerce. Elle s'est popularisée, puis a été l'objet d'améliorations dès 1995 et a été appliquée avec succès à d'autres problèmes d'optimisation combinatoire.

❖ Comportement des fourmis

Les fourmis utilisent l'environnement comme support de communication : elles échangent indirectement de l'information en déposant des phéromones, le tout décrivant l'état de leur « travail ». L'information échangée a une portée locale, seule une fourmi située à l'endroit où les phéromones ont été déposées y a accès. Un modèle expliquant ce comportement est le suivant :

1. Une fourmi parcourt plus ou moins au hasard l'environnement autour de la colonie

2. Si celle-ci découvre une source de nourriture, elle rentre directement au nid, en laissant sur son chemin une piste de phéromones

3. Ces phéromones étant attractives, les fourmis passant à proximité vont avoir tendance à suivre, de façon plus ou moins directe, cette piste. En revenant au nid, ces mêmes fourmis vont renforcer la piste.

4. Si deux pistes sont possibles pour atteindre la même source de nourriture, celle étant la plus courte sera, dans le même temps, parcourue par plus de fourmis que la longue piste.

5. La piste courte sera donc de plus en plus renforcée, et donc de plus en plus attractive. La longue piste, elle, finira par disparaître, les phéromones étant volatiles.

6. À terme, l'ensemble des fourmis a donc déterminé et « choisi » la piste la plus courte.

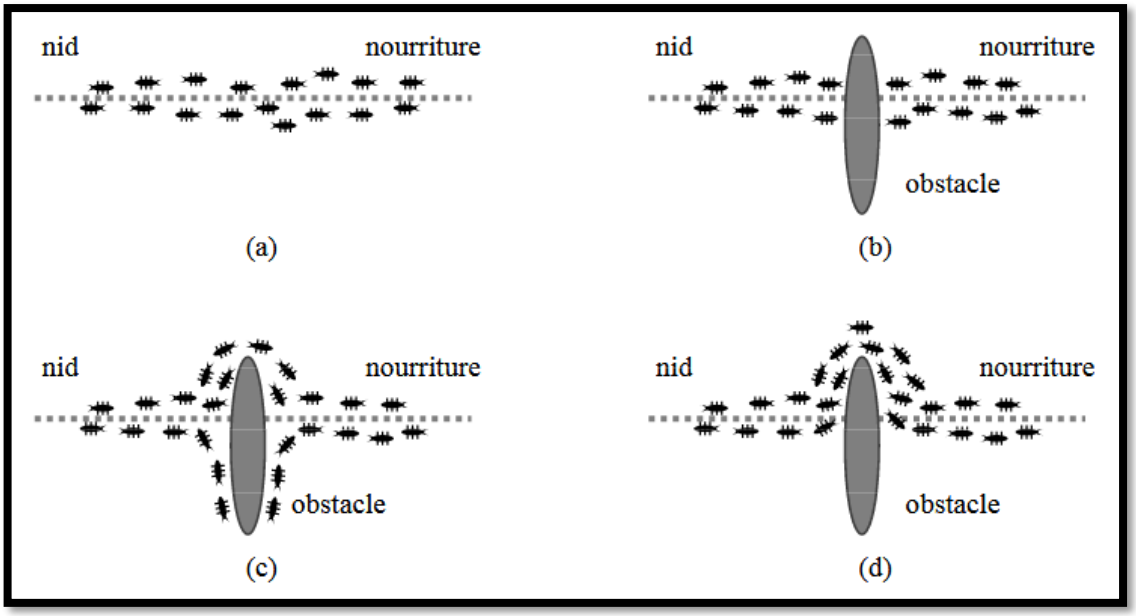


Figure 1.11. Détermination du plus court chemin par une colonie de fourmis.

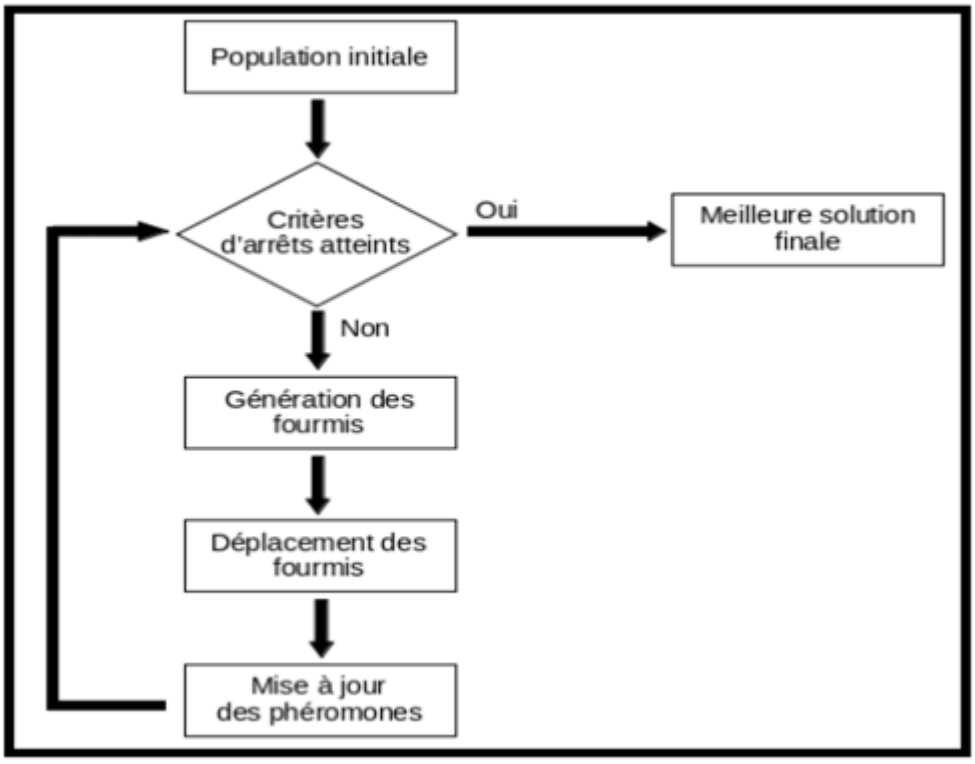


Figure 1.12. Organigramme de colonies de fourmis.

1.6.3.7. Algorithme des Essaims de particules (PSO)

L'algorithme des essaims de particules ou « Particle Swarm Optimization » (PSO) est une méthode d'optimisation proposée en 1995 par :

- Russel Eberhart (ingénieur)
- Jim Kennedy (socio-psychologue)

Inspiré par les interactions sociales ou les déplacements d'un groupe d'animaux (nuée d'oiseaux ou banc de poissons)

Pour appliquer la PSO il faut définir un espace de recherche constitué de particules et une fonction objective à optimiser. Le principe de l'algorithme est de déplacer ces particules afin qu'elles trouvent l'optimum.

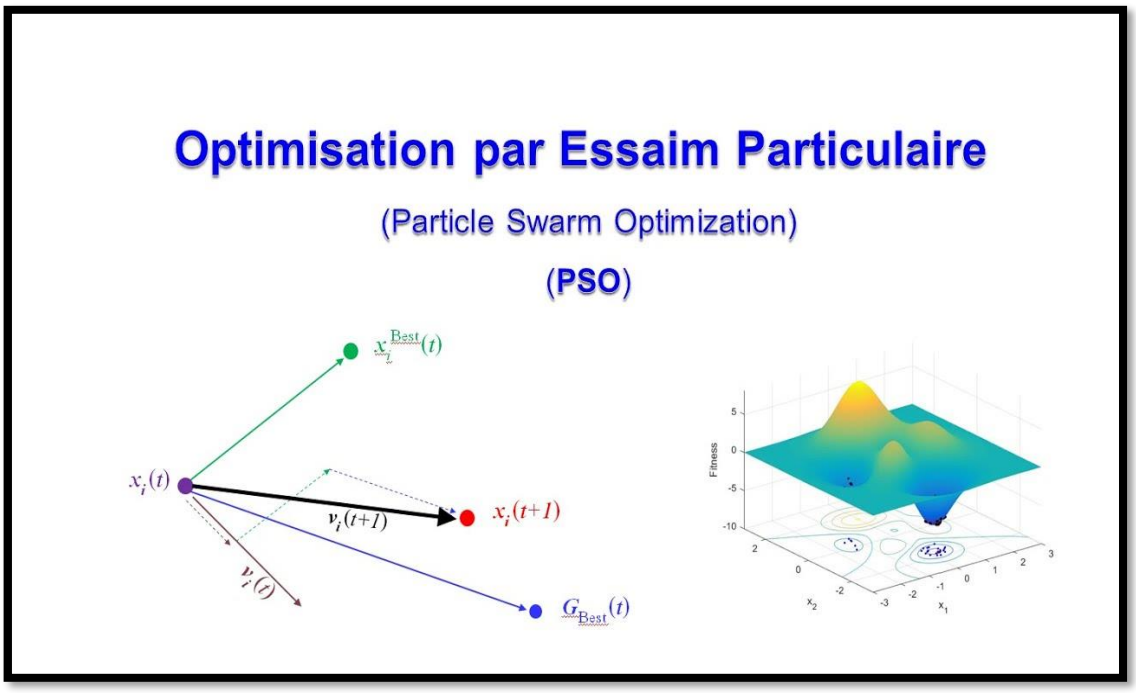


Figure 1.13. Optimisation par Essaim Particulaire (Particle Swarm Optimization) PSO.

1.6.3.8. Algorithme des Lucioles (Firefly Algorithm)

L'algorithme des lucioles est une méta-heuristique à base de population développée par Xin-She Yang à l'université de Cambridge en 2008.

L'algorithme est inspiré par l'atténuation de la lumière sur la distance et l'attraction mutuelle entre les lucioles.

Les lucioles (en anglais firefly) sont de petits coléoptères ailés, capables de produire une lumière clignotante froide pour une attraction mutuelle.

L'algorithme est basé sur les essaims de lucioles (fireflies). Il utilise les trois règles idéalisées suivantes :

1) Toutes les lucioles sont unisexes, cela signifie qu'une luciole est attirée par une autre, indépendamment de son sexe.
2) L'attractivité et la luminosité sont proportionnelles entre elles, de sorte que, pour deux lucioles clignotantes, la moins lumineuse se déplacera vers la plus lumineuse. L'attractivité et la luminosité diminuent à l'augmentation de leur distance. S'il n'y a pas une luciole plus brillante que l'autre, elles vont se placer de manière aléatoire.
3) La luminosité d'une luciole est déterminée par le point de vue de la fonction objective à optimiser. Pour un problème de maximisation, la luminosité est simplement proportionnelle à la valeur de la fonction objective.

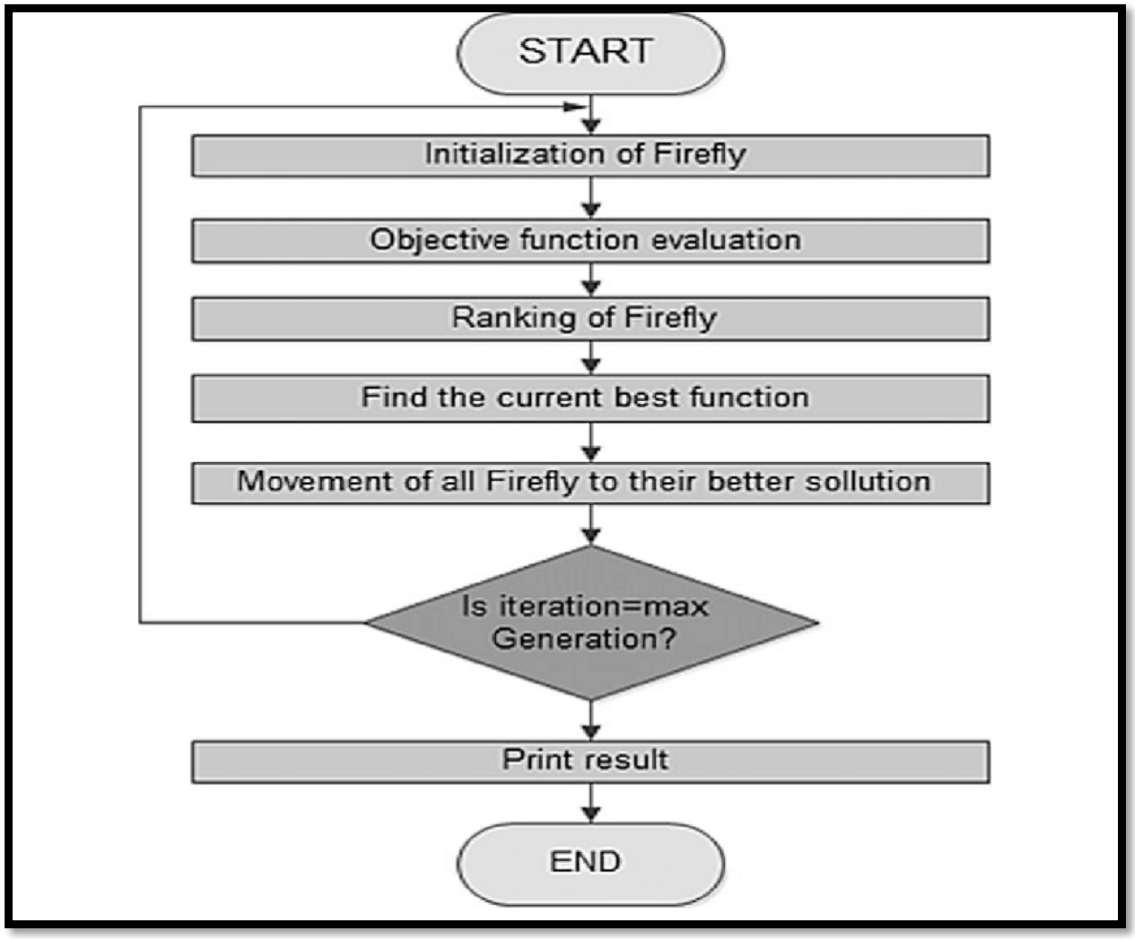


Figure 1.14. Organigramme des Lucioles (Firefly Algorithm).

1.7. Conclusion

L'optimisation c'est l'art d'exploiter cette caractérisation afin de déterminer les algorithmes qui les calculent mais aussi, de mettre en évidence les limites sur l'efficience et l'efficacité de ces algorithmes ». Un problème d'optimisation se définit comme la recherche du minimum ou du maximum d'une fonction donnée.

Ce premier chapitre a effectué un état de l'art sur l'optimisation ainsi les différentes qualités de service. Ce chapitre a également illustré la notion du chemin le plus court ainsi qu'un aperçu sur quelques algorithmes d'optimisation tels que les algorithmes Génétiques, Dijkstra, l'algorithme A*, floyd-Warshall et les algorithmes Métaheuristiques (gloutons, recuit simulé, colonies de fourmis, recherche tabou, Algorithme des Lucioles, Algorithme des Essaims de particules). Dans le prochain chapitre, nous allons présenter quelques généralités sur les systèmes de recommandations ainsi que quelques algorithmes utilisés dans la recommandation.

Chapitre 2

Travaux connexes

2.1. Introduction

L'optimisation est en effet un axe clé dans de nombreux domaines. Il contribue à améliorer l'efficacité, la productivité et les performances dans divers secteurs.

Dans ce chapitre, nous fournissons un aperçu complet des différents travaux connexes à notre projet. Nous avons sélectionné les travaux récents qui sont pertinents et se rapprochent de notre domaine de recherche, tels que l'optimisation. Recherche de plus court chemin, etc.

2.2. Quelques Travail connexe

2.2.1. Recherche de plus court chemin multimodal de point à point dépendant du temps

Le problème du chemin le plus court est un sujet fondamental de société étudié depuis de nombreuses années. Il s'agit d'un problème bien connu en théorie des graphes et en recherche opérationnelle. Une variante de ce problème se pose spécifiquement pour les réseaux de transports publics, et de nombreux algorithmes ont été développés pour résoudre ces problèmes complexes.

Dans cette thèse, l'étude du calcul d'itinéraires pour les transports en commun et son application dans les applications de mobilité pour smartphone sont au cœur des préoccupations. Voici les principales contributions :

- **Algorithme Multi objectif**

  Pour répondre aux différents besoins des utilisateurs, qui utilise du Précalcul afin de diminuer le temps de réponse. Cet algorithme est basé sur la recherche guidée par le but.

- **Problème des k Plus Courts Chemins**

  Appliqué aux réseaux de transports publics**,** ce problème énumère les plus courts chemins entre un point de départ et un point d'arrivée. Les résultats produits sont souvent diversifiés, offrant aux utilisateurs des options potentiellement intéressantes avec un temps de réponse rapide.

- **Algorithme Intermodal**

  Cet algorithme combine transports en commun et covoiturage. Il résout les inconvénients de chaque réseau : les horaires de passage des transports en

Commun peuvent ne pas convenir aux utilisateurs, tandis que trouver un covoitureur passant par les points de départ et d'arrivée est rare. Cette solution répond à une demande fréquente dans le monde moderne. (Finkelstein, 2020).

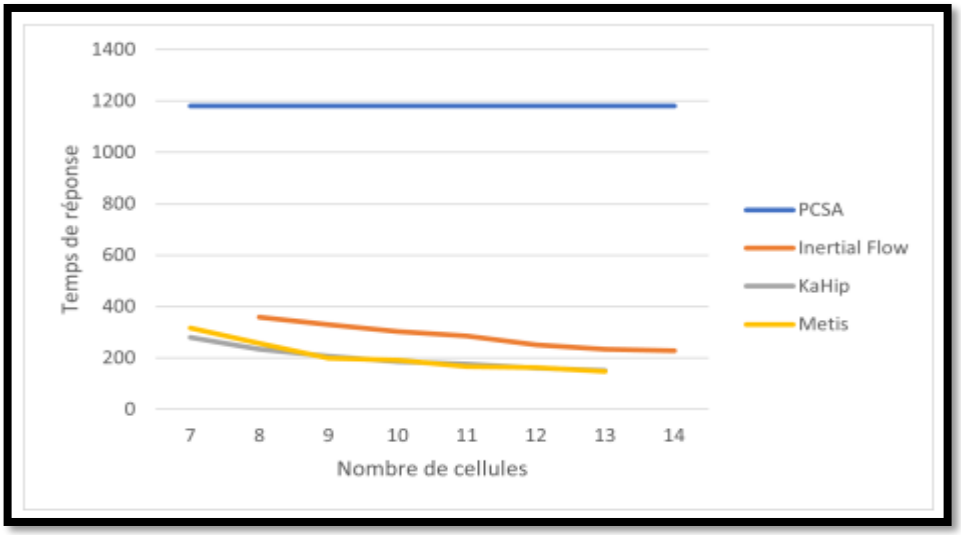


Figure 2.1. Temps de réponse en fonction du nombre de zones pour le réseau de train de la Suisse.

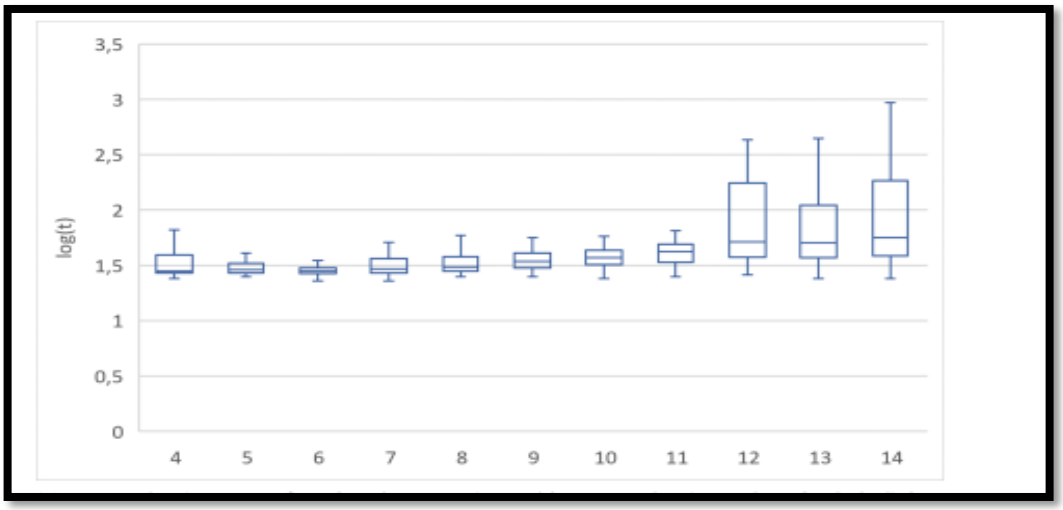


Figure 2.2. Temps de réponse en fonction du rang géographique pour le réseau de train de la Suisse.

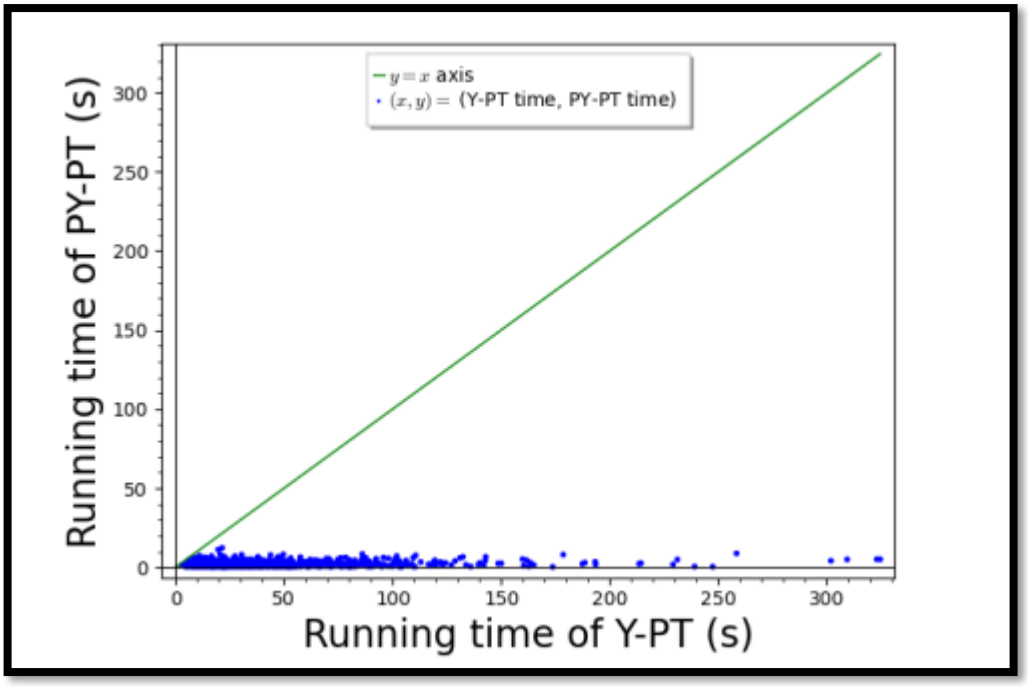


Figure 2.3. Temps de réponse du YPT en fonction du temps de réponse du PYPT pour le réseau de trains de la Suisse.

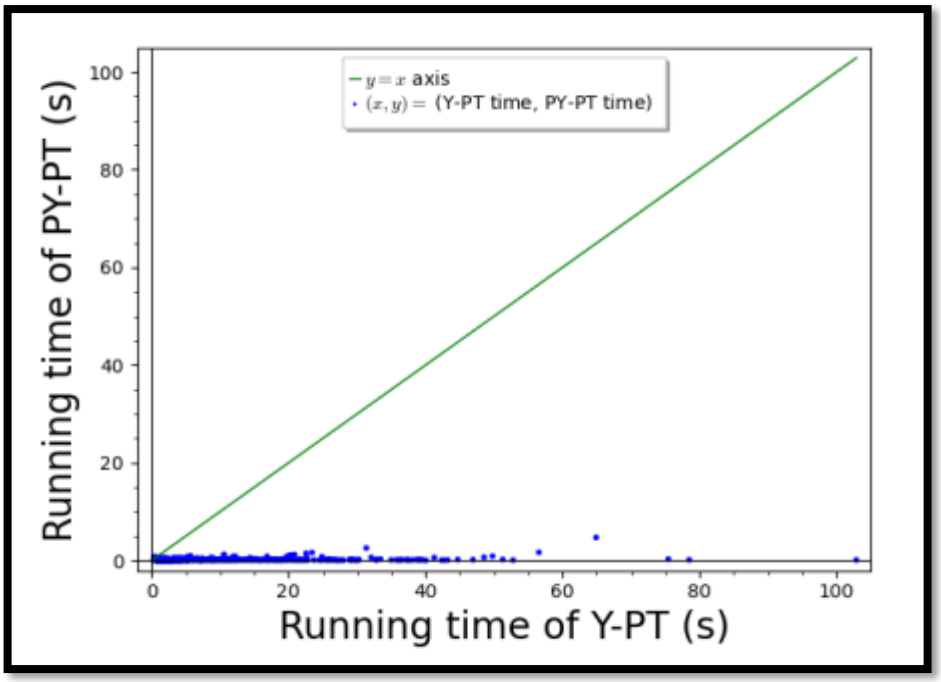


Figure 2.4. Temps de réponse du YPT en fonction du temps de réponse du PYPT pour la ville de Stockholm.

2.2.2. Extraction optimisée de règles d'association

L'exploration de données, connue aussi sous différentes autres appellations dont la fouille de données, vise à extraire des connaissances à partir de vastes ensembles de données. L'utilisation des techniques de data mining permet de convertir ces données en informations pertinentes pour la prise de décision. Parmi ces techniques, la recherche de règles d'association est largement utilisée pour découvrir des relations significatives, notamment lorsque les bases de données sont volumineuses.

Le but est de créer des règles d'association optimales en termes de pertinence. Pour ce faire, il a proposé une nouvelle métrique de qualité pour évaluer les règles d'association générées. Les résultats expérimentaux montrent l'apport de l'intégration de techniques d'exploration de données dans les algorithmes génétiques.

Dans cet article, nous présentons un cadre de travail visant à générer des règles d'association optimisées. Trois approches sont considérées :

- **Approche classique** : Elle applique trois algorithmes de data mining bien connus : **Apriori, Close** et **Charm.**
- **Approche hybride** : Elle intègre un algorithme génétique aux algorithmes classiques pour améliorer la qualité des règles d'association.
- **Approche séquentielle** : Elle commence par appliquer l'algorithme génétique sur la base de données, dans un souci de diversification, suivi par les algorithmes de data mining (Berkani, 2020).

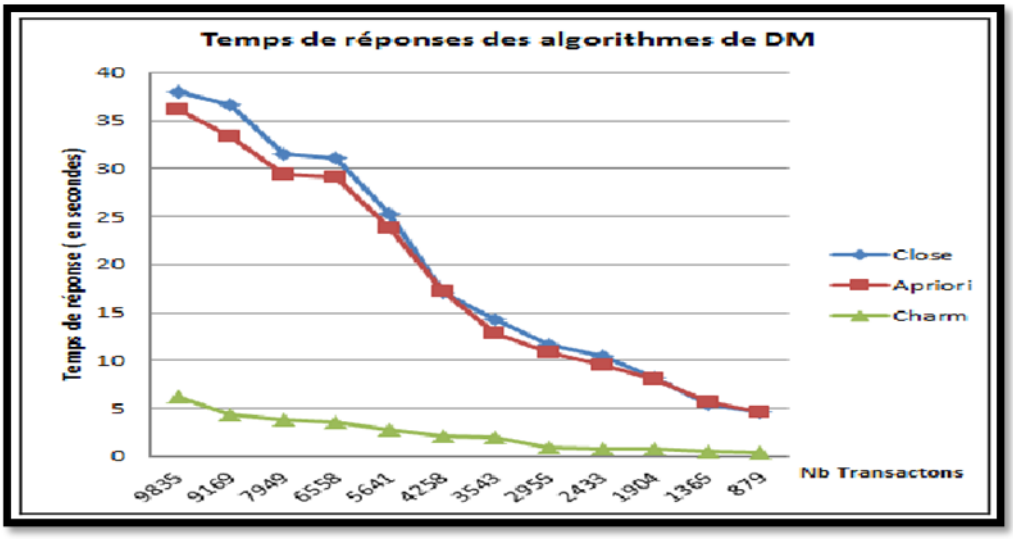


Figure 2.5. Evaluation du temps de réponse des algorithmes de DM.

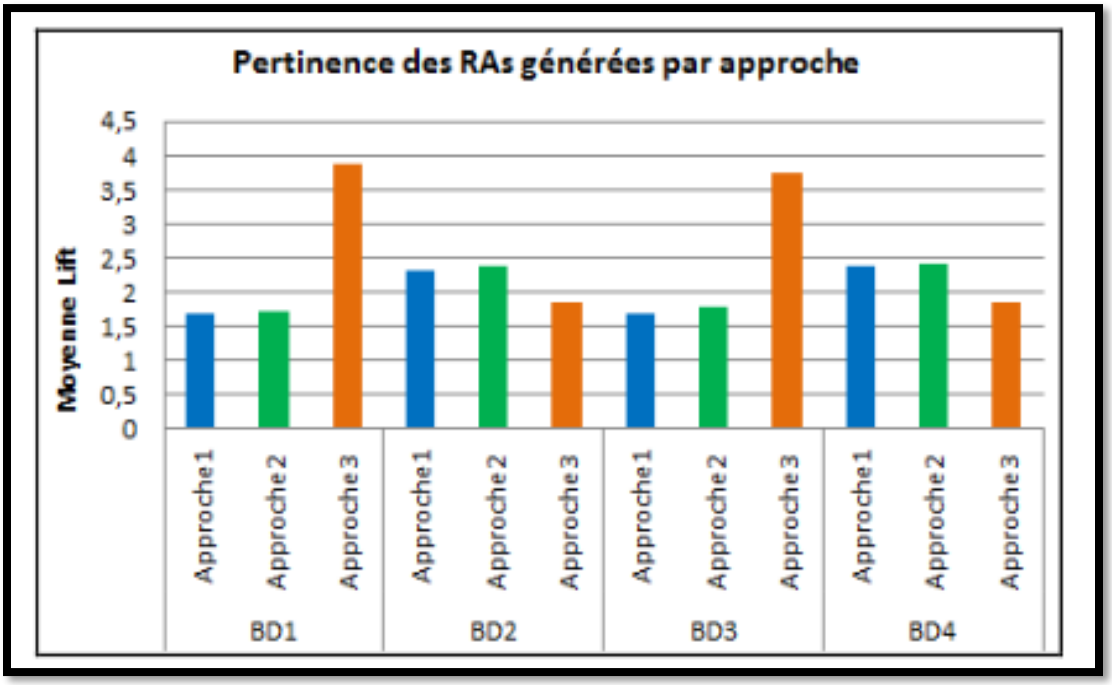


Figure 2.6. Comparaison de pertinence des RAs générée.

2.2.3. Amélioration de l'algorithme de chauve-souris par modification de règles d'évolution et introduction de mécanisme de croisement

L'algorithme bat est en effet un outil heuristique prometteur et a récemment été proposé pour résoudre des problèmes d'optimisation. Il s'inspire du comportement d'écholocation des chauves-souris. Dans cet article, cet algorithme est amélioré en modifiant certaines règles évolutives et en introduisant le mécanisme de croisement génétique. Les résultats obtenus montrent que ce nouvel algorithme surpasse l'algorithme standard des chauves-souris ainsi que d'autres algorithmes similaires publiés récemment.

Ils ont testé une approche de fonctions standards couramment utilisée dans la littérature sur l'optimisation numérique. Les résultats montrent clairement la supériorité de l'algorithme. Dans le futur, nous envisageons d'appliquer cet algorithme à des problèmes de data mining et à des applications temps réel, où le respect des contraintes temporelles est essentiel. (Bedboudi, 2018).

2.2.4. Chimp optimization algorithm

Le chimpanzé, également connu sous le nom de chimpanzé, est l'une des deux espèces africaines de grands singes. Ils ont une relation génétique remarquablement proche de l'homme et partagent un rapport cerveau-corps similaire, ressemblant à la fois aux humains et aux dauphins. Cette similitude est due au fait que les chimpanzés et les humains partagent une espèce d'ancêtre

commune, connue sous le nom d'Hominoïde, qui vivait il y a environ sept ou huit millions d'années.

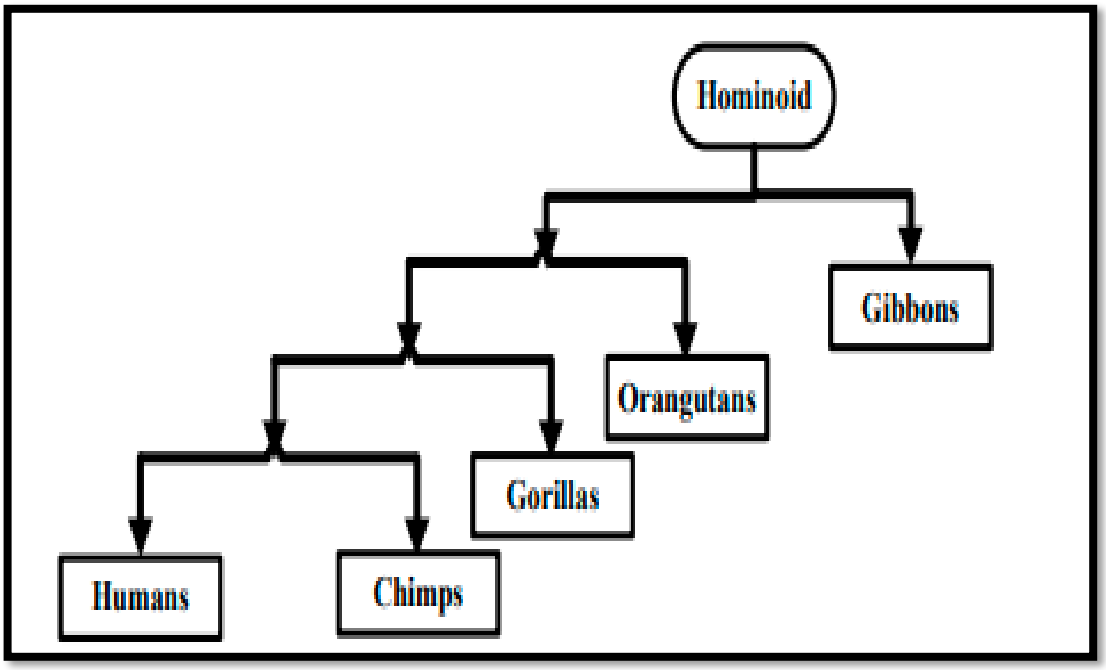


Figure 2.7. Phylogénie des hominoïdes de la superfamille.

Selon une étude, ces deux espèces ont une similitude de 98,8 % dans leur ADN.

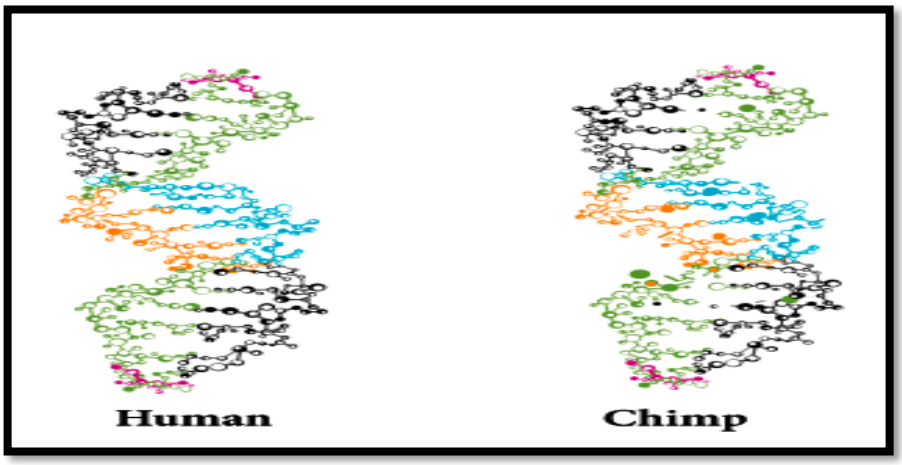


Figure 2.8. ADN humain et chimpanzé.

Cet article présente un algorithme innovant appelé Chimp Optimization Algorithm (ChOA), qui s'inspire de l'intelligence individuelle et de la motivation sexuelle des chimpanzés lors de chasses en groupe. ChOA vise à résoudre deux problèmes courants lors de la résolution de problèmes de grande dimension, tels que l'apprentissage d'algorithmes pour les réseaux de neurones : une vitesse de convergence lente et le fait d'être piégé dans des optima locaux. Pour simuler diverses

intelligences, l'algorithme utilise quatre types de chimpanzés : les attaquants, les barrières, les poursuivants et les conducteurs. Il met également en œuvre quatre étapes principales : chasser, conduire, bloquer et attaquer. L'algorithme est ensuite testé sur 30 fonctions de référence bien connues et comparé à quatre autres algorithmes métaheuristiques en termes de vitesse de convergence, de probabilité de rester coincé dans des minimums locaux et de précision des résultats. Les résultats démontrent que ChOA surpasse les autres algorithmes d'optimisation de référence. (Khishe, 2020).

2.2.5. Dynamic Programming for Shortest Path Problem in a Multimodal Transportation Network Comprising Intermediate Sinks

L'objectif principal de cette étude est de trouver une solution au problème du transport. Il vise à optimiser l'itinéraire entre le point de départ et la destination, qui comprend plusieurs arrêts tout au long du parcours. L'accent est mis sur la minimisation du coût du transport en considérant différents modes de déplacement, tels que le rail, la route et l'air. Cependant, il est important de noter que le mode aérien est limité dans sa capacité à transporter des marchandises via les escales intermédiaires, car le nombre d'aéroports disponibles est limité.

- **Revue de la littérature**

La revue de la littérature est divisée en deux sections : (1) SPP pour un mode de transport unique et (2) SPP pour plusieurs modes de transport.

Divers algorithmes et procédures ont été utilisés pour résoudre les problèmes de transport mixte dans différents scénarios :

- L'algorithme de Dijkstra.
- La programmation dynamique.
- La programmation floue.
- Les algorithmes génétiques.

Cependant, peu de recherches ont été menées sur la PPS mixte rail-route, et cette étude vise à combler cette lacune en développant un modèle mathématique pour la PPS qui utilise le transport modal mixte rail-route-air d'une source à une destination avec de multiples destinations intermédiaires. Ce modèle prend en compte différents modes de transport et optimise le coût par kilogramme pour la livraison des matériaux.

- **MODÈLE MATHÉMATIQUE**

L'étude développe des équations récursives avant et arrière pour un problème de chemin le plus court où il est nécessaire de passer par des puits intermédiaires tout en trouvant le chemin le plus court d'une source au puits final. Les notations utilisées pour développer des modèles mathématiques pour les calculs récursifs avant et arrière sont présentées.

La conclusion générale de cette étude sur le problème du plus court chemin avec plusieurs destinations intermédiaires entre une source initiale et une destination finale pour le transport de marchandises est la suivante :

Les résultats ont montré que :

- Le mode ferroviaire offre la distance la plus courte (1475 km) suivi du mode mixte route-rail-air (1513 km) et du mode routier (1527 km).
- Pour le coût du transport d'un kilogramme de marchandises, le train est le mode le moins cher (0,3667 PKR), suivi du mode mixte (0,4335 PKR) et du mode routier (0,4668 PKR).
- Le modèle développé optimise l'acheminement des biens et services (colis, produits, médicaments, etc.), en explorant différents modes dans un réseau de transport multimodal avec plusieurs destinations intermédiaires.
- Cependant, le choix du mode peut dépendre de priorités telles que le coût, le temps de transport ou les délais.



Figure 2.9. Comparaison sur la base de la distance la plus courte.



Figure 2.10. Comparaison sur la base du coût au kg.

En conclusion, le mode ferroviaire offre le chemin le plus court et le coût le plus bas, mais le choix final dépend de contraintes spécifiques. Cette étude aide à déterminer le mode optimal dans un cas avec des destinations multiples. Les travaux futurs pourraient prendre en compte des quantités variables de marchandises à transporter avec des délais différents. (Mobeen, 2023).

2.2.6. Optimization design of forebody shape of an underwater axisymmetric body for transition delay using a genetic algorithm

L'introduction offre un aperçu complet de l'importance de la transition laminaire-turbulente dans la couche limite des corps axisymétriques sous-marins. Plus précisément, il met en évidence le rôle de la position de transition dans l'influence de l'intensité du bruit et souligne le potentiel de l'optimisation de la forme pour retarder la transition et améliorer les performances, réduisant ainsi le bruit induit par l'écoulement pour améliorer les capacités de détection du sonar.

En outre, l'introduction souligne l'importance des algorithmes d'optimisation intelligents pour résoudre des problèmes d'ingénierie complexes. Il fournit des exemples issus de divers domaines, tels que la conception du profil aérodynamique, la mise en forme de la coque des navires et l'amélioration des performances des véhicules sous-marins, pour illustrer la nécessité de tels algorithmes. Il note également une lacune dans la littérature concernant l'application de l'optimisation intelligente spécifiquement pour retarder la transition dans les corps axisymétriques sous-marins.

En conséquence, l'article vise à combler cette lacune en proposant une approche combinant la méthode eN pour le calcul de la position de transition, la description paramétrique basée sur NURBS des formes de corps axisymétriques et un algorithme génétique pour l'optimisation

Intelligente. L'étude se concentre sur les dimensions des véhicules à l'échelle réelle et les vitesses d'écoulement fixes pour fournir des informations pratiques sur les défis de l'ingénierie océanique.

En conclusion, l'introduction prépare efficacement le terrain pour les sections suivantes en décrivant les méthodologies utilisées, les objectifs de l'étude et l'importance de la recherche proposée pour faire progresser le domaine de la conception et de l'optimisation des véhicules sous-marins.

➢ Méthodologies :

Cette section décrit l'approche permettant de prévoir le déplacement du point de transition de la couche limite sur un corps sous-marin cylindrique. Le processus comporte plusieurs étapes essentielles :

1. **Calcul initial de l'écoulement fluide :** Cette étape comprend le calcul de l'écoulement laminaire de la couche limite, en supposant qu'il n'y a pas d'angle d'attaque. Le solveur Ansys Fluent est utilisé pour résoudre numériquement les équations de Navier-Stokes et obtenir le flux de base.

2. **Analyse de stabilité** : L'analyse de stabilité est effectuée à l'aide d'un système de coordonnées qui s'aligne sur la forme du corps. Le flux de base, déterminé en coordonnées cylindriques, est projeté sur ce système. Les équations de perturbation sont linéarisées et les problèmes de valeurs propres sont résolus pour déterminer les taux de croissance dans la direction du cours d'eau.

3. **Prédiction de transition avec la méthode eN** : La méthode eN, une approche semi-empirique basée sur une analyse de stabilité linéaire, est utilisée pour prédire le point de transition naturel de la couche limite. Le taux de croissance est intégré dans la direction du courant pour calculer un facteur d'amplification, qui est ensuite comparé à une valeur seuil pour déterminer le point de transition.

4. **Analyse des spectres WPF dans la région laminaire instable** : à l'aide de l'analyse de stabilité et du point de transition prévu, les spectres des fluctuations de pression des parois (WPF) dans la région laminaire instable sont examinés.

➢ Modèle d'optimisation de forme

✓ Paramétrage des formes :

Afin d'optimiser efficacement la forme d'un corps axisymétrique sous l'eau, nous devons d'abord établir une manière de décrire la forme. Dans cette recherche, nous avons choisi d'utiliser la méthode NURBS pour le paramétrage, qui est brièvement expliquée ci-dessous.

La méthode NURBS fonctionne en représentant une courbe comme une combinaison de fonctions de base. Chaque point de la courbe est décrit à l'aide d'un paramètre spécifique, qui détermine sa position. Une courbe NURBS d'un ordre particulier peut être exprimée en utilisant le paramètre de la manière suivante.

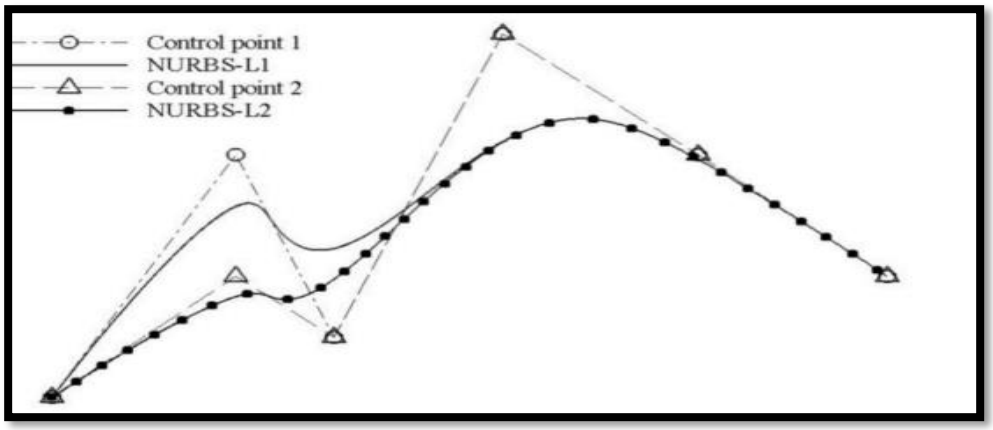


Figure 2.11. Deux courbes NURBS et les points de contrôle correspondants.

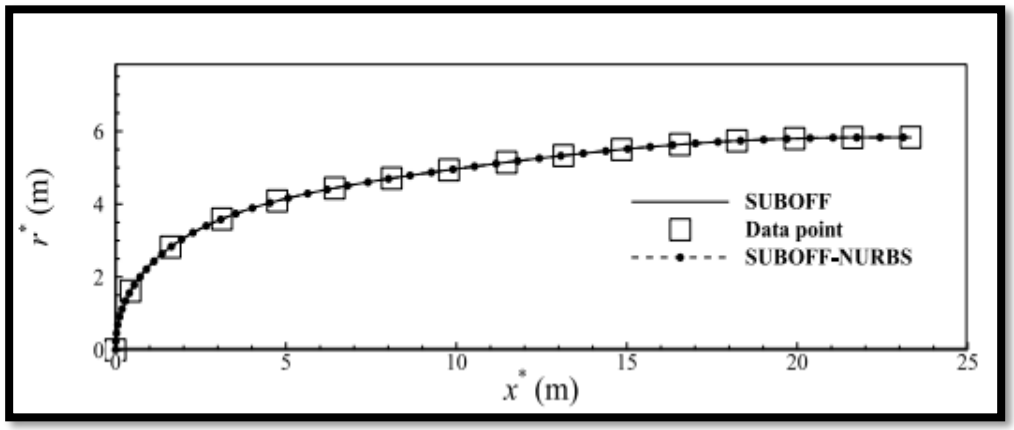


Figure 2.12. Forme originale de l'avant-corps de SUBOFF et forme de la méthode de paramétrage NURBS. Les symboles carrés sont les 16 points de données.

- NURBS curves, known as Non-Uniform Rational B-Spline curves, are employed for making local changes to shapes. This is exemplified through the comparison of two NURBS cubic curves that possess slightly varied control points.
- To depict the forebody shape of SUBOFF with precision, the NURBS parameterization technique utilizes 16 data points. The consistency between the NURBS shape and the original shape is confirmed by the growth rates, N factors, and Nmax curves.

- Based on the analysis conducted, it can be inferred that 16 data points adequately capture the shape for transition prediction within the scope of this study.

✓ **Variables de conception:**

Les auteurs décrivent une technique de représentation paramétrique de la forme de l'avant-corps d'un corps axisymétrique à l'aide de 18 points de contrôle NURBS, avec 31 variables définissant les coordonnées de ces points de contrôle. Les plages de variation de ces points de contrôle sont déterminées en fonction des valeurs des points de contrôle de sept formes classiques de l'avant du corps, obtenues à l'aide de la méthode NURBS.

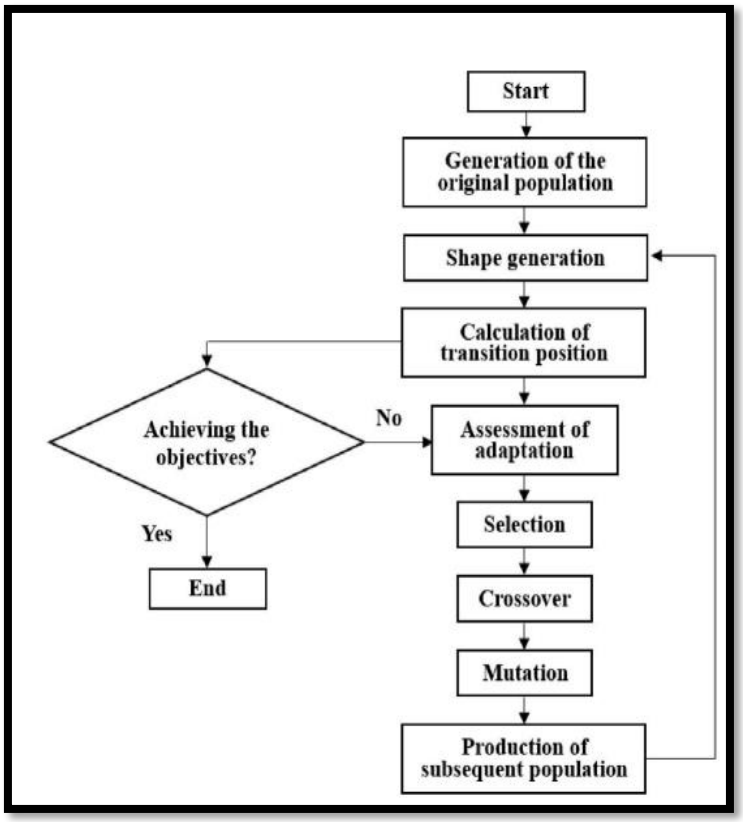


Figure 2.13. Organigramme de la méthode d'optimisation.

En résumé, l'étude suggère d'utiliser une technique d'optimisation basée sur l'IA, en particulier un algorithme génétique, pour créer la forme frontale d'un corps sous-marin avec

une symétrie de rotation. Cette approche vise à retarder la transition de la couche limite de 20,8 % et à minimiser les fluctuations de pression pariétale. Ce faisant, il améliore l'efficacité hydrodynamique et réduit le bruit provoqué par l'écoulement du fluide, améliorant ainsi les capacités sonar du système**.** (Lv, 2024).

2.2.7. Inner-outer layer co-optimization of sizing and energy management for renewable energy microgrid with storage

Cet article présente un nouveau cadre pour la Co-optimisation des couches internes-externes afin de résoudre le problème de Co-optimisation du dimensionnement des EMS interconnectés dans les micro-réseaux. En s'appuyant sur les résultats d'études antérieures et en incorporant l'optimisation à horizon mobile (RHO) et l'optimisation par essaims de particules (PSO), ce cadre réduit le risque de négliger les solutions optimales globales. L'analyse démontre la nature convexe du problème d'optimisation du dimensionnement et offre un aperçu de divers scénarios opérationnels.

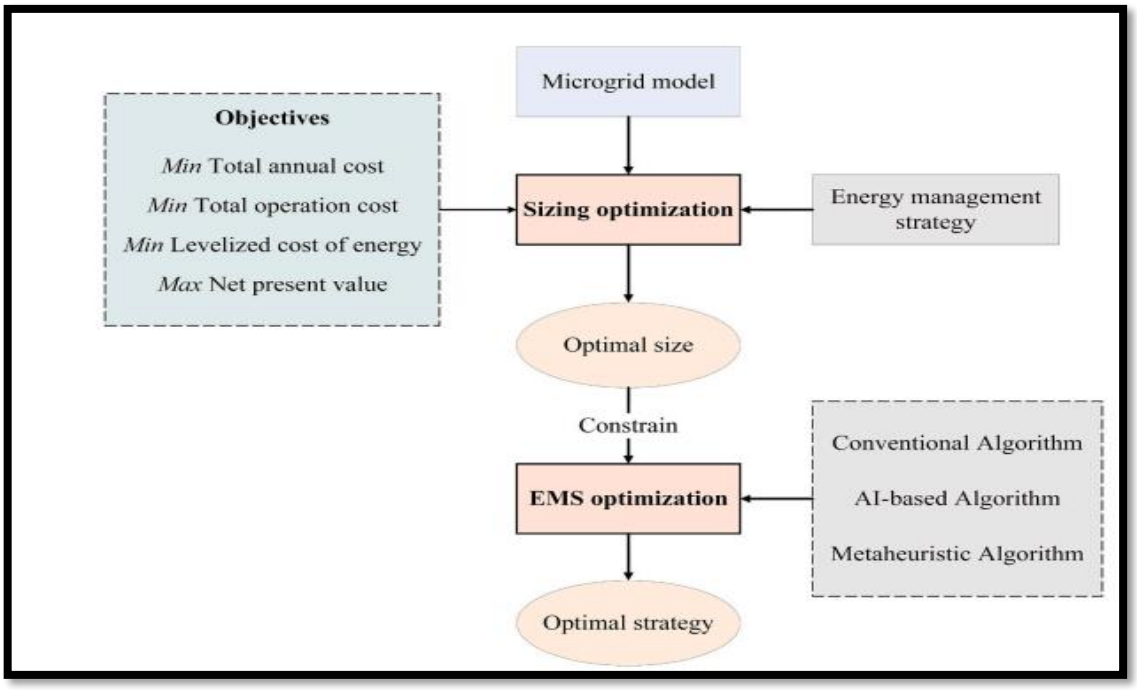


Figure 2.14. Schéma séquentiel conventionnel de conception de micro-réseau (révisé).

- **Modèle de micro-réseau et aspects économiques**

- ✓ **Modèle de micro-réseau AC :**

L'étude porte sur le dimensionnement des systèmes de production d'énergie renouvelable et de stockage par batterie dans un micro-réseau industriel à courant alternatif, en tenant compte des stratégies de gestion de l'énergie et de la fiabilité assurée par la connexion au réseau électrique.

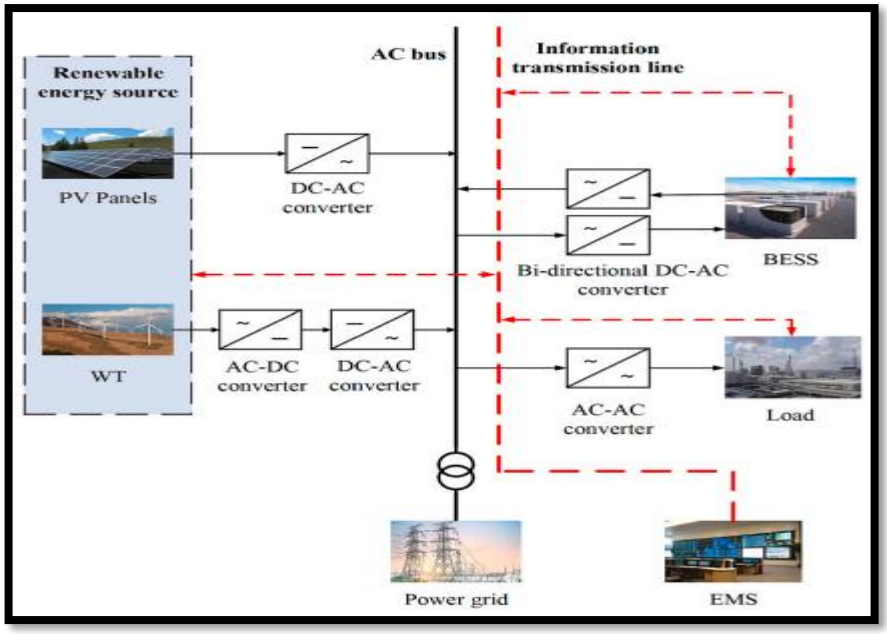


Figure 2.15. Micro-réseau AC proposé.

- ✓ **Modèle d'énergie renouvelable et profils de puissance :**

En utilisant le regroupement de k-moyennes, cette étude extrait des profils distincts pour la production d'énergie renouvelable à long terme à partir d'unités PV et WT. Il prend en compte le rayonnement solaire, la température ambiante et la vitesse du vent comme facteurs clés. Ces profils offrent des informations précieuses sur divers scénarios, aidant au dimensionnement et à l'analyse efficaces des systèmes d'énergie renouvelable.

- ✓ **Profils de puissance de charge :**

En employant une analyse de regroupement, cette étude identifie des modèles de charge uniques dans les parcs industriels, qui sont essentiels pour la conception et la gestion des micro-réseaux, mettant en évidence les niveaux variables et les fluctuations de la demande.

- ✓ **Modèle de système de stockage d'énergie par batterie :**

Les principaux aspects abordés comprennent l'intégration de paramètres clés tels que l'état de charge (SOC), la capacité de stockage, la puissance du système de stockage d'énergie de la batterie (BESS) et le taux de perte de puissance (a) pour caractériser la relation entre les états du BESS et le fonctionnement du micro-réseau. En outre, des transformations sont appliquées à la puissance du BESS pour faciliter l'optimisation et les calculs de coûts pour les processus de charge et de décharge.

- **Aspects économiques**

Les aspects économiques sont analysés pour formuler les fonctions objectives dans ce travail. L'analyse englobe non seulement les coûts et les revenus, mais intègre également des facteurs de pénalité. Dans ce travail, la monnaie utilisée est le yuan chinois (CNY).

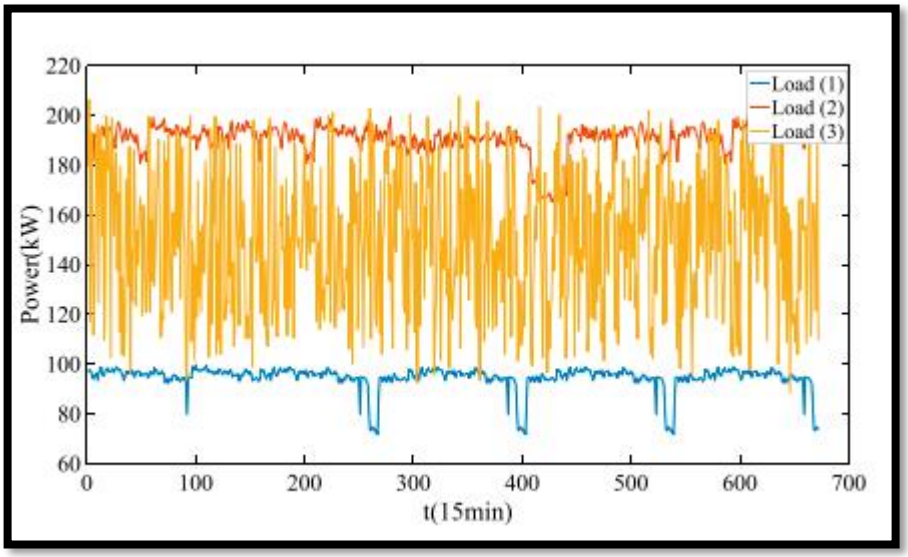


Figure 2.16. Charger des profils. Trois profils sont fournis pour représenter les petits, grands et charges fluctuantes, avec une résolution temporelle de 15 minutes sur une période de 7 jours.

Le cadre décrit fournit une évaluation complète de la viabilité économique des systèmes de micro-réseaux en intégrant les facteurs de coût, de revenu et de pénalité associés au délestage et à la réduction des SER.

Le cadre de Co optimisation des couches interne-externe relève le défi de l'optimisation des tailles et de la distribution d'énergie dans un système de gestion d'énergie intégré. En passant par un processus de pré-optimisation puis en optimisant le dimensionnement dans la couche externe et le système de gestion de l'énergie dans la couche interne, il trouve efficacement les meilleures stratégies tout en tenant compte des contraintes du système. Ce cadre fournit une solution réalisable pour optimiser les systèmes énergétiques de manière globale.

En conclusion, le nouveau cadre de la couche intérieure-extérieure pour l'optimisation des micro-réseaux incorpore avec succès les aspects de dimensionnement et d'exploitation, ce qui permet une optimisation globale supérieure aux méthodes conventionnelles. Son efficacité est démontrée par une réduction allant jusqu'à 23,1 % des coûts totaux annuels, ce qui souligne son importance dans l'avancement des stratégies de gestion des micro-réseaux (Lyu, 2024).

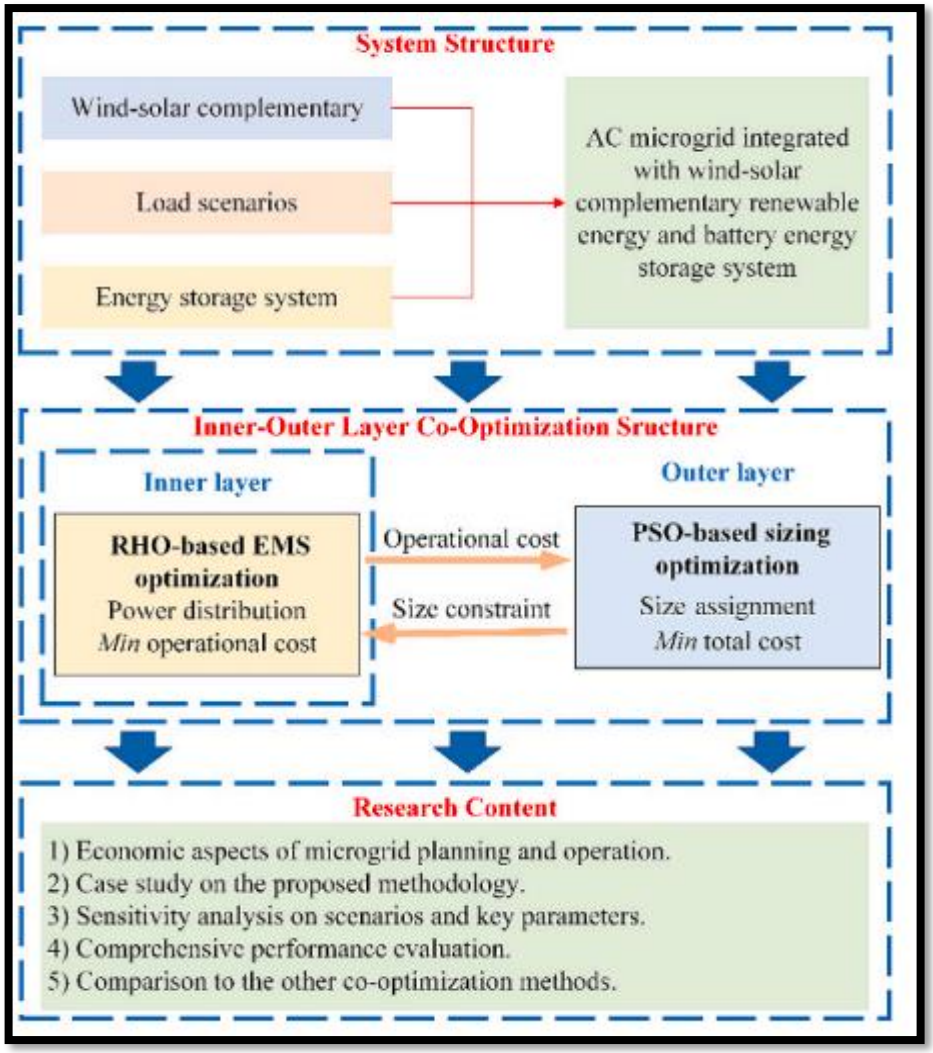


Figure 2.17. Résumé graphique.

2.2.8. Un algorithme hybride pour la résolution du problème de partitionnement matériel/logiciel

Pour développer un système embarqué avec un rapport coût / performance raisonnable, les concepteurs s'orientent vers la conception conjointe matériel/logiciel (M/L) ou codesign. Cette méthodologie comporte une suite d'étapes dont les résultats de la première est une spécification exécutable et la dernière est un prototype virtuel du système. Le partitionnement M/L, une étape cruciale du processus de Co design, a pour finalité de diviser le comportement d'un système en deux sous-ensembles de fonctions, l'un matériel et l'autre logiciel.

Le problème de partitionnement M/L étant NP-difficile, la littérature foisonne d'approches complémentaires pour remédier approximativement à ce problème. Dans ce papier, propose un nouvel algorithme hybride et fiable de partitionnement M/L basé sur le principe de l'algorithme Late Acceptance Hill Climbing (LAHC) et de l'algorithme génétique (AG). LAHC est connue pour

sa recherche locale efficace, conjuguée à la potentialité importante de recherche globale de l'AG. L'algorithme hybride, a été appliqué et a donné de meilleurs résultats, comparés à ceux de l'état de l'art.

L'algorithme proposé fournit de meilleurs résultats.la combinaison LAHC-AG est nouvelle et relance l'optimisation du problème de partitionnement M/L, pierre angulaire du développement embarqué. (Khetatba, 2018).

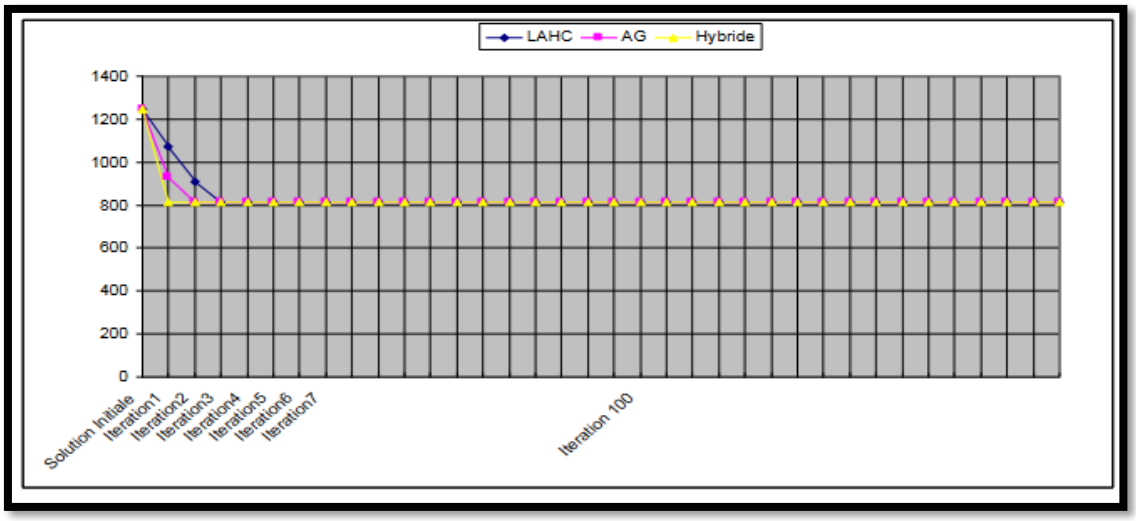


Figure 2.18. Convergence des 3 algorithmes vers la même configuration.

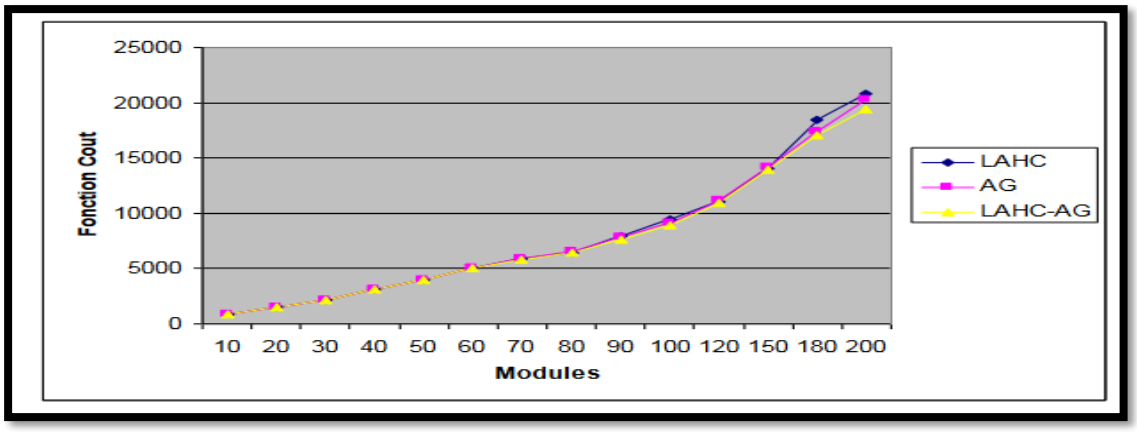


Figure 2.19. Comparaison des 3 algorithmes.

2.2.9. Deep reinforcement learning for intelligent risk optimization of buildings under hazard

Ce travail propose un cadre intelligent pour l'optimisation des risques en tirer parti de l'apprentissage par renforcement profond pour améliorer les performances construisez des structures en considérant de nombreuses offres utiles facteurs comprenant les conséquences

sociales, économiques et environnementales, et les coûts de rénovation. Le cadre est également comparé au Deep Q réseaux, optimisation des politiques de proximité, optimisation génétique et l'avantage de l'approche proposée en construction est démontré Structures compte tenu de scénarios de risques (Anwar, 2024).

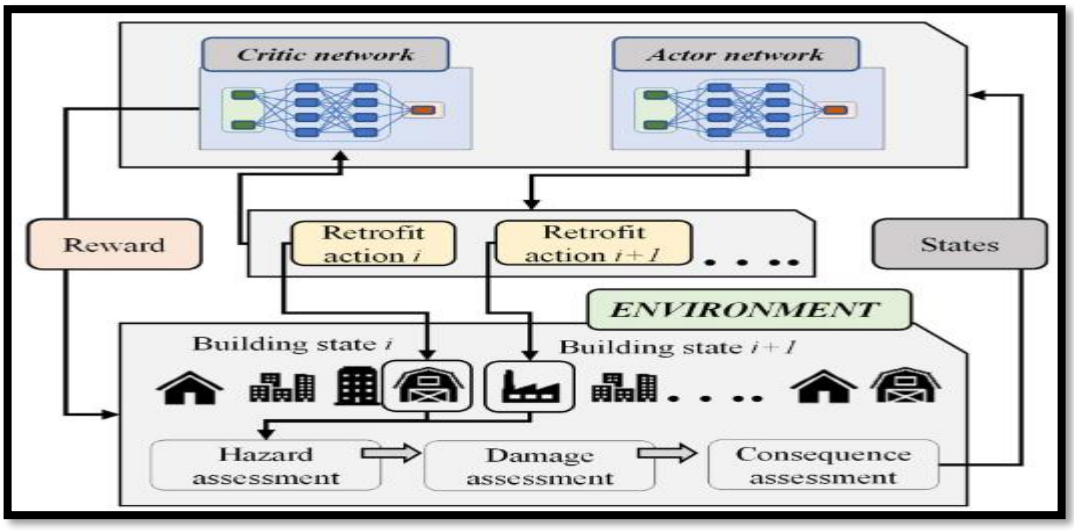


Figure 2.20. Illustration of the proposed DRL architecture for intelligent risk optimization of building structures.

2.3. Travaux connexes d'optimisation

Le tableau (2.1) montre quelques critères de comparaison de certains travaux connexes d'optimisation tels que : les algorithmes utilisés, l'ensemble de données utilisées et le type d'optimisation.

Référence	Année	Algorithmes	Base de données	Type de graphe
(Anwar, 2024)	2024	Algorithme Génétique Algorithme PPO Deep Q-Network (DQN) DRL-Enabled Retrofit Optimization	Structures de construction et Risques sismiques	Orienté
(Lv, 2024)	2024	Algorithme Génitique	Example numérique	Non mentionnée
(Lyu, 2024)	2024	Algorithme conventionnel Algorithme métaheuristique	Micro-réseaux	Non mentionnée

(Shadmani, 2024)	2024	Algorithme Génitique	Non mentionnée	Non Orienté
(Wang, 2024)	2024	Algorithm DE Algorithms optimization	Système d'eau central pour centre de données à Hangzhou, Chine	Orienté
(Jin, 2024)	2024	Algorithme Génitique	Non mentionnée	Orienté
(Mobeen, 2023)	2023	Algorithme Génétique Algorithme Dijkstra Programmation dynamique Programmation floue	Planification de trajectoire	Orienté
(Finkelstein, 2020)	2020	Algorithme dey (Y-PT) (PY-PT) Algorithme CSA Algorithme PNC Algorithme GDCSA Algorithme PCSA Algorithme Dijkstra Algorithme MCSA	Zones pour les réseaux de Paris, Berlin et l'Allemagne.	Orienté
(Berkani, 2020)	2020	Algorithme génétique Algorithme de data mining	Non mentionnée	Non mentionnée
(Khishe, 2020)	2020	Algorithme ChOA	Non mentionnée	Non Orienté
(Bedboudi, 2018)	2018	Algorithme de chauve-souris, Algorithme (HBA), Algorithme (MBA), Algorithme (NABA), (BA-SAM)	Nombre de Fonctions benchmark de référence	Non mentionnée
(Khetatba, 2018)	2018	Algorithme LAHC algorithme HC Algorithme Génétique	Non mentionnée	Orienté

Tableau 2.1. Travaux connexes.

2.4. Conclusion

Après avoir examiné les études connexes exposées dans ce chapitre, il est évident que l'optimisation joue un rôle crucial dans divers domaines, allant de la recherche en transport à la conception de structures en passant par la gestion de l'énergie. Chaque étude apporte des contributions significatives à son domaine respectif, en utilisant différentes approches comme les algorithmes génétiques, la programmation dynamique, l'apprentissage par renforcement profond, et bien d'autres. Les principales conclusions qui ressortent de cette revue comprennent :

- Diversité des approches : Les chercheurs utilisent une variété d'algorithmes et de méthodes, adaptés aux besoins spécifiques de leurs problèmes, ce qui démontre la souplesse de l'optimisation dans différents contextes.

- Amélioration des performances : De nombreuses études démontrent que les nouvelles approches proposées surpassent souvent les méthodes existantes en termes d'efficacité, de précision ou de capacité à résoudre des problèmes complexes.

- Applications pratiques : Les recherches présentées ne se limitent pas à des exercices théoriques, mais sont également axées sur des applications pratiques telles que l'optimisation des réseaux de transport, la conception de structures, la gestion de l'énergie, etc.

De manière nouvelle et originale, il est important de souligner que ces travaux mettent en évidence l'importance croissante de l'optimisation dans la résolution de problèmes concrets. Ils mettent également en lumière le potentiel continu de cette discipline pour améliorer l'efficacité, la productivité et les performances dans différents domaines. Pour résumer, ces études mettent en évidence l'importance de l'optimisation et son effet bénéfique dans divers domaines. Cette information est synthétisée dans un tableau comparatif qui prend en compte plusieurs critères tels que l'année, les algorithmes, les techniques et la base de données utilisée, ainsi que le type de graphe.

Chapitre 3

Approche proposée

3.1. Introduction

Dans ce chapitre, nous présentons les études qui ont attiré notre intérêt dans le cadre de notre projet.

Nous avons sélectionné les plus récentes et les plus pertinentes pour notre recherche, en particulier dans le domaine de l'optimisation. Bien que l'optimisation soit une discipline ancienne, elle a connu une évolution remarquable au fil du temps.

Nous avons souligné les points forts et les faiblesses des méthodes actuelles consacrées à la résolution de l'optimisation du chemin le plus court. Nous avons également repéré certains besoins et obstacles à surmonter.

3.2. Approche proposée

La méta-heuristique est une méthode bien connue dans le domaine de l'intelligence artificielle et a été très active ces dernières années. Il est tout à fait naturel d'utiliser ces techniques pour résoudre de nouveaux problèmes d'optimisation complexes. Avec la croissance massive de l'information, l'optimisation du Big Data et devenue une priorité de recherche. Les réseaux complexes, tels que les réseaux de transport, de communication, Internet et de distribution, ont également attiré l'attention de divers domaines de la science et de l'ingénierie. En combinant la méta-heuristique avec d'autres techniques d'optimisation, appelées méta-heuristiques hybrides, il est possible d'obtenir une plus grande flexibilité et un comportement plus efficace pour résoudre des problèmes de grande envergure et du monde réel.

3.2.1. Architecture globale de l'approche proposée

La figure (3.1) représente une architecture globale des bases de données et comment les utiliser.

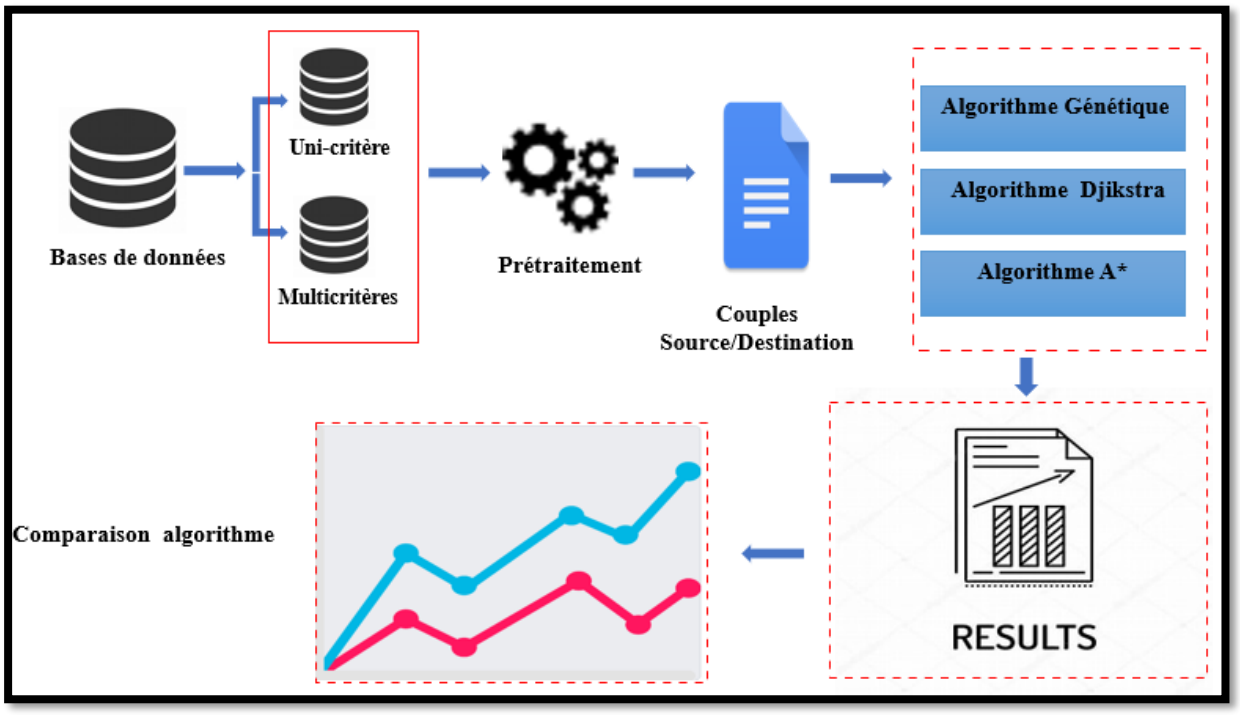


Figure 3.1. Architecture globale de l'approche proposée.

3.2.2. Comparaison Algorithme

Une comparaison d'algorithmes d'optimisation sur plusieurs bases de données unies-critère et multicritère consiste à évaluer et à comparer différentes méthodes d'optimisation pour résoudre des problèmes à une seule variable (uni-critère) ou à plusieurs variables (multicritères).

Les bases de données unies-critère ont pour objectif de réduire ou maximiser une seule métrique ou critère, tandis que les bases de données multicritères peuvent avoir plusieurs objectifs contradictoires à optimiser en même temps.

Dans le cas d'un problème de recherche du plus court chemin pour minimiser le coût, il s'agit généralement d'un problème d'optimisation classique dans le domaine de la recherche opérationnelle. Les algorithmes d'optimisation, tels que l'algorithme de Dijkstra ou l'algorithme A*, sont souvent utilisés pour trouver efficacement le chemin le plus court entre deux points dans un graphe pondéré.

Pour cette étude comparative, vous pourriez :

Sélectionner plusieurs algorithmes d'optimisation couramment utilisés pour résoudre des problèmes de recherche du chemin, tels que Dijkstra, A*, l'algorithme de Bellman-Ford, etc.

Collecter des jeux de données représentatives qui couvrent différents scénarios et contraintes, notamment des données unis-critères (par exemple, le coût le plus bas) et multicritères (par exemple, minimiser à la fois le coût et la distance).

Définir des métriques d'évaluation appropriées pour mesurer la performance des algorithmes, telles que le temps d'exécution, le nombre d'itérations, la qualité de la solution obtenue, etc.

Mettre en œuvre les algorithmes sélectionnés et les exécuter sur les jeux de données pour recueillir des données sur leur performance.

Analyser les résultats obtenus pour évaluer les forces et les faiblesses de chaque algorithme dans différents contextes et identifier celui qui convient le mieux à chaque type de problème.

Dans votre approche se concentre sur l'utilisation d'algorithmes d'optimisation afin de déterminer la meilleure approche dans différentes bases de données. Le but principal est de donner des renseignements précieux aux décideurs et aux professionnels pour qu'ils puissent sélectionner l'algorithme d'optimisation le plus approprié à leur problème particulier, qu'il s'agisse d'un algorithme à critère unique ou à pluri critère.

Pour expliquer en détail l'approche de votre travail, vous pourriez commencer par décrire les différentes bases de données que vous utilisez et les caractéristiques spécifiques de chaque

Problème. Ensuite, vous pourriez présenter les différents algorithmes d'optimisation que vous avez évalués et expliquer comment vous les avez appliqués à vos données. Il serait également utile d'inclure des informations sur les critères que vous avez utilisés pour évaluer la performance des algorithmes, ainsi que sur les résultats obtenus. Nous détaillons les étapes de notre projet.

3.2.3. Environnement et outils de développement

3.2.3.1 Python

Python est un langage de programmation orienté objet interprété de haut niveau avec une sémantique dynamique. Ses structures de données de haut niveau intégrées, combinées au typage et à la liaison dynamique, le rendent très attractif pour le développement rapide d'applications, ainsi que pour une utilisation comme langage de script ou de collage pour connecter des composants existants entre eux. La syntaxe simple et facile à apprendre de Python met l'accent sur

la lisibilité et réduit donc le coût de maintenance du programme. Python prend en charge les modules et les packages, ce qui encourage la modularité des programmes et la réutilisation du code. L'interpréteur Python et la vaste bibliothèque standard sont disponibles gratuitement sous forme source ou binaire pour toutes les principales plates-formes et peuvent être distribués gratuitement.[1]

3.2.3.2. Kaggle

Kaggle est une plateforme de concours de science des données et une communauté en ligne pour les scientifiques des données et les praticiens de l'apprentissage automatique. Initialement fondée en avril 2010 par Anthony Goldbloom et Ben Hamner, elle fournit une plate-forme permettant aux utilisateurs d'accéder à des ensembles de données, de créer et d'explorer des modèles dans un environnement Web, de collaborer avec d'autres et de participer à des concours pour résoudre des défis réels en matière de science des données. Kaggle appartient à Google depuis mars 2017 et compte plus de 15 millions d'utilisateurs dans 194 pays en octobre 2023. Il propose une variété de ressources, y compris des modèles pré-entraînés introduits en février 2023, pour soutenir l'apprentissage, la collaboration et l'innovation dans le domaine de la science des données et de l'apprentissage automatique.[2]

3.2.3.3. Google Colab

Colab est un service Jupyter Notebook hébergé qui ne nécessite aucune configuration et offre un accès gratuit aux ressources informatiques, y compris les GPU et les TPU. Colab est particulièrement bien adapté à l'apprentissage automatique, à la science des données et à l'éducation.[3]

3.2.3.4. Les bibliothèques utilisées

Ces bibliothèques sont utilisées dans des domaines différents mais souvent interconnectés : la manipulation de données, l'analyse de réseaux et la visualisation graphique.

A. Xlwt

Cette bibliothèque est utilisée pour créer et manipuler des fichiers Excel au format .xls.[4] Elle est particulièrement utile lorsque vous devez automatiser la création de feuilles de calcul Excel à

[1] https://www.python.org
[2] https://www.kaggle.com
[3] https://colab.google
[4] https://pypi.org/project/xlwt/

partir de données générées par votre programme Python. Par exemple, vous pouvez l'utiliser pour écrire des données dans des cellules spécifiques, formater les cellules ou créer des graphiques à partir des données.

B. NetworkX

NetworkX est une bibliothèque python très utile pour modéliser vos données sous forme de graphes.[1] Elle fournit des outils pour créer des graphes, ajouter des nœuds et des arêtes, calculer des mesures de centralité, trouver des chemins les plus courts,

Les réseaux peuvent être utilisés pour représenter une variété de données, notamment les réseaux sociaux, les réseaux de transport, les réseaux biologiques, etc.

C. Matplotlib. Pyplot

Matplotlib est une bibliothèque Python open source permettant de créer des visualisations de données.[2] Matplotlib.pyplot est une sous-bibliothèque de Matplotlib qui fournit une interface de style MATLAB pour tracer des graphiques en 2D. Vous pouvez l'utiliser pour créer des graphiques tels que des diagrammes en barres, des courbes, des nuages de points, des histogrammes, etc., C'est souvent utilisé en conjonction avec d'autres bibliothèques comme Pandas pour visualiser des données extraites de feuilles de calcul Excel ou générées à partir d'autres sources.

D. Xlrd

Cette bibliothèque permettant de lire des données et de formater des informations à partir de fichiers Excel au format .xls.[3]

En résumé, xlrd vous permet d'interagir avec des fichiers Excel dans votre programme Python, ce qui peut être utile pour diverses tâches telles que l'analyse de données, le traitement de feuilles de calcul ou la génération de rapports à partir de données Excel.

E. Random

L'importation de la bibliothèque random en Python permet de générer des nombres aléatoires.[4] Cette bibliothèque offre différentes fonctionnalités pour générer des nombres entiers, des nombres à virgule flottante et des séquences aléatoires. Elle peut être utilisée pour des simulations, mélanger des éléments dans une liste ou choisir un élément aléatoire dans une séquence. La bibliothèque

[1] https://datascientest.com/networkx-tout-savoir
[2] https://datascientest.com/matplotlib-tout-savoir
[3] https://xlrd.readthedocs.io/en/latest/
[4] https://docs.python.org/fr/3/library/random.html

random est polyvalente et peut être utilisée dans divers contextes, tels que la génération de données de test, la simulation, et le développement de jeux et d'applications interactives.

Lorsque vous importez la bibliothèque random en Python, vous pouvez générer des nombres aléatoires, mélanger des éléments dans une liste, choisir un élément aléatoire dans une séquence, etc., Cette bibliothèque est polyvalente et peut être utilisée dans de nombreux contextes, tels que la génération de données de test, la simulation, ou le développement de jeux et d'applications interactives.

F. Time

La bibliothèque "time" de Python offre des fonctionnalités cruciales pour la gestion du temps, incluant la mesure du temps écoulé, la suspension de l'exécution de programmes, et la manipulation de dates et heures. En utilisant des fonctions comme time (), sleep (), ctime (), gmtime (), localtime (), et strftime (), les développeurs peuvent efficacement gérer les aspects temporels de leurs applications, que ce soit pour des scripts système, des applications Web, ou des jeux vidéo. Avec ces outils, la gestion du temps devient une tâche simplifiée et précise, facilitant ainsi le développement d'applications robustes et fonctionnelles.

G. Heapq

Est un module en Python qui fournit une implémentation de l'algorithme de file de priorité, également connu sous le nom de tas binaire (ou tas de priorité). Ce module permet de maintenir une liste triée efficacement, où l'élément le plus petit est toujours à la racine du tas.

3.3. Description et pré-traitement des bases de données

Dans notre projet, nous avons choisis quatre bases de données pour évaluer notre algorithme et faire plusieurs tests et comparaisons. Les quatre bases de données sont caractérisées par des critères qui sont différents en termes de taille, de types et de catégories, afin d'avoir une diversité et faire des tests sur des échantillons différents. Nous présentons dans les prochaines

Ligne les quatre jeux de données et leurs propriétés et caractéristiques, ainsi que leurs sources.

3.3.1. Google Web Graph

Après avoir lu l'étude de (Scarselli, F., Yong, S. L., Gori, M, 2005) , nous avons choisi d'utiliser Kaggle comme plateforme de recherche. Nous avons trouvé une base de données en ligne pertinente pour notre sujet d'études.

Le google Web Graph est une base de données massive qui représente la structure du World Wide Web. Il s'agit essentiellement d'un graphe où les nœuds représentent des pages Web et les arêtes représentent les liens entre ces pages.

La base de données sert à alimenter les algorithmes de recherche de Google et à analyser

La structure et le comportement du Web. Pour construire le google web Graph, des milliards de pages Web sont parcourues et indexées en suivant les liens entre les pages pour créer un modèle complet de la structure du Web. La base de données est régulièrement misée à jour pour refléter les changements et les évolutions du Web. Les professionnels utilisent le Google Web Graph pour différentes applications, telles que l'analyse de la connectivité du Web, l'étude des tendances de recherche, l'amélioration des algorithmes de recherche et bien d'autres encore. C'est une ressource précieuse pour comprendre et explorer la nature complexe et dynamique du World Wide Web. Les pages Web sont représentées par des nœuds et les hyperliens entre les pages par des bords dirigés. Les données ont été publiées en 2002 par Google dans le cadre du concours de programmation Google.

- **Contenu**

Cette base de données présente des caractéristiques uni-critères. Elle contient un total de 875 713 points de données et 5 105 039 connexions. Le plus grand groupe de données faiblement connecté (WCC) est composé de 855 802 points de données et 5 066 842 connexions, ce qui représente une forte densité de 0,977 pour les points de données et 0,993 pour les connexions. Le plus grand groupe de données fortement connecté (SCC) est composé de 434 818 points de données et 3 419 124 connexions, avec un coefficient de regroupement moyen de 0,5143. La base de données contient également un nombre significatif de triangles, totalisant 13 391 903, avec une fraction de triangles fermés de 0,01911. Le diamètre, qui représente la distance maximale entre deux points de données, est de 21, tandis que le diamètre effectif au 90e centile est de 8,1.

3.3.1.1. Traitement des données

La configuration de la base de données nécessite la conversion du fichier web-Google.txt dans un format utilisable tel que bdd1.xls. Cette étape permet d'organiser les données de manière structurée et accessible pour un traitement ultérieur. Une fois terminée, la base de données est prête à être utilisée pour des analyses, des requêtes et d'autres opérations utiles. La préparation est une étape essentielle car elle garantit que les données sont correctement formatées et prêtes à être utilisées efficacement dans différents contextes.

Ce code Python est utilisé pour lire un fichier texte, stocker les données dans une feuille de calcul Excel et créer un graphe orienté à partir de ces données.

Le code est expliqué de manière générale :

```
import xlwt
import networkx as nx
import matplotlib.pyplot as plt
workbook = xlwt.Workbook()
sheet = workbook.add_sheet('1')
filin = open("bdd.txt", "r")
ss=(filin.readlines())
print("NB des arêtes = ",len(ss))
cpt=0
G = nx.DiGraph()
G.graph['node']={'shape':'point'}
for j in range(0,102):#0 102
   print(j)
   k1=j*50000
   k2=(j+1)*50000
   cpt=cpt+2
   ligne=0
   for i in range(k1,k2):
       #print(i)
       t=ss[int(i)].split("\t")
       #print(t)
       sheet.write(ligne, 0+cpt, t[0])
       sheet.write(ligne, 1+cpt, t[1])
       ligne=ligne+1
for j in range(102,103):#102,103
   k1=j*50000
   k2=(j+1)*50000
   cpt=cpt+2
   ligne=0
   for i in range(k1,5105039):#k1
       #print(i)
       t=ss[int(i)].split("\t")
       sheet.write(ligne, 0+cpt, t[0])
       sheet.write(ligne, 1+cpt, t[1])
       ligne=ligne+1

       #5105039

workbook.save('bdd1.xls')
```

Figure 3.2. Capteur de Code Python pour convertir la bdd de txt vers excel.

Ce code importe des bibliothèques pour créer des fichiers Excel, manipuler des graphes et visualiser les graphes. Il lit le contenu d'un fichier texte et crée un graphe orienté. Ensuite, il écrit les données dans une feuille Excel et enregistre le fichier.

Nous avons un graphe orienté G qui n'est pas complet. Chaque nœud du graphe représente une page Web, et le nombre de nœuds est donné en entrée. Les arêtes du graphe représentent les liens entre les pages.

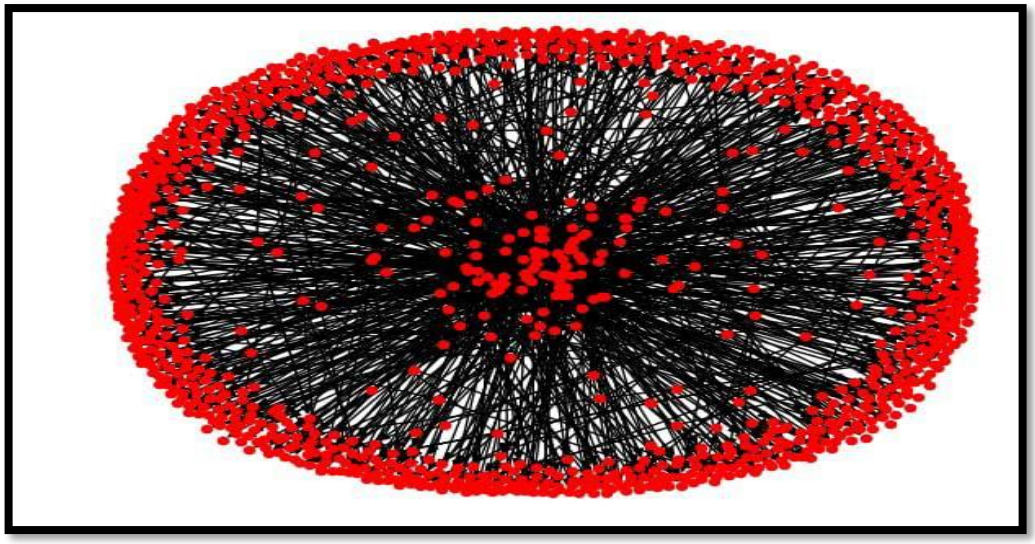


Figure 3.3. Graphe de base de données 1.

La figure (3.4) représente la base de données finale après l'avoir convertie du texte en Excel.

1	id	Source	Destination
2	1	0	11342
3	2	0	824020
4	3	0	867923
5	4	0	487427
6	5	11342	0
7	6	11342	27469
8	7	11342	38716
9	.	.	.
10	.	.	.
11	.	.	.
12	.	.	.
13	.	.	.
14	.	.	.
15	.	.	.
16	.	.	.
17	.	.	.
18	.	.	.
19	5 105 040	916425	837379

Figure 3.4. Capture de la BDD1 Web-Google.

Après une analyse minutieuse de la base de données, nous avons identifié la présence de certains nœuds qui ne figuraient pas dans notre ensemble de données initial. Pour remédier à cette situation, nous avons entrepris une extraction ciblée de ces nœuds et les avons compilés dans un fichier Excel distinct.

	A	B	C
1	id	Noueds vides	Existence
2	1	39	n'existe pas
3	2	46	n'existe pas
4	3	66	n'existe pas
5	4	98	n'existe pas
6	5	100	n'existe pas
7	6	143	n'existe pas
8	7	250	n'existe pas
9	8	264	n'existe pas
10	9	274	n'existe pas
11	...	...	...
12	...	...	...
13	40715	916422	n'existe pas
14	40716	916428	n'existe pas

Figure 3.5. Capture de fichier Excel qui contient les Nœuds vides.

```
import xlwt
import networkx as nx
import matplotlib.pyplot as plt
import xlrd
workbook1 = xlwt.Workbook()
sheet1 = workbook1.add_sheet('1')

myBook = xlrd.open_workbook('bdd_Noueds_vides.xls')
index = myBook.sheet_by_index (0)

debut=0
fin=40715
#print(int(index.cell_value(rowx=40715, colx=0)))
ligne=916428
cpt=0
while (debut < fin):
  trouver=False
  while (trouver==False):
    if int(index.cell_value(rowx=fin, colx=0)) == ligne :
      ligne=ligne-1
      fin=fin-1
      #print(ligne)
    if int(index.cell_value(rowx=fin, colx=0)) != ligne :#else
        #print("on va remplacé",int(index.cell_value(rowx=debut, colx=0)),ligne)
        try:sheet1.write(cpt, 2, index.cell_value(rowx=debut, colx=0))
        except:print(debut)
        sheet1.write(cpt, 0, ligne)
        sheet1.write(cpt, 1, "remplacer par ")
        #print(index.cell_value(rowx=debut, colx=0),ligne)
        cpt=cpt+1
        ligne=ligne-1
        debut=debut+1
        trouver=True
workbook1.save('resultats.xls')
```

Figure 3.6. Capteur Code Python pour Remplacé les Noueds vides.

Ce code Python semble effectuer une opération de remplacement de valeurs dans un fichier Excel existant nommé "bdd_Noueds_vides.xls". Voici un résumé des principales étapes du code :

- Importation des bibliothèques xlwt, xlrd, networkx et matplotlib.

- Initialisation des objets :

✓ Création d'un nouveau classeur Excel avec xlwt.Workbook().

✓ Ajout d'une nouvelle feuille nommée '1' dans ce classeur.

✓ Ouverture des données d'un fichier Excel existant avec xlrd.open_workbook() et récupération de l'index de la première feuille.

- Boucle principale utilisant les variables debut et fin :

✓ À l'intérieur de cette boucle, une autre boucle est utilisée pour traiter les données du fichier Excel.

- Traitement des données :

✓ Comparaison des valeurs de certaines cellules avec la variable ligne.

✓ Si une condition est vérifiée (trouver == False), certaines cellules sont écrites dans la nouvelle feuille Excel.

✓ Remplacement des valeurs d'une colonne par une autre valeur (ligne - 1) et insertion dans la nouvelle feuille Excel.

✓ Écriture du numéro de ligne actuel et de l'indication "remplacer par" dans la feuille Excel.

- Le résultat final est enregistré dans un nouveau fichier Excel nommé 'resultats.xls'.

	A	B	C	D
1	id	Noueds a Remplacé		Noueds Remplacé
2	1	916427	→	39
3	2	916426	→	46
4	3	916425	→	66
5	4	916424	→	98
6	5	916423	→	100
7	6	916421	→	143
8	7	916420	→	250
9	8	916419	→	264
10	9	916418	→	274
11	...	...	→	...
12	...	...	→	...
13	...	...	→	...
14	39665	875715	→	875675
15	39666	875714	→	875678

Figure 3.7. Capture de fichier excel qui contient les Nœuds vides Remplacé.

En résumé, ce code être pour remplacer certaines valeurs d'un fichier Excel par d'autres valeurs, en créant un nouveau fichier Excel avec les valeurs remplacées.

Après avoir extrait les nœuds vides et tenté de les ajouter à la première base de données, nous avons rencontré un problème dû à la taille excessive de cette dernière.

Pour résoudre ce défi, nous avons opté pour une solution innovante : diviser la base de données en 25 parties distinctes.

Grâce à cette approche, chaque partie de données ainsi fractionnées peut être traitée de manière indépendante, réduisant considérablement la charge de travail.

Cependant, cette méthode implique un temps de traitement plus long, chaque jeu de données nécessitant environ 3 heures pour être entièrement traité.

Enfin, nous avons obtenu la base de données finale.

	A	B	C
1	id	Source	Destination
2	1	856260	331684
3	2	856260	363572
4	3	856260	645265
5	4	856260	681075
6	5	856260	861979
7	6	769	155282
8	7	769	181860
9	8	769	353454
10	9	769	575918
11	...	...	...
12	...	...	...
13	49999	272905	853212
14	50000	272905	868431

Figure 3.8. Capture de la BDD1 Web-Google Finale.

3.3.2. Melbourne Restaurant Food Delivery data

Melbourne Restaurant Food Delivery est une base de données, L'ensemble de données contient des informations sur la livraison de nourriture d'un restaurant à Melbourne, en Australie. Le restaurant possède trois succursales dans la zone CBD. Les trois succursales partagent le même menu, mais elles ont une direction différente et fonctionnent donc de manière distincte.

La base de données se compose de six fichiers .csv, Nous avons choisi parmi eux nodes.csv et edges.csv.

La base de données nodes.csv et edges.csv a été téléchargée.

	A	B	C	D	E	F
1		u	v	distance(m)	street type	speed(km/h)
2	0	711327755	711332946	58.0	1	15.0
3	2	711327755	55725841	15.0	2	20.0
4	3	711327755	711327760	165.0	1	15.0
5	4	711327756	703557073	4.0	0	10.0
6	5	711327756	711332946	8.0	1	15.0
7	7	711327756	711331420	44.0	0	10.0
8	8	711327759	711331451	8.0	2	20.0
9	10	711327759	703557072	5.0	1	15.0
10	11	711327759	711331434	32.0	0	10.0

Figure 3.9. Capture de la BDD2 edges.

	A	B	C
1	node	lat	lon
2	711327755	-37.807675	144.9558726
3	711327756	-37.8070943	144.9559785
4	711327759	-37.8073009	144.9578172
5	711327760	-37.8078848	144.9577193
6	777781264	-37.8052207	144.9521732
7	711327758	-37.8062498	144.9580023
8	164102232	-37.8203223	144.9675308
9	26476632	-37.8227347	144.9311156
10	26476637	-37.8262913	144.930619
11	26476642	-37.8262801	144.9307411
12	356125087	-37.8165761	144.9801473

Figure 3.10. Capture de la BDD2 Nodes.

- **Contenu**

Cette base de données nodes.csv contient des informations permettant d'identifier des points géographiques spécifiques, appelés "nœuds". Elle contient un total de 17 117 nœuds.

Contient 3 colonnes, qui sont :

1) Node : C'est l'identifiant unique de chaque nœud

2) Lat : C'est la latitude du nœud

3) Lon : C'est la longitude du nœud

En ce qui concerne l'ensemble de données edge.csv, il représente les arêtes du réseau routier, où chaque arête relie deux nœuds. Il contient 17 117 nœuds et 42 224 arêtes.

Cet ensemble de données contient les informations suivantes :

1) U : C'est l'identifiant du nœud de départ de l'arête (ou de la route).

2) V : C'est l'identifiant du nœud de fin de l'arête (ou de la route).

3) Distance(m) : Il s'agit de la distance entre les nœuds de départ (u) et de fin (v) en mètres. Cela représente la longueur de la route entre deux points.

4) Street type : Cette colonne représente probablement le type de rue ou de route.

5) Speed(km/h) : C'est la vitesse typique à laquelle on peut voyager sur chaque segment de route, exprimée en kilomètres par heure.

3.3.2.1. Traitement des données

La modification des noms de nœuds en nombres ordonnés non aléatoires basés sur un index et l'enregistrement des données modifiées dans des fichiers Excel nodes_updated1.xls et edges_updated1.xls

Sont des étapes importantes dans la préparation des données. Cela garantit que les données sont correctement formatées et prêtes à être utilisées efficacement dans différents contextes.

Ce code Python effectue plusieurs opérations sur des données de graphes stockées dans des fichiers CSV. Tout d'abord, il charge deux ensembles de données à partir de fichiers CSV, qui représentent respectivement les nœuds et les arêtes du graphe. Ensuite, il définit la fonction "generate_non_random_node_name" pour générer de nouveaux noms de nœuds en se basant sur un index et un nom précédent. En utilisant cette fonction, il met à jour les noms des nœuds dans le premier ensemble de données " nodes.csv" en stockant les correspondances entre les anciens et les nouveaux noms de nœuds dans un dictionnaire appelé "node_mapping". Enfin, il met à jour les noms des nœuds dans le deuxième ensemble de données "edges.csv" en utilisant ce dictionnaire et enregistre les modifications dans de nouveaux fichiers Excel. "nodes_updated1.xls" et "edges_updated1.xls"

```python
import pandas as pd
# Charger les données à partir des fichiers CSV
nodes_df = pd.read_csv('/content/drive/MyDrive/nodes.csv')
edges_df = pd.read_csv('/content/drive/MyDrive/edges.csv')
# Fonction pour générer un nom de nœud non aléatoire basé sur un index
def generate_non_random_node_name(index, last_name):
    return str(index)
# Créer un dictionnaire pour stocker les correspondances entre les anciens et les nouveaux noms de nœuds
node_mapping = {}
# Changer les noms des nœuds dans la première base de données
new_node_names = []
for index, node in enumerate(nodes_df['node']):
    new_name = generate_non_random_node_name(index + 1, number_of_last_names)
    node_mapping[node] = new_name
    new_node_names.append(new_name)
nodes_df['node'] = new_node_names
# Changer les noms des nœuds dans la deuxième base de données en utilisant le dictionnaire de correspondance
edges_df['u'] = edges_df['u'].map(node_mapping)
edges_df['v'] = edges_df['v'].map(node_mapping)
# Enregistrer les modifications dans de nouveaux fichiers Excel (.xls)
nodes_df.to_excel('nodes_updated1.xls',  index=False, engine='openpyxl')
edges_df.to_excel('edges_updated1.xls',  index=False, engine='openpyxl')
print("Les noms des nœuds ont été mis à jour avec succès et les fichiers .xls ont été enregistrés.")
```

Figure 3.11. Capteur de code Python pour modifier la base de données.

	A	B	C
1	node	lat	lon
2	1	-37.807675	144.9558726
3	2	-37.8070943	144.9559785
4	3	-37.8073009	144.9578172
5	4	-37.8078848	144.9577193
6	5	-37.8052207	144.9521732
7	6	-37.8062498	144.9580023
8	7	-37.8203223	144.9675308
9	8	-37.8227347	144.9311156
10	9	-37.8262913	144.930619
11	10	-37.8262801	144.9307411
12	11	-37.8165761	144.9801473
13	12	-37.8240968	144.9307186

Figure 3.12. Capture de la BDD2 nodes_updated1.xls.

	A	B	C	D	E	F
1	Unnamed: 0	u	v	distance(m)	street type	speed(km/h)
2	0	1	694	58	1	15
3	2	1	2875	15	2	20
4	3	1	4	165	1	15
5	4	2	12316	4	0	10
6	5	2	694	8	1	15
7	7	2	517	44	0	10
8	8	3	540	8	2	20
9	10	3	12314	5	1	15
10	11	3	527	32	0	10

Figure 3.13. Capture de la BDD2 edges_updated1.xls.

C'est un graphe orienté et incomplet dans lequel chaque nœud est déterminé par ses coordonnées de longitude et de latitude, et les arêtes correspondent aux connexions entre les nœuds.

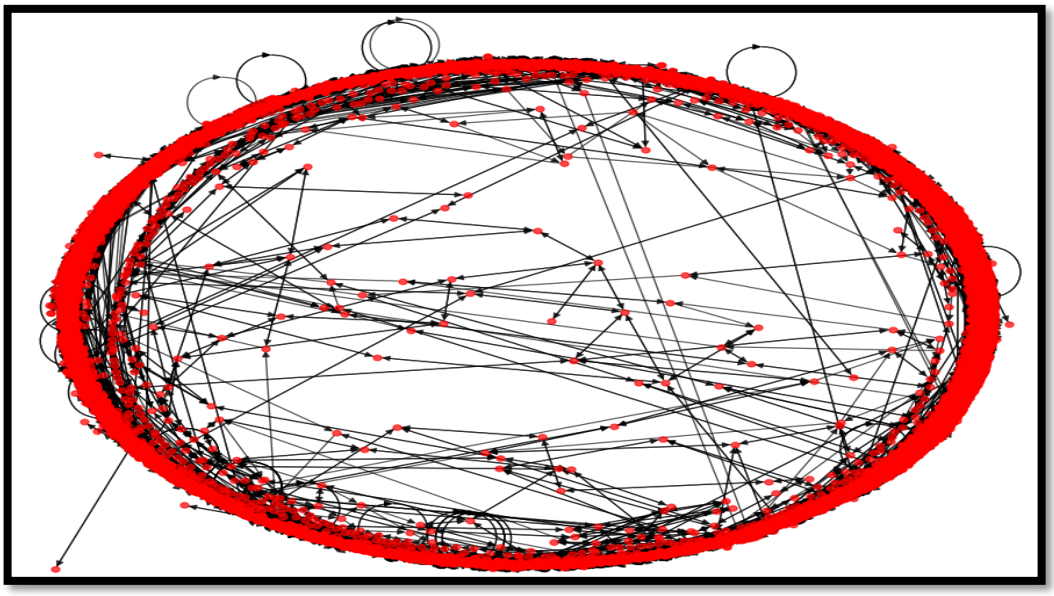


Figure 3.14. Graphe de base de données 2.

3.3.3. Matrices adjacentes

Tout d'abord, nous avons créé deux bases de données qui contiennent deux matrices symétriques adjacentes aléatoires : une matrice symétrique de temps et une matrice symétrique de distance. Les deux matrices ne sont pas orientées, complètes et multicritères. Elles contiennent toutes les deux les mêmes nœuds, soit de 1 à 1 000 nœuds et 499 500 arêtes chacune.

Les figures (3.15) et (3.16) représentent deux captures d'écran de deux matrices non orientées générées.

	A	B	C	D	E	F	G	H	I
1		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
2	**1**	0	272	200	861	568	771	645	439
3	**2**	272	0	691	857	131	366	17	582
4	**3**	200	691	0	18	434	430	91	79
5	**4**	861	857	18	0	544	757	306	546
6	**5**	568	131	434	544	0	12	548	924
7	**6**	771	366	430	757	12	0	989	411
8	**7**	645	17	91	306	548	989	0	474
9	**8**	439	582	79	546	924	411	474	0

Figure 3.15. Capture de la matrice symétrique de temps non orientée.

	A	B	C	D	E	F	G	H	I
1		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
2	**1**	0	772	661	140	161	25	542	101
3	**2**	772	0	837	414	807	137	659	600
4	**3**	661	837	0	557	974	463	645	843
5	**4**	140	414	557	0	709	79	941	907
6	**5**	161	807	974	709	0	822	447	357
7	**6**	25	137	463	79	822	0	650	815
8	**7**	542	659	645	941	447	650	0	17

Figure 3.16. Capture de la matrice symétrique de distance non orientée.

L'algorithme présente le code qui crée la base de données non orienté pour un nombre de nœuds donné en entrée.

Ce script Python génère deux matrices symétriques aléatoires de taille 1000x1000, représentant le temps et la distance, les stocke dans des Data Frames pandas, puis les enregistre dans des fichiers Excel distincts pour une utilisation ultérieure.

```python
import random
import pandas as pd
import networkx as nx
def generate_symmetric_matrix(n):
    matrix = [[random.randint(1, 1000) for _ in range(n)] for _ in range(n)]

    # Assurer la symétrie de la matrice
    for i in range(n):
        for j in range(i+1, n):
            matrix[j][i] = matrix[i][j]

    # Mettre à zéro les éléments sur la diagonale
    for i in range(n):
        matrix[i][i] = 0

    return matrix
# Générer deux matrices symétriques avec des étiquettes de nœuds numérotées de 1 à 1000
def generate_dataframe_with_labels(n):
    matrix1 = generate_symmetric_matrix(n)
    matrix2 = generate_symmetric_matrix(n)

    labels = [str(i) for i in range(1, n+1)]  # Numéroter les étiquettes de 1 à 1000

    # Convertir les matrices en DataFrame avec des étiquettes de nœuds pour les index et les colonnes
    df1 = pd.DataFrame(matrix1, index=labels, columns=labels)
    df2 = pd.DataFrame(matrix2, index=labels, columns=labels)  # Utiliser la même matrice pour les deux DataFrames

    return df1, df2
# Générer et enregistrer les DataFrames dans des fichiers Excel avec l'extension .xls
n = 1000
df1_with_labels, df2_with_labels = generate_dataframe_with_labels(n)
df1_with_labels.to_excel("/content/drive/MyDrive/bdd 4/matrice_symetrique de temps.xls", engine='openpyxl')
df2_with_labels.to_excel("/content/drive/MyDrive/bdd 4/matrice_symetrique de distance.xls", engine='openpyxl')
```

Figure 3.17. Capteur de code Python de Création base de données.

La figure (3.18) représente un graphe non orienté et complet, c'est le graphe de la BDD 3.

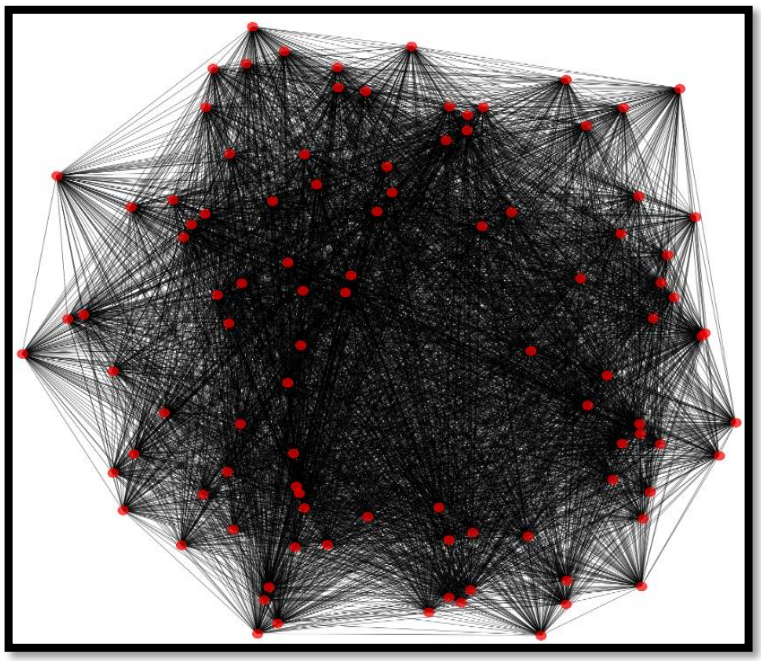


Figure 3.18. Graphe de base de données 3.

3.3.4. Chess Game Dataset (Lichess)

Il s'agit d'un ensemble d'un peu plus de 20000 jeux recueillis auprès d'une sélection d'utilisateurs sur le site Lichess.org, et comment collecter plus. Je téléchargerai également d'autres jeux à mesure que je les rassemblerai. Cet ensemble contient :

1) Game ID : Un identifiant unique attribué à chaque jeu.
2) Rated (T/F) : Indique si le jeu a été noté (Vrai ou Faux).
3) Start Time : La date et l'heure de début du jeu.
4) End Time : La date et l'heure de fin du jeu.
5) Number of Turns : Le nombre total de coups effectués dans le jeu.
6) Game Status : Le statut final du jeu.
7) Winner : Le joueur qui a gagné la partie.
8) Time Increment : L'incrément de temps ajouté par déplacement.
9) White Player ID : Un identifiant unique pour le joueur qui a joué en blanc.
10) White Player Rating : La note du joueur qui a joué comme blanc au moment de la partie.
11) Black Player ID : Un identifiant unique pour le joueur qui a joué en noir.
12) Black Player Rating : La notation du joueur qui a joué en noir au moment de la partie.
13) All Moves in Standard Chess Notation : Un enregistrement de chaque mouvement effectué pendant le jeu.
14) Opening Eco (Standardised Code for any given opening, list here) : L'Encyclopédie des ouvertures d'échecs (ECO) code représentant l'ouverture utilisée dans le jeu. Ces codes sont normalisés et consistent en une lettre (A-E) suivie d'un numéro (00-99), tel que B12.
15) Opening Name : Le nom de l'ouverture d'échecs jouée, correspondant au code ECO.
16) Opening Ply (Number of moves in the opening phase) : Le nombre de demi-coups (un coup par blanc ou noir) qui définissent la phase d'ouverture du jeu.

Chacun d'eux a des jeux séparés de Lichess. Il collecte ces données à l'aide de l'API Lichess, qui permet de collecter l'historique du jeu de tout utilisateur donné. La partie difficile a été de rassembler les noms d'utilisateurs à utiliser, mais l'API permet également de gratter tous les utilisateurs de l'équipe Lichess. Il y a beaucoup d'équipes à Lichess avec plus de 1500 joueurs,

donc cette méthode s'est avérée être un moyen efficace d'obtenir des utilisateurs de collecter des jeux auprès d'eux.

Beaucoup d'informations sont contenues dans un seul jeu d'échecs, sans parler d'un ensemble de données complet de plusieurs jeux. Il s'agit principalement d'un jeu de modèles, et la science des données consiste à détecter les modèles dans les données, c'est pourquoi les échecs ont été l'un des plus investis dans les domaines de l'IA dans le passé. Cet ensemble de données recueille toutes les informations disponibles de 20000 jeux et les présente dans un format facile à traiter pour l'analyse, par exemple, de ce qui permet à un joueur de gagner en noir ou en blanc, dans quelle mesure les facteurs méta (hors-jeu) affectent un jeu, la relation entre les ouvertures et la victoire pour le noir et blanc et plus encore.

La figure (3.19) représente un jeu d'échecs :

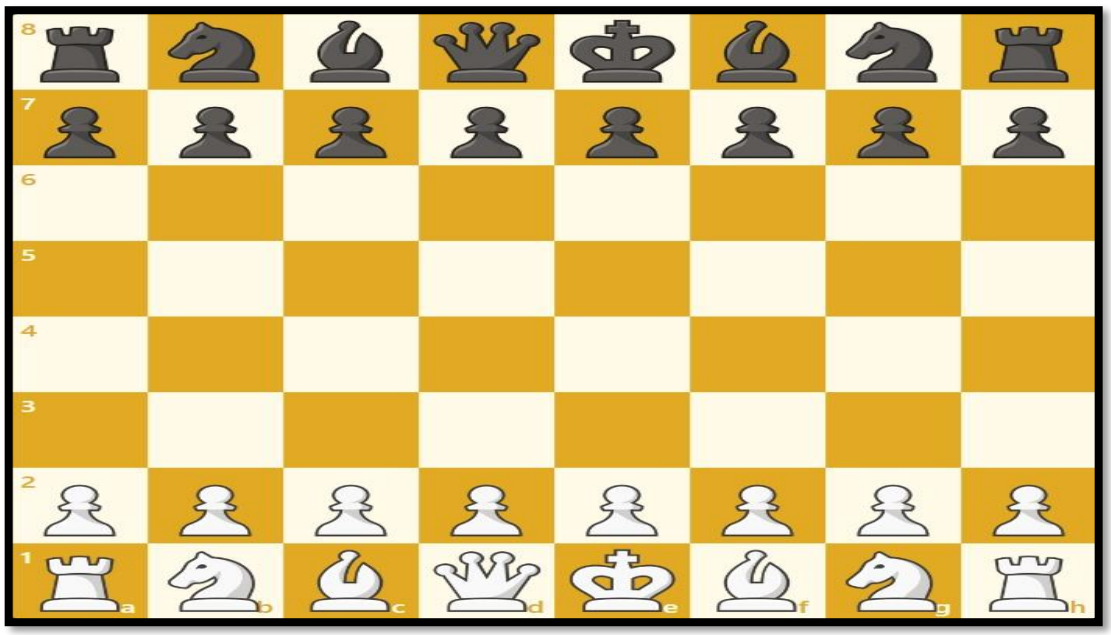


Figure 3.19. État initial de Jeu d'échecs.

Voici les positions initiales des pièces sur le plateau (pour le blanc et le noir) :

Roi (♔ pour les blancs, ♚ pour les noirs) :

- Position initiale : e1 pour les blancs, e8 pour les noirs.
- Mouvement : se déplace d'une case dans n'importe quelle direction (horizontale, verticale, diagonale).

Reine (♕ pour les blancs, ♛ pour les noirs) :

- Position initiale : d1 pour les blancs, d8 pour les noirs.
- Mouvement : se déplace de n'importe quel nombre de cases dans n'importe quelle direction (horizontale, verticale, diagonale). C'est la pièce la plus puissante du jeu.

Tour (♖ pour les blancs, ♜ pour les noirs) :

- Position initiale : a1 et h1 pour les blancs, a8 et h8 pour les noirs.
- Mouvement : se déplace de n'importe quel nombre de cases horizontalement ou verticalement.

Fou (♗ pour les blancs, ♝ pour les noirs) :

- Position initiale : c1 et f1 pour les blancs, c8 et f8 pour les noirs.
- Mouvement : se déplace de n'importe quel nombre de cases en diagonale.

Cavalier (♘ pour les blancs, ♞ pour les noirs) :

- Position initiale : b1 et g1 pour les blancs, b8 et g8 pour les noirs.
- Mouvement : se déplace en formant un "L" (deux cases dans une direction et une case perpendiculairement, ou une case dans une direction et deux cases perpendiculairement). Il est la seule pièce qui peut sauter par-dessus d'autres pièces.

Pion (♙ pour les blancs, ♟ pour les noirs) :

- Position initiale : rangée 2 pour les blancs, rangée 7 pour les noirs.
- Mouvement : se déplace d'une case vers l'avant. Lors du premier déplacement, un pion peut se déplacer de deux cases. Les pions capturent les pièces adverses en diagonale, d'une case vers l'avant. Les pions ont aussi une règle spéciale appelée "prise en passant" et peuvent être promus en une autre pièce (sauf en roi) lorsqu'ils atteignent la rangée de l'adversaire.

3.3.4.1. Les valeurs de chaque pièce

Après avoir lu cet article :

- **Determining Chess Piece Values Using Machine Learning**

(Gupta, 2023). Cet article explore l'optimisation multicritère « Determining Chess Piece Values Using Machine Learning » présente une analyse innovante des valeurs des pièces d'échecs à l'aide de méthodes d'apprentissage automatique. À l'aide d'une base de données de plus d'un million de parties d'échecs en ligne, les auteurs ont tenté de déterminer les valeurs pratiques des pièces d'échecs en fonction des déséquilibres physiques afin de prédire les résultats des parties. Cette approche se distingue des valeurs strictes traditionnellement acceptées et explore comment ces valeurs peuvent varier en fonction des différentes phases du jeu et des catégories de classification des joueurs.

De cela nous avons extrait les valeurs de chaque pièce :

- Roi : est pièce le plus importante de jeu mais il n'a pas de valeur numérique car il ne peut pas être Captur sa sécurité est primordiale.
- Pion =1 points.
- Reine= 9 points.
- Fou=3 points.
- Cavalier=3 points.
- Tours=5 points.

3.3.4.2.Encodage des éléments d'un jeu d'échecs avec des lettres.

L'encodage des éléments d'un jeu d'échecs avec des lettres est une méthode couramment utilisée pour représenter de manière concise et standardisée les pièces et leurs positions sur l'échiquier. Chaque pièce est représentée par une lettre spécifique :

- 'K' : Le Roi (King).
- 'Q' : La Reine (Queen).
- 'R' : Tour (Rook).
- 'B' : Fou (Bishop).
- 'N ': Cavalier (Knight).

- 'P' : Pion (pawn).

3.3.4.3. Traitement des données

A. Déchiffrage et Modélisation des Relations dans un Jeu d'Échecs :

- ✓ Lire la base de données et les différentes relations**.**
- ✓ Déchiffrer les relations **:** Interpréter les notations des mouvements des pièces, exemple :
  - **Nf3** : Le cavalier se déplace vers la case f3.
  - **h6** : Le pion avance vers la case h6.
  - **Qdxh3# :** la reine démarre de la ligne d vers la case h3 et elle retire (capturer) l'élément qui se trouve dans la case h3 et aussi elle déclare un échec et mat.
- ✓ **Créer le graphe**
  - Chaque étape est représentée par un nœud, dont tous les jeux commencent par le même nœud (1) mais elle termine par plusieurs nœuds. Chaque relation est représentée par deux arcs.
  - arc 1 : il représente l'existence d'une relation entre deux nœuds (toujours son poids égale 1).
  - arc 2 : il représente l'existence ou non d'une retiration d'un élément (Prendre un élément), il existe plusieurs poids dans ce arc.

La relation est représentée entre deux nœuds, le nœud source représente l'état précédent et le nœud de destination représente l'état suivant.

La figure (3.20) représente un graphe orienté et incomplet, c'est le graphe de la BDD 4.

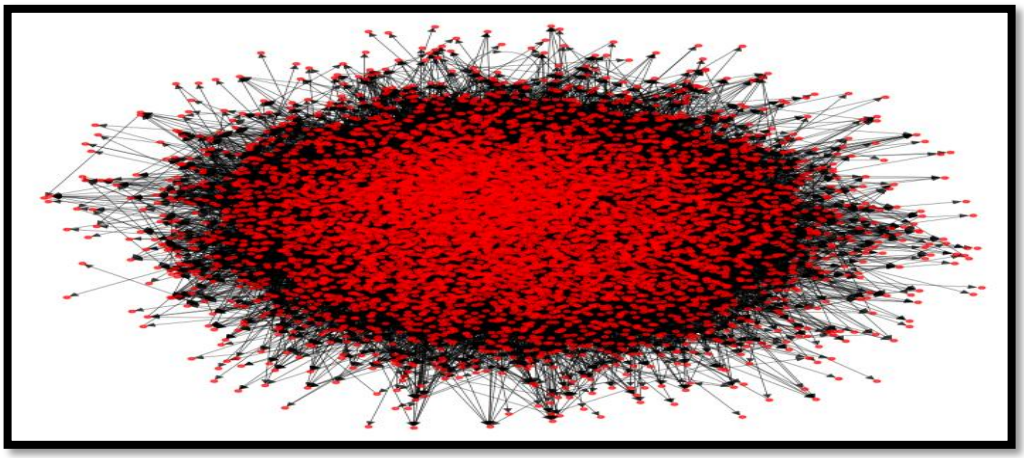


Figure 3.20. Graphe de base de données 4.

3.4. Tableaux de comparaison des bases de données

Ce tableau (3.1) contient une comparaison entre les bases de données utilisées :

Nom de bases de données	Nombre des nœuds	Nombre Des arêtes	Types de graphe	Optimisation	Multicritères / Uni-critère
Google Web Graph [1]	875713	5105039	Orienté Incomplet	Minimale	Uni-critère
Melbourne Restaurant Food Delivery data[2]	17117	42224	Orienté Incomplet	Minimale	Uni-critère
Matrices adjacentes	1000	499500	NON Orienté Complet	Minimale	Multicritères
Chess Game Dataset (Lichess)[3]	2638	4447	Orienté Incomplet	Maximale	Multicritères

Tableau 3.1. Comparaison des bases de données.

3.5. Les algorithmes utilisés

Tout d'abord, nous avons généré aléatoirement 100 paires source/destination pour chaque base de données, Par exemple pour la base de données des matrices adjacentes comme expliquées dans le tableau (3.2).

Numéro de Couple	Source	Destination
Couple1	271	682
Couple2	710	823
…	…	…
Couple 100	421	853

Tableau 3.2. Matrice des couples Source / Destination.

[1] https://www.kaggle.com/datasets/pappukrjha/google-web-graph A visité :15.02.2024
[2] https://www.kaggle.com/datasets/jianhanma/melbourne-restaurant-food-delivery-data
A visité : 22.02.2024
[3] https://www.kaggle.com/datasets/datasnaek/chess A visité : 28.02.2024

Pour notre approche, nous avons utilisé les 3 algorithmes d'optimisation. Ces algorithmes ont été appliqués aux couples :

3.5.1. Algorithme génétique

Un algorithme génétique est un algorithme de recherche et d'optimisation aléatoire et évolutif qui imite l'évolution biologique naturelle. Il a été appliqué pour résoudre divers problèmes dans différents domaines d'activité humaine et utilisé pour obtenir une solution optimale dans un délai raisonnable.

Les étapes d'algorithme génétique :

1) Générer une population aléatoire de n individus (chromosomes) : Il s'agit d'un ensemble d'individus, où chaque individu représente une solution au problème que nous souhaitons résoudre. Ainsi, une population est un ensemble de solutions potentielles.
2) Evaluer la performance (fitness) de chaque individu (chromosome) de la population :

   La fonction de fitness évalue la qualité de chaque solution
3) La sélection consiste à définir une méthode de choix des individus (parents) dans la population, afin d'appliquer les opérateurs de croisement et de mutation pour produire de nouveaux individus (enfants).
4) Le croisement a pour but d'enrichir la diversité de la population en manipulant la structure des chromosomes existants. En général, les croisements sont envisagés avec deux parents et génèrent deux enfants. Plusieurs des Opérateurs de croisement :

   - Croisement uni point : la méthode de croisement uni point dans les algorithmes génétiques, deux parents sont sélectionnés parmi la population. Ensuite, un point de croisement est choisi au hasard sur leurs chromosomes (qui sont essentiellement des séquences de gènes). Les parties des chromosomes situés après ce point de croisement sont échangées entre les parents pour donner naissance à deux enfants. Les enfants ainsi obtenus héritent ainsi d'une partie des gènes de chaque parent.
   - Croisement bipoint : Deux points de coupure sont sélectionnés de manière aléatoire sur les solutions parentes. Les parties situées entre ces deux points de coupure sont échangées entre les deux parents pour créer de nouvelles solutions, appelées enfants. Cette méthode permet de générer une diversité de solutions tout en conservant certaines caractéristiques des solutions parentes.

5) La mutation se fait en utilisant un opérateur de mutation, et il faut donc définir un opérateur de mutation pour le problème à résoudre. La mutation vise à améliorer la qualité des individus. Il existe 3 principaux types de mutation :

   - ❖ Insertion : Une mutation par insertion est appliquée à la solution. Un gène est sélectionné au hasard dans la séquence de la solution, puis inséré à une position différente, également sélectionnée au hasard dans la même séquence. Ce processus permet d'introduire des modifications locales dans la solution sans altérer sa structure globale, ce qui peut améliorer la diversité de la population et conduire à de meilleures solutions.
   - ❖ Swap : Lorsqu'une mutation par swap est appliquée à une solution, deux positions de la séquence de la solution sont sélectionnées de manière aléatoire, puis les éléments correspondants à ces positions sont échangés. En d'autres termes, deux gènes de la solution sont choisis au hasard et échangés.
   - ❖ Réversion : La mutation par réversion est particulièrement utile pour explorer de nouvelles régions de l'espace des solutions et pour éviter de rester coincé dans des optimaux locaux. Elle contribue ainsi à la diversité de la population et peut conduire à de meilleures solutions dans le cadre de l'optimisation de problèmes.

6) Critères d'arrêt : Les critères d'arrêt sont des conditions définies pour mettre fin à l'exécution d'un algorithme, Ces critères sont essentiels pour garantir que l'algorithme ne tourne pas indéfiniment et pour permettre de contrôler les ressources de calcul et le temps d'exécution.

3.5.2. Algorithme Dijkstra

L'algorithme de Dijkstra (prononcer approximativement « Dextra ») permet de trouver le plus court chemin entre deux sommets d'un graphe (orienté ou non orienté).

L'algorithme de Dijkstra fonctionne de manière itérative pour trouver le chemin le plus court, Voici sa méthode de fonctionnement :

1) Initialisation des distances : au début de l'algorithme, les distances sont initialisées à l'infini pour tous les nœuds sauf le nœud source, pour lequel la distance est mise à zéro, On maintient également un ensemble de nœuds non traités.
2) Sélection du nœud le plus proche : à chaque itération, on sélectionne le nœud non traité avec la distance minimale parmi ceux encore non visités.

3) Mise à jour des distances :
   - Pour chaque voisin du nœud sélectionné, on calcule la distance totale depuis le nœud source en passant par ce voisin.
   - Si cette distance est plus courte que la distance actuellement connue pour le voisin, on met à jour sa distance.
4) Marquage du nœud sélectionné comme traité : Une fois qu'on a mis à jour les distances de tous les voisins du nœud sélectionné, on marque ce dernier comme traité (ou visité). Il est retiré de l'ensemble des nœuds non traités.
5) Répétition jusqu'à ce que tous les nœuds soient traités : On répète ce processus jusqu'à ce que tous les nœuds aient été traités. À chaque itération, on sélectionne le nœud non traité avec la distance minimale, et on met à jour les distances de ses voisins, Si la destination est inaccessible à partir du nœud source, l'algorithme renvoyé Nan pour la distance et le chemin.
6) Reconstruction du chemin le plus court (optionnel) : Une fois que toutes les distances ont été calculées, on peut reconstruire le chemin le plus court entre le nœud source et n'importe quel autre nœud en utilisant les informations sur les prédécesseurs stockées pendant le processus.
7) Terminaison de l'algorithme : L'algorithme se termine lorsque tous les nœuds ont été traités, ou lorsque le nœud destination (si spécifié) a été traité.

L'algorithme de Dijkstra garantit que lorsque tous les nœuds ont été traités, la distance la plus courte de chaque nœud au nœud source a été trouvée. Cependant, il peut ne pas fonctionner correctement si le graphe contient des arêtes de poids négatif. Dans ce cas, l'algorithme de Bellman-Ford est souvent utilisé.

3.5.2. Algorithme A*

L'algorithme A* est un algorithme de recherche de chemin qui étend l'algorithme de Dijkstra en utilisant une heuristique pour guider la recherche afin de trouver le chemin le plus court entre des paires de nœuds dans un graphe.

Les étapes d'algorithme A* :

1) Initialisation des distances : Tout comme dans l'algorithme de Dijkstra, les distances des nœuds sont initialisées à l'infini, sauf pour le nœud source qui est initialisé à zéro.

2) Initialisation des structures de données : L'algorithme commence par initialiser les structures de données nécessaires, telles que les ensembles ou les dictionnaires pour garder une trace des nœuds à explorer, des coûts des chemins parcourus et des scores estimés pour chaque nœud.

3) Évaluation des nœuds : A* évalue chaque nœud en utilisant deux coûts :

   - Le coût réel du départ jusqu'au nœud courant (g_score).
   - Une estimation du coût du nœud courant jusqu'à l'objectif (f_score). Cela inclut le coût réel déjà parcouru (g_score) et une estimation heuristique de la distance restante (heuristic).

4) Sélection du nœud courant : À chaque itération, A* sélectionne le nœud ayant le plus faible score f parmi ceux qui sont encore à explorer.

5) Arrêt si l'objectif est atteint : Si le nœud sélectionné est l'objectif recherché, l'algorithme reconstruit le chemin optimal depuis le départ jusqu'à l'objectif.

6) Mise à jour des scores : Pour chaque nœud voisin du nœud courant, A* met à jour les scores g et f si un meilleur chemin est trouvé.

7) Exploration des voisins : A* explore les voisins du nœud courant qui ne sont pas encore explorés. Pour chaque voisin, il calcule le coût total du chemin en ajoutant le coût de l'arête entre le nœud courant et le voisin au coût du chemin parcouru jusqu'au nœud courant.

8) Itération : L'algorithme continue d'itérer jusqu'à ce que tous les nœuds possibles aient été explorés ou que l'objectif soit atteint. Si aucun chemin optimal n'est trouvé, l'algorithme renvoie null.

3.6. Conclusion

Dans ce chapitre, nous avons examiné les études récentes et pertinentes dans le domaine de l'optimisation, en mettant particulièrement l'accent sur la résolution du problème du chemin le plus court. Nous avons identifié les forces et les faiblesses des méthodes actuelles, ainsi que les besoins et les obstacles à surmonter. Notre approche propose l'utilisation de méta-heuristiques pour aborder ces défis. Nous avons également détaillé la méthodologie de comparaison des algorithmes, ainsi que l'environnement et les outils de développement utilisés. De plus, nous avons décrit les quatre bases de données sélectionnées pour évaluer notre algorithme, mettant en lumière leurs propriétés et caractéristiques, ainsi que les étapes de pré-traitement des données. Enfin, nous avons présenté

les algorithmes utilisés, notamment l'algorithme génétique, A*, et Dijkstra, et avons expliqué la génération aléatoire de paires source/destination pour chaque base de données. L'architecture globale de notre approche a été esquissée pour fournir une vue d'ensemble cohérente. Ce chapitre jette ainsi les bases nécessaires à une étude exhaustive et méthodique dans le cadre de notre projet d'optimisation du chemin le plus court.

Chapitre 4

Résultats et discussion

4.1. Introduction

Dans ce chapitre, nous explorons l'implémentation et la comparaison des algorithmes d'optimisation (algorithme de Dijkstra, algorithme génétique et algorithme A*) discutés dans le chapitre 3. Notre objectif est d'analyser, d'évaluer et de comparer les résultats obtenus en appliquant ces algorithmes à quatre bases de données uni-critère et multicritère.

Le but de cette étude est d'obtenir :

- Trouvez le chemin le plus court.
- Comparaison des temps d'exécution des trois algorithmes.

4.2. Les paramètres de l'algorithme génétique

Nous avons travaillé avec les paramètres suivants :

- La taille de la population = 100 et 500 et 1000.
- Nombre d'itérations =100 et 500 et 1000.

Les opérateurs génétiques sont appliqués à la population de solutions pour créer une nouvelle génération de solutions. Où les opérateurs génétiques sont appliqués à la population de solutions pour créer une nouvelle génération de solutions. Voici une explication détaillée de chaque partie :

1. **Sélection des meilleurs individus :** sélectionne les meilleurs 20 % des individus de la population actuelle en fonction de leur fitness. Ces individus seront utilisés comme parents pour les opérateurs génétiques.
2. **Mutation par insertion :** Appliquer l'insertion à une partie de la population, de 20% à 40% de la population.
3. **Mutation par swap :** Appliquer swap à une partie de la population, de 40% à 60% de la population.
4. **Croisement (crossover) à un point :** Appliquer croisement à une partie de la population, de 60% à 80% de la population.
5. **Génération de nouvelles solutions aléatoires :** Appliquer à 80% à 100% de la population, Créez une nouvelle solution en sélectionnant aléatoirement des nœuds entre la source et la destination et en mettant à jour le contenu avec la nouvelle solution générée.

6. **Vérification de la solution finale :** Après avoir généré la nouvelle population, vérifiez si une solution satisfait les critères, (début et fin à la source et à la destination) Si une solution valide cette solution est retournée comme la meilleure solution.

Ces étapes représentent le cœur de l'algorithme génétique, où la population évolue au fil des générations pour converger vers une solution optimale au problème donné.

4.3. Analyse et évaluation des performances des algorithmes appliqués aux bases de données Uni-critère

Une évaluation détaillée des algorithmes utilisant des bases de données à critère unique, tels que :

4.3.1. Résultats de base de données 1(Google Web Graph)

Après avoir appliqué les algorithmes à la base de données, nous expliquerons les résultats des 10 couples que nous avons choisies au hasard, afin de voir en détail les résultats obtenus.

Algorithmes	Génétiques		Djikstra		A*	
N. Couples	Distance	TE (s)	Distance	TE (s)	Distance	TE (s)
Couple 15	12	224.023	16	21.293	16	15.868
Couple 86	11	224.005	11	10.755	11	5.692
Couple 85	14	214.016	14	17.200	14	12.3871
Couple 61	NAN	214.000	13	2.690	13	2.341
Couple 1	NAN	196.008	10	10.831	10	5.999
Couple 95	NAN	216.008	8	0.927	8	0.827
Couple 52	13	224.017	13	15.939	13	11.315
Couple 48	18	199.003	18	19.242	18	14.482

Couple 5	10	214.005	10	9.8111	10	7.8124
Couple 18	9	220,102	9	1.506	9	1.179
AVG	12.43	214.52	12.20	11.02	**12.20**	7.79

Tableau 4.1. Comparaison des 3 algorithmes sur les couples choisis au hasard BDD1.

Ce tableau (4.1) représente les résultats de l'application et de la comparaison des performances de trois algorithmes : les Algorithmes Génétiques, Dijkstra et A*, appliqués à une base de données contenant des paires de données sélectionnées aléatoirement. Chaque ligne du tableau correspond à une paire spécifique de source et de destination, et chaque colonne présente les résultats des algorithmes en termes de distance et de temps d'exécution.

Pour chaque case du tableau, l'algorithme correspondant a calculé le plus court chemin entre la source et la destination donnée. Les distances obtenues représentent la longueur de ce chemin, tandis que les temps d'exécution indiquent la durée nécessaire à chaque algorithme pour trouver cette solution.

La figure (4.1) représente un graphique comparant les distances entre les trois algorithmes : Dijkstra, A* et Génétiques.

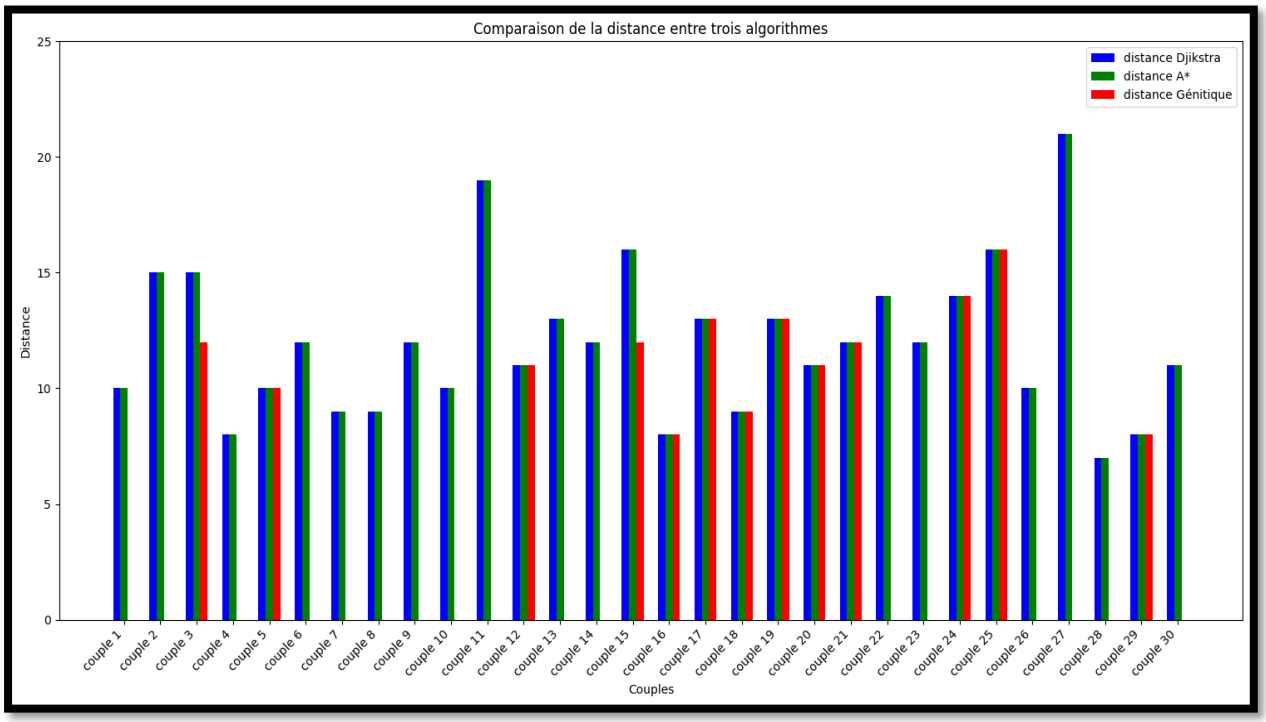


Figure 4.1. Comparaison de la distance entre trois algorithmes BDD1.

Sur le graphique dans les données fournies, nous remarquons que dans plusieurs cas, tant l'algorithme de Dijkstra que celui d'A* produisent le chemin le plus court, comme le montrent les distances identiques pour certains couples. Cela suggère une performance similaire en termes de recherche du chemin optimal entre ces deux algorithmes. Cependant, il arrive que l'algorithme génétique ne parvienne pas à fournir de solution, comme indiqué par les "NAN" dans la colonne de distance pour certains couples, indiquent qu'un résultat n'a pas pu être calculé Cela est dû au fait que le graphe n'est pas complet.

Enfin, on remarque qu'en utilisant Dijkstra et A*, on obtient la même distance, et parfois l'algorithme génétique donne également la même distance.

La figure (4.2) représente un graphique comparant les Temps d'exécution entre les trois algorithmes : Dijkstra, A* et Génétiques.

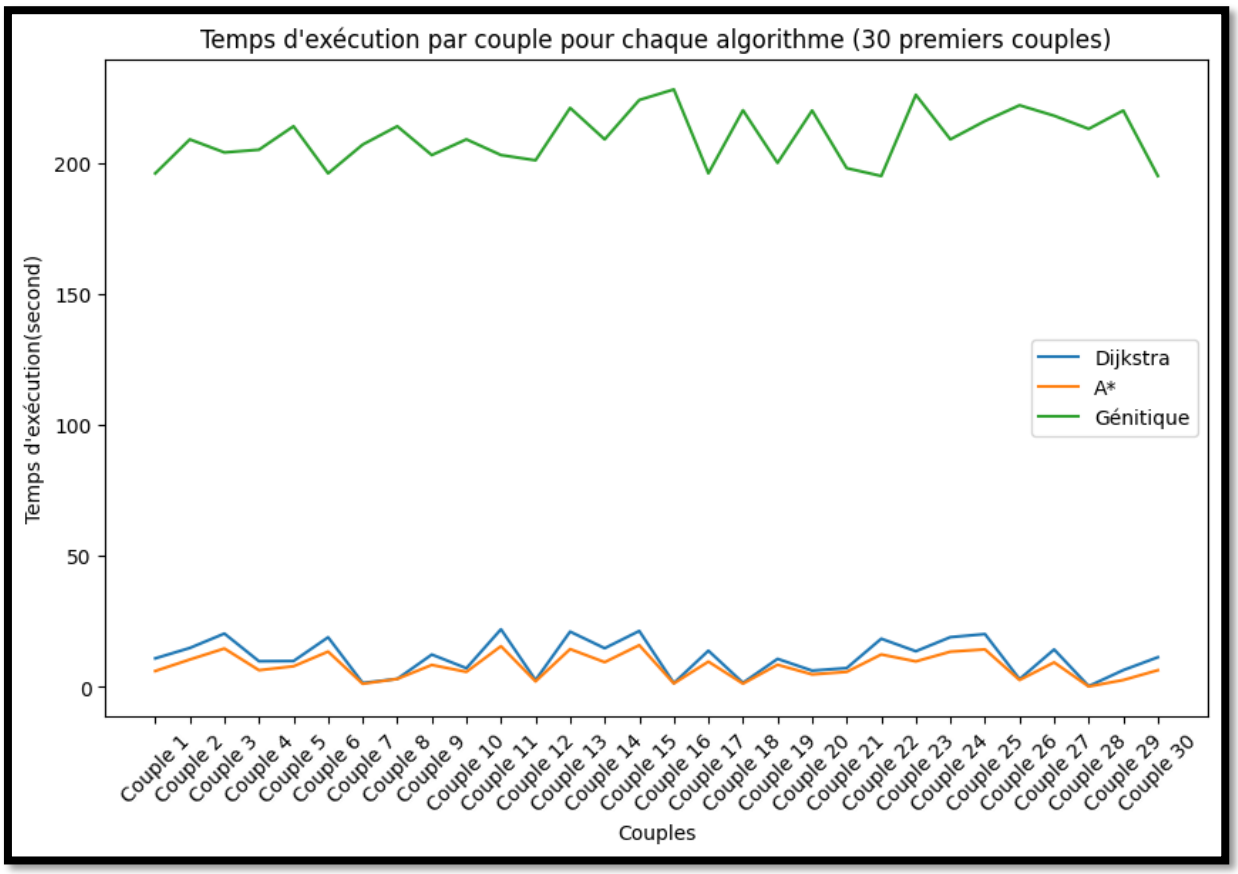


Figure 4.2. Temps d'exécution par couple pour chaque algorithme (30 premiers couples) BDD1.

Le graphique montre les temps d'exécution (en secondes) pour trois algorithmes : Dijkstra, A* et Génétique, à travers 30 paires différentes (couples). Voici une comparaison :

- Dijkstra et A* ont des temps d'exécution relativement similaires, restant constamment faibles sur les 30 paires.
- Génétique a des temps d'exécution significativement plus élevés que Dijkstra et A*.
- Dijkstra et A* : Leur temps d'exécution est généralement inférieur à 50 secondes, avec des pics mineurs occasionnels.
- Génétique : Les temps d'exécution de cet algorithme restent constamment au-dessus de 200 secondes, avec quelques variations mais restant dans la même plage élevée.

En résumé, Dijkstra et A* surpassent l'algorithme Génétique par une marge significative en termes de temps d'exécution, Dijkstra et A* étant tout à fait comparables. L'algorithme Génétique, bien que cohérent, a des temps d'exécution beaucoup plus élevés, Enfin, on note que le temps d'exécution obtenu rapidement par Dijkstra ou A* est le choix le plus adapté.

4.3.2. Résultats de base de données 2 (Melbourne Restaurant Food Delivery data)

Après avoir appliqué les algorithmes à la base de données, nous analyserons les résultats des 10 couples que nous avons choisis au hasard. Nous examinerons en détail les résultats obtenus.

Algorithmes	Génétiques		Djikstra		A*	
N. Couples	Distance	TE (s)	Distance	TE (s)	Distance	TE (s)
Couple13	43	0.492	43	0،004	43	0،012
Couple 9	21	0،253	21	0،004	21	0،014
Couple 6	182	0،254	182	0,003	182	0,016
Couple 18	NAN	0،513	NAN	0.004	NAN	0.023
Couple 100	83	0،262	83	0،003	83	0،016
Couple 8	391	0،274	41	0,003	41	0,017
Couple 40	4	0،264	4	0،004	4	0،014

Couple 58	71	0،485	71	0،004	71	0،016
Couple 30	5	0،253	5	0،004	5	0،015
Couple 94	148	0،281	148	0،004	148	0،027
AVG	105.33	0.33	**66.44**	0.0037	66.44	0.01703

Tableau 4.2. Comparaison des 3 algorithmes sur les couples choisis au hasard BDD2.

Ce tableau (4.2) compare les performances des trois algorithmes (algorithmes génétiques, Dijkstra et A *) en termes de temps d'exécution et de distance, Nous avons appliqué ces algorithmes à une base de données, Nous avons sélectionné 10 nœuds couple (source et destination) au hasard,

Dont les résultats sont résumés dans ce tableau pour déterminer l'algorithme le plus approprié, et Les valeurs "NaN" indiquent qu'un résultat n'a pas pu être calculé.

La figure (4.3) représente un graphique comparant les Temps d'exécution entre les trois algorithmes : Dijkstra, A* et Génétiques.

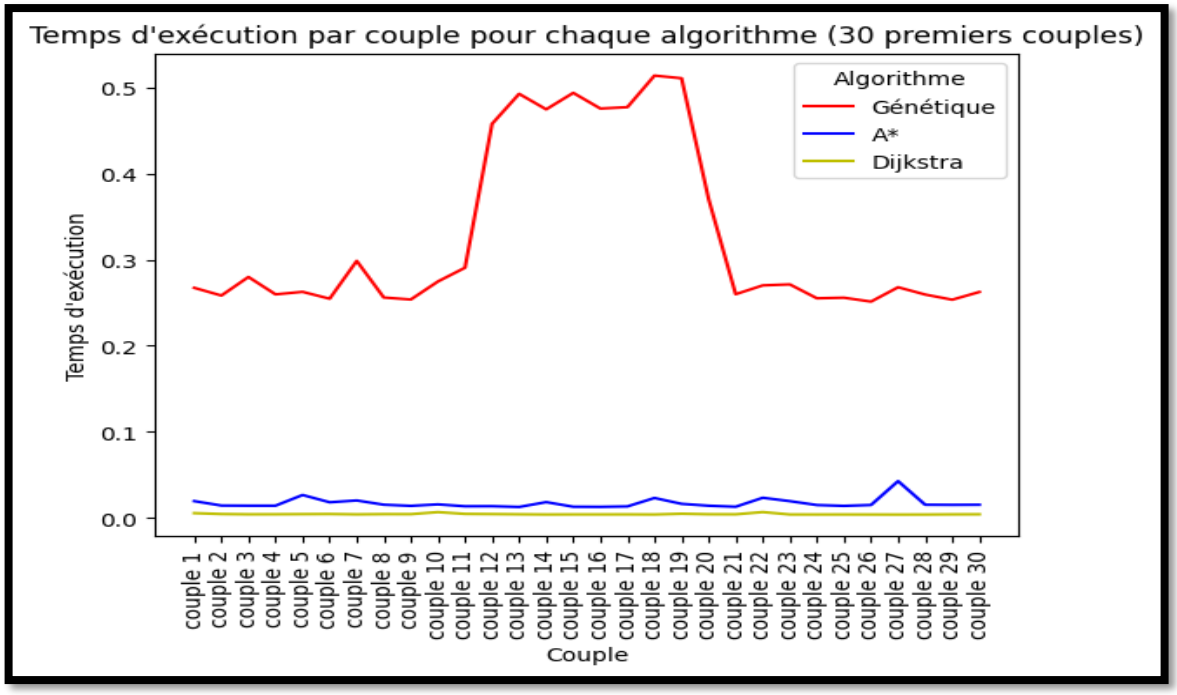


Figure 4.3. Temps d'exécution par couple pour chaque algorithme (30 premiers couples) BDD2.

En ce qui concerne les temps d'exécution, les résultats indiquent que Dijkstra se révèle être l'algorithme le plus rapide en moyenne parmi les trois. Ses temps d'exécution sont généralement plus courts que ceux des algorithmes génétiques et de A*. Bien que A* affiche des temps d'exécution légèrement plus longs que Dijkstra dans la plupart des cas, ils restent inférieurs à ceux des algorithmes génétiques. Cette différence peut être attribuée à la nature probabiliste des algorithmes génétiques, qui nécessitent souvent de nombreuses itérations pour converger vers une solution. Ainsi, les résultats suggèrent que si la rapidité d'exécution est une priorité, Dijkstra est préférable. Cependant, A* reste compétitif en termes de temps d'exécution et offre souvent une meilleure qualité de solution, tandis que les algorithmes génétiques peuvent être plus lents en raison de leur processus itératif.

La figure (4.4) représente un graphique comparant les distances entre les trois algorithmes : Dijkstra, A* et Génétiques :

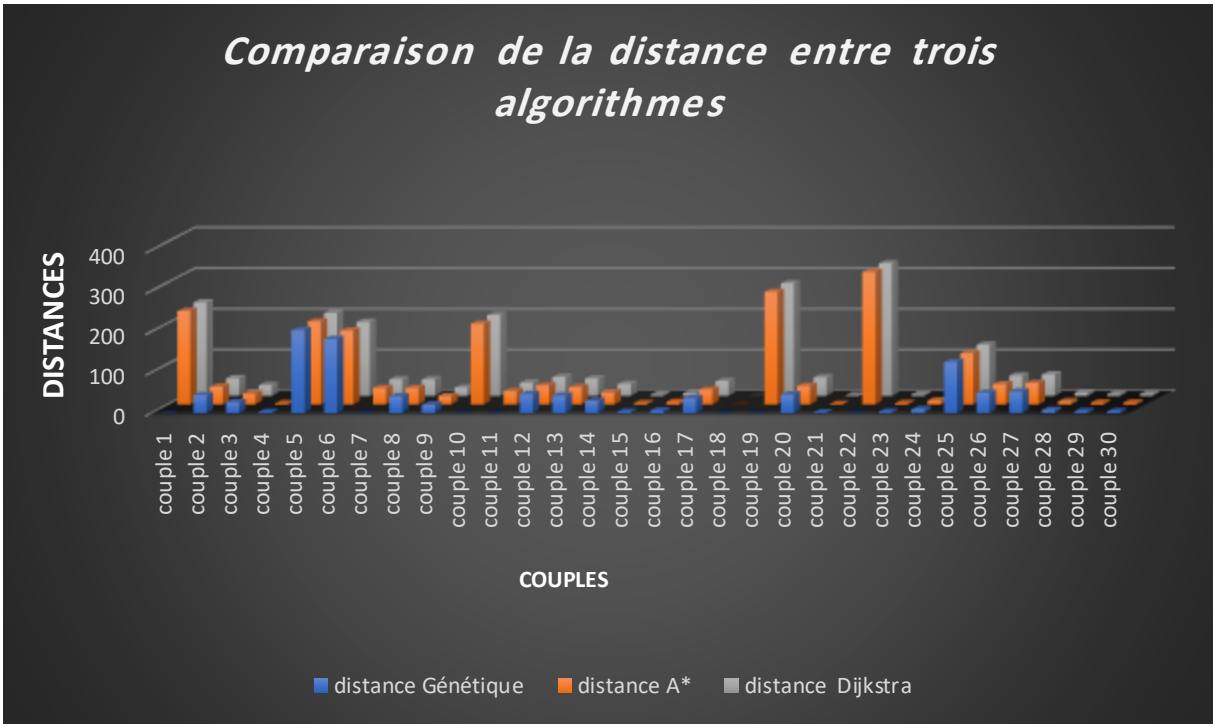


Figure 4.4. Comparaison de la distance entre trois algorithmes BDD2.

A travers ce graphique, nous remarquons que les distances obtenues par les algorithmes A* et Dijkstra sont identiques pour toutes ces paires, ce qui montre que parfois les deux algorithmes produisent les mêmes résultats. Les distances génétiques (NaN) sont absentes pour certains couples. L'algorithme génétique ne fournit pas de solution C'est pour plusieurs raisons.

Enfin, les algorithmes de Dijkstra et A* ont produit la même distance pour tous les couples, et parfois l'algorithme génétique donne également la même distance pour certains couples.

Ce tableau (4.3) représente les résultats après ajustement des paramètres génétiques BDD2.

Paramètre	Partie 1	Partie 2	Partie 3	Partie 4
Taille population	1000	100	500	500
Nb itération	100	100	100	500
Moyenne des solutions existe	47.32	47.32	47.32	47.32
Temps d'exécution (s)	0.627	0.0567	0.3073	1.542

Tableau 4.3. Les résultats après ajustement des paramètres génétiques BDD2.

Le tableau (4.3) présente les résultats d'expérimentations sur un algorithme génétique appliqué à un Base de données "BDD2". Il montre comment la taille de la population et le nombre d'itérations affectent le temps d'exécution et la qualité des solutions. Les tailles de population testées étaient de 1000, 100, 500 et, tandis que les nombres d'itérations étaient de 100 et 500. Les résultats indiquent que la moyenne des solutions trouvées reste constante à 47,32 pour toutes les configurations, ce qui signifie que les variations de ces paramètres n'affectent pas la qualité moyenne des solutions. Cependant, le temps d'exécution varie significativement : il est de 0,627 seconde pour une population de 1000 et 100 itérations, de 0,0567 seconde pour une population de 100 et 100 itérations, de 0,3073 seconde pour une population de 500 et 100 itérations, et de 1,542 seconde pour une population de 500 et 500 itérations. Ainsi, il est clair que le temps d'exécution augmente avec la taille de la population et le nombre d'itérations.

4.4. Analyse et évaluation des performances des algorithmes appliqués aux bases de données Multicritère

Après avoir lu cet article et cette thèse de doctorat :

- **Multiobjective-Based Decision-Making for the Optimization of an Urban Passenger Traffic System Structure** (Zhang, 2023). Cet article parle de l'étude de la congestion routière urbaine à Harbin, en Chine. Les chercheurs ont développé un modèle multi-cible visant à améliorer le système de transport de passagers en prenant en compte les émissions de carbone, les coûts de transport et l'utilisation des ressources.

Ils ont utilisé des données opérationnelles sur les différents modes de transport et les déplacements des passagers à Harbin.

- **Les techniques d'optimisation multicritère en optimisation à deux niveaux** (Pieume, 2011). Cette thèse explore l'optimisation multicritère et à deux niveaux en se concentrant sur les méthodes de résolution, les applications pratiques et les liens entre ces deux domaines. Elle développe des méthodes pour résoudre les problèmes de programmation linéaire multicritère et à deux niveaux en introduisant des approches innovantes pour caractériser et trouver les solutions efficaces. Parmi les applications pratiques, on compte la planification de la distribution de l'énergie électrique au Cameroun et la protection des industries locales contre la concurrence internationale. La thèse examine également les conditions de Pareto-optimalité et propose des approches pour résoudre des problèmes d'optimisation complexes.

Étant donné que les bases de données 'Adjacency Matrix' et 'Chess Game Dataset (Lichess)' sont multicritères, nous souhaitons appliquer l'optimisation multicritère à ces deux bases de données en utilisant trois méthodes, comme le suggèrent cet article et cette thèse.

- Les trois méthodes :

1) A+B
2) A*B
3) Ax+By

4.4.1. Résultats de base de données 3 (Matrices adjacentes)

1) La méthode A+B

Après avoir appliqué des algorithmes à 100 couples, nous avons sélectionné au hasard 5 couples, comme indiqué dans ce tableau (4.4).

Algorithmes	Génétiques		Djikstra		A*	
N. Couples	Distance + temps	TE (s)	Distance + temps	TE (s)	Distance + temps	TE (s)
Couple 2	257	0،805	245	0،253	257	0،0117
Couple 99	514	0،955	199	0،248	506	0،013

Couple 10	470	0،505	283	0،908	378	0،014
Couple 21	291	0،509	271	0،890	291	0،015
Couple 40	491	0،519	192	0،504	491	0،014
AVG	404.6	0.658	**238**	0.5606	384.6	0.0135

Tableau 4.4. Comparez les trois algorithmes sur des paires sélectionnées aléatoirement en utilisant la méthode (A+B) BDD3.

Ce tableau (4.4). Compare les performances de trois algorithmes (algorithmes génétiques, Dijkstra et A*) en termes de temps d'exécution et la somme (distance + temps). Nous avons choisi cinq paires de nœuds (source et destination) au hasard et avons utilisé la méthode A + B, où A représenté la distance et B représente le temps. Notre objectif est de trouver un chemin entre une source et une destination où la somme (distance + temps) est minimale le long du chemin.

La figure (4.5) représente un graphique comparant les Temps d'exécution entre les trois algorithmes : Dijkstra, A* et Génétiques, utilisé la méthode A + B.

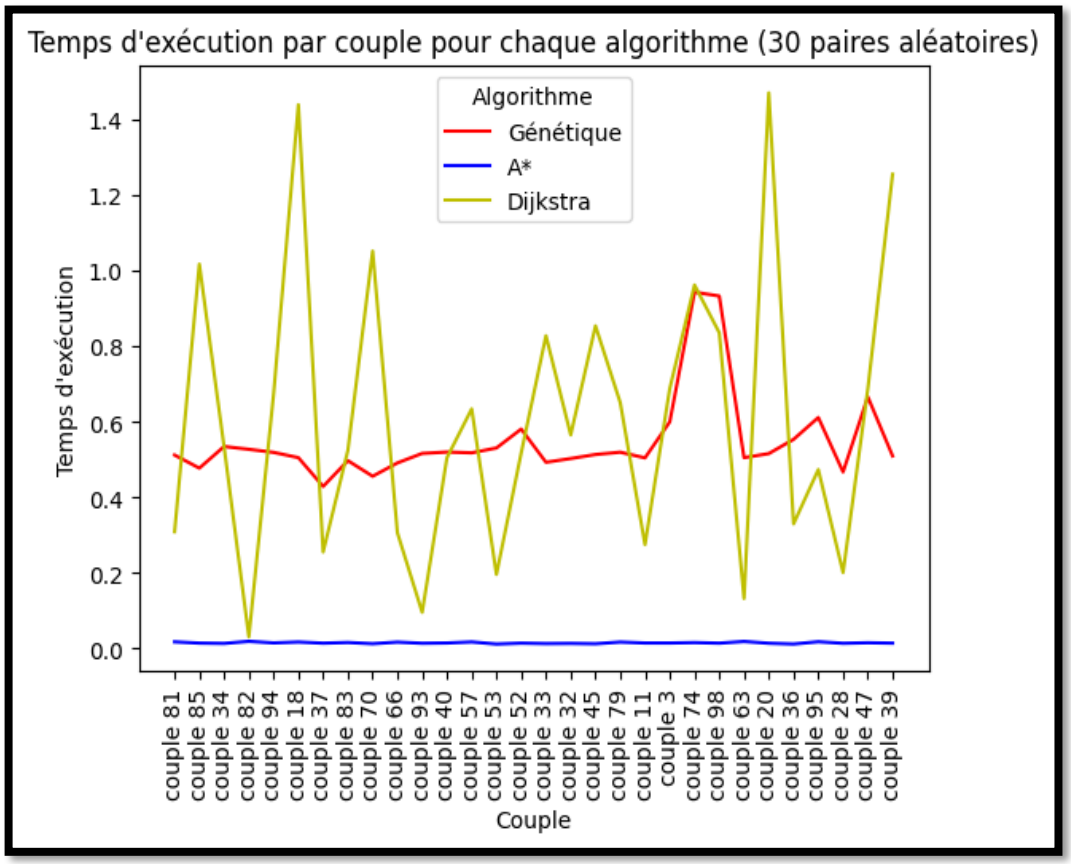


Figure 4.5. Temps d'exécution par couple pour chaque algorithme (30 paires aléatoires) en utilisant la méthode (A+B) BDD3.

La courbe représente le temps d'exécution de chaque algorithme (Algorithme génétique, A* et Dijkstra) en fonction de 30 paires aléatoires. Elle montre comment le temps d'exécution varie en fonction des paires.

- L'algorithme A* semble avoir le temps d'exécution le plus bas de tous. Il est généralement plus efficace que les autres.
- L'algorithme génétique présente une plus grande variabilité, Cela suggère une complexité potentielle dans certains cas.
- L'algorithme Dijkstra avec des pics atteignant les temps d'exécution les plus élevés de tous les algorithmes.

La figure (4.6) représente un graphique comparant les (distance + temps) entre les trois algorithmes : Dijkstra, A* et Génétiques utilisé la méthode A + B.

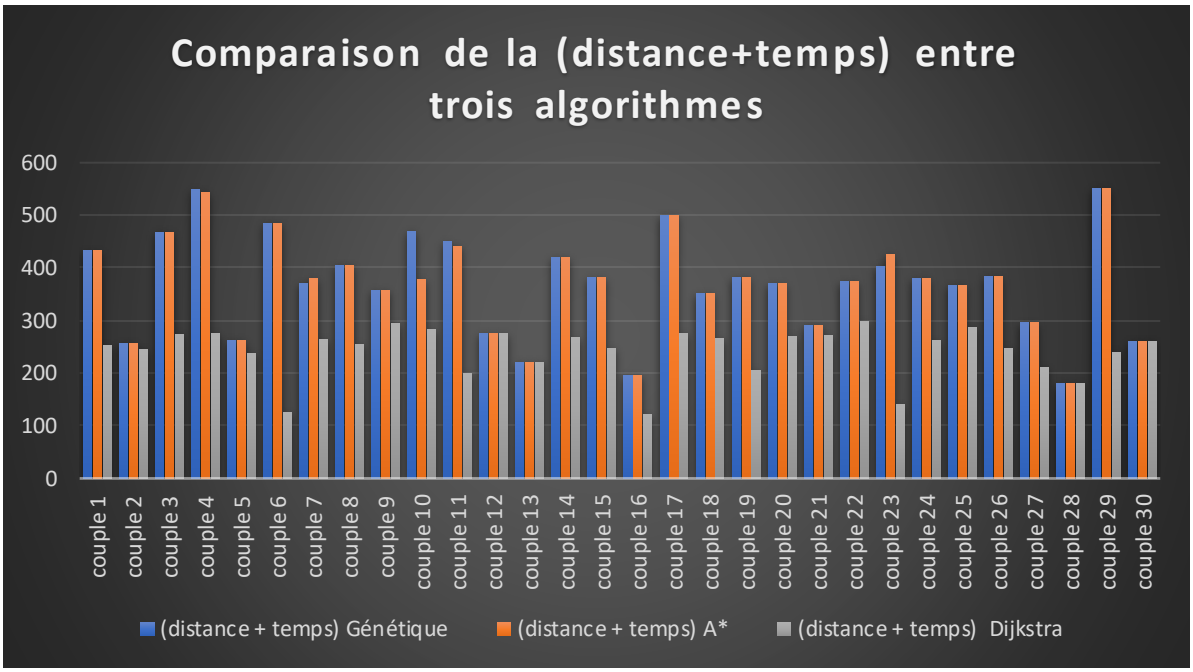


Figure 4.6. Comparaison de là (distance + temps) entre trois algorithmes BDD3.

L'analyse des courbes révèle que l'algorithme de Dijkstra surpasse systématiquement les autres en termes de performance, en offrant les (distance + temps) les plus courtes. Les algorithmes génétiques et A* ont montré des performances similaires pour certaines paires, mais dans plusieurs cas, l'algorithme A* a surpassé l'algorithme génétique. Tandis que l'algorithme génétique reste moins performant dans ces cas spécifiques.

2) La méthode A*B

Après avoir appliqué des algorithmes à 100 couples, nous avons sélectionné au hasard 5 couples, comme indiqué dans ce tableau (4.5).

Algorith-mes	Génétiques		Djikstra		A*	
N. Couples	Distance * temps	TE (s)	Distance * temps	TE (s)	Distance*temps	TE (s)
Couple 4	11242	0,509	1175	1,152	3157	0,013
Couple 30	4496	0.538	1200	0.651	4496	0.016
Couple 41	9343	0.527	681	0.039	7204	0.014
Couple100	2774	0.501	1115	0.275	2774	0.022
Couple 66	1676	0.490	883	0.293	1679	0.014
AVG	5906.2	0.513	**1010.8**	0.482	3862	0.0158

Tableau 4.5. Comparaison des 3 algorithmes sur les couples choisis au hasard en utilisant la méthode (A*B) BDD3.

Ce tableau (4.5) compare les performances de trois algorithmes (algorithmes génétiques, Dijkstra et A*) en termes de temps d'exécution et de multiplicatif (distance * temps). Nous avons sélectionné au hasard cinq paires de nœuds (source et destination) et avons utilisé la méthode A*B, où A représenté la distance et B représente le temps. Notre objectif est de trouver un chemin entre une source et une destination où la multiplication (distance * temps) est minimale le long du chemin.

La figure (4.7) représente un graphique comparant les Temps d'exécution entre les trois algorithmes : Dijkstra, A* et Génétiques, utilisé la méthode A * B.

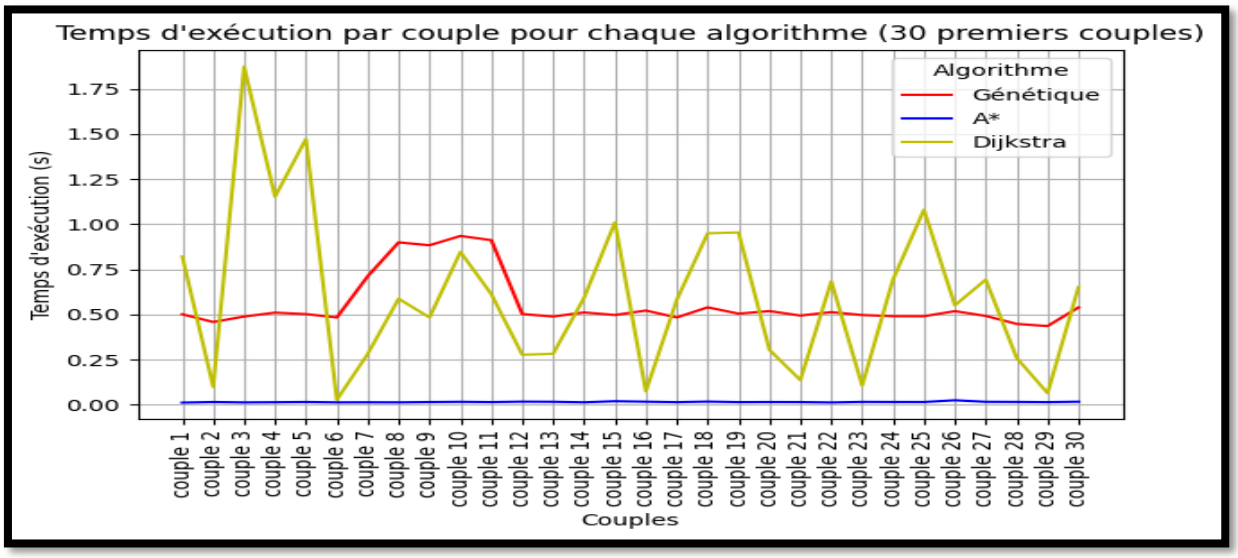


Figure 4.7. Temps d'exécution par couple pour chaque algorithme (30 premiers couples) en utilisant la méthode (A*B) BDD3.

La courbe représente le temps d'exécution de chaque algorithme (Algorithme génétique, A* et Dijkstra) en fonction de 30 premiers couples. Elle montre comment le temps d'exécution varie en fonction des paires.

L'algorithme A* se distingue par son temps d'exécution généralement le plus bas, ce qui en fait l'option la plus efficace par rapport aux autres. En revanche, l'algorithme génétique montre une plus grande variabilité dans ses temps d'exécution, suggérant une complexité potentielle dans certains cas spécifiques. À l'autre extrémité du spectre, l'algorithme de Dijkstra affiche parfois des temps d'exécution plus élevés, atteignant des pics significatifs. Ces observations soulignent la supériorité d'A* en termes d'efficacité, tout en mettant en évidence la variabilité des performances des algorithmes génétiques et la lenteur potentielle de Dijkstra dans certaines situations.

La figure (4.8) représente un graphique comparant les (distance * temps) entre les trois algorithmes : Dijkstra, A* et Génétiques, en utilisant la méthode A*B.

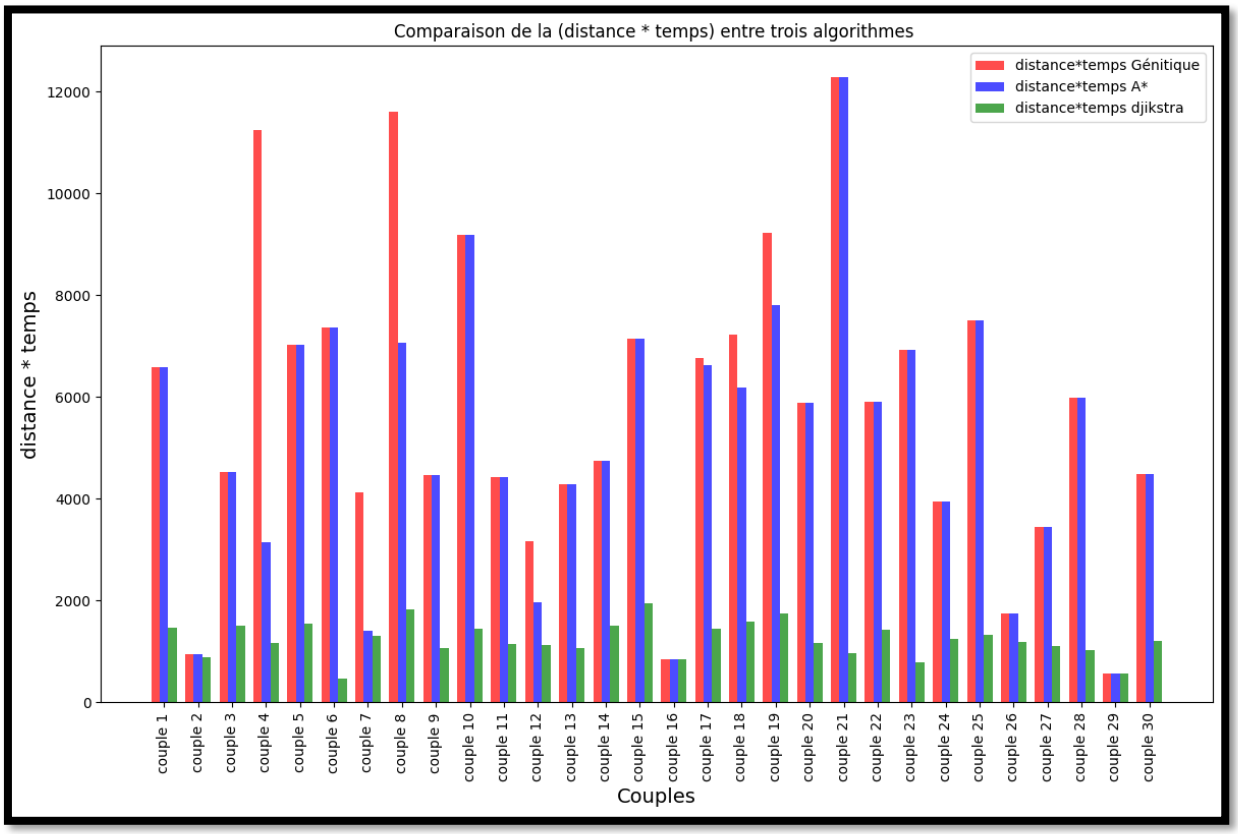


Figure 4.8. Comparaison de la (distance * temps) entre trois algorithmes BDD3.

Le graphique montre la comparaison du produit de la distance par le temps pour trois algorithmes (Génétique, A*, et Dijkstra) appliqués à 30 couples. Ce produit (distance * temps) permet de mesurer l'efficacité globale des algorithmes en tenant compte à la fois de la qualité de la solution (distance * temps).

- Dijkstra (Vert) tend à être le plus performant, offrant les valeurs de (distance * temps) les plus basses, ce qui en fait le choix optimal pour minimiser le coût global.
- A* (Bleu)suit de près, montrant une performance raisonnable avec des valeurs modérées.
- Algorithme Génétique (Rouge)affiche des coûts plus élevés, reflétant une moindre efficacité globale pour ce critère.

Pour certains couples, comme le couple 4 et le couple 21, la différence entre les algorithmes est particulièrement prononcée, avec l'algorithme génétique donnant des valeurs très élevées.

Pour d'autres couples, comme les couples 10 et 30, les valeurs de (distance * temps) sont plus proches entre A* et Dijkstra, indiquant une performance comparable.

Enfin, La comparaison révèle que l'algorithme de Dijkstra est souvent le plus performant en termes de minimisation du produit de la distance, suivi de près par l'algorithme A* et l'algorithme génétique.

3) La méthode Ax + By

Définir des coefficients pour la fonction linéaire : (x * distance + y * times)

- x = 2.0, y = 3.0 , où le temps est plus important que la distance.
- A : distance
- B : times

Après avoir appliqué les algorithmes à la base de données, nous expliquerons les résultats des 5 couples que nous avons choisies au hasard, afin de voir en détail les résultats obtenus.

Algorit-hmes	**Génétiques**		**Djikstra**		**A***	
N. Couples	**Ax+ By**	**TE (s)**	**Ax+ By**	**TE (s)**	**Ax+ By**	**TE (s)**
Couple 6	1268	0,497	349	0,054	1268	0.014
Couple 12	692	0,892	680	1,004	692	0,012
Couple 18	743	0,562	650	1,049	743	0,016
Couple 99	1139	0,508	487	0,265	1139	0,014
Couple 90	850	0,531	710	1,131	850	0,015
AVG	938.4	0.598	**575.2**	0.7006	938.4	0.0142

Tableau 4.6. Comparaison des 3 algorithmes sur les couples choisis au hasard en utilisant la méthode (Ax + By) BDD3.

Ce tableau compare les performances de trois algorithmes (algorithmes génétiques, Dijkstra et A*) en termes de temps d'exécution et de composition (x*distance + y*temps). Nous avons sélectionné au hasard cinq paires de nœuds (source et destination) et avons utilisé la méthode (x*distance +y*temps), où 'x' représenté le nombre 2 et 'y' représente le nombre 3. Notre objectif est de trouver un chemin entre une source et une destination où la composition (x * distance + y * temps) est minimale le long du chemin.

La figure (4.9) représente un graphique comparant les Temps d'exécution entre les trois algorithmes : Dijkstra, A* et Génétiques, utilisé la méthode (x*distance + y*temps).

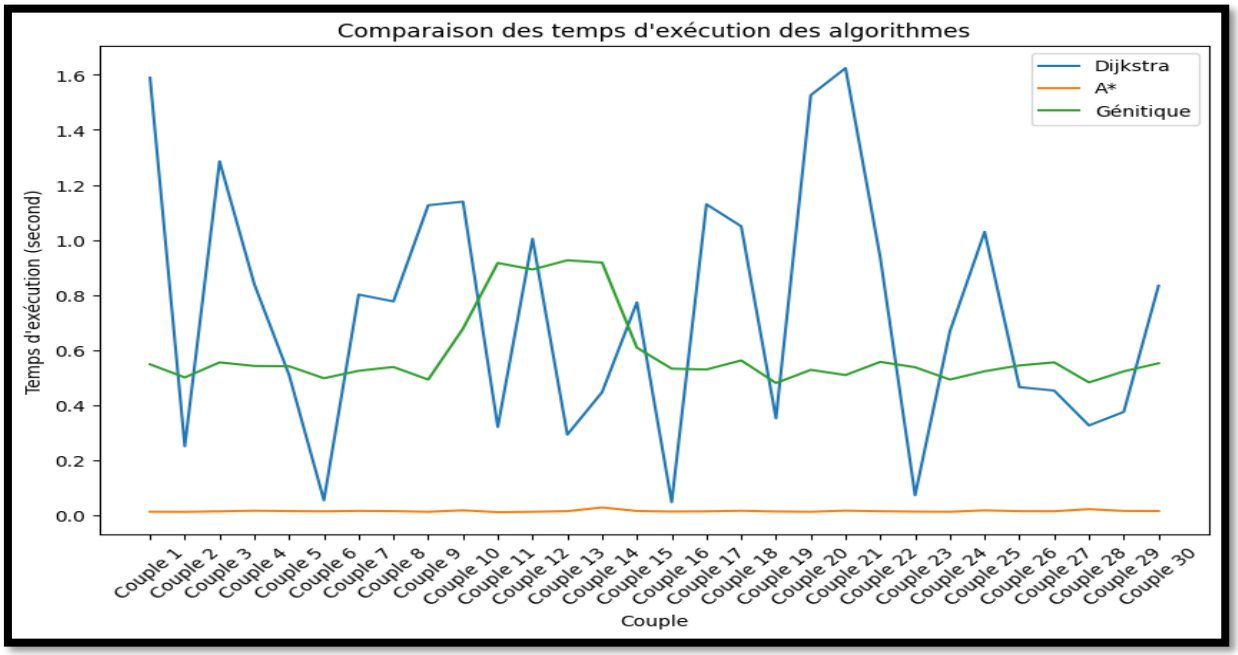


Figure 4.9. Temps d'exécution par couple pour chaque algorithme (30 premiers couples) en utilisant la méthode (Ax + By) BDD3.

Le graphique montre une comparaison des temps d'exécution pour trois algorithmes différents : Dijkstra, A* et Génétique, appliqués à 30 couples. L'algorithme Dijkstra (ligne bleue) affiche des variations significatives de temps d'exécution, allant de 0,1 seconde à plus de 1,6 seconde, indiquant une variabilité élevée et une dépendance potentielle à la structure spécifique de chaque couple. L'algorithme génétique (ligne verte), quant à lui, montre une performance plus stable, oscillant généralement autour de 0,6 à 0,8 seconde, ce qui suggère une meilleure consistance mais un temps d'exécution globalement plus élevé par rapport à Dijkstra dans certains cas. En revanche, l'algorithme A* (ligne orange) se distingue par ses temps d'exécution extrêmement bas et constants, presque indiscernables en bas du graphique, ce qui indique une performance très rapide et stable à travers tous les couples testés. En résumé, A* est le plus rapide et le plus stable, tandis

que Dijkstra varie considérablement en temps d'exécution, et l'algorithme génétique offre une stabilité modérée mais à un coût de temps d'exécution plus élevé.

La figure (4.10) représente un graphique comparant le résultat entre les trois algorithmes : Dijkstra, A* et Génétique ont utilisé la méthode (x*distance + y*temps).

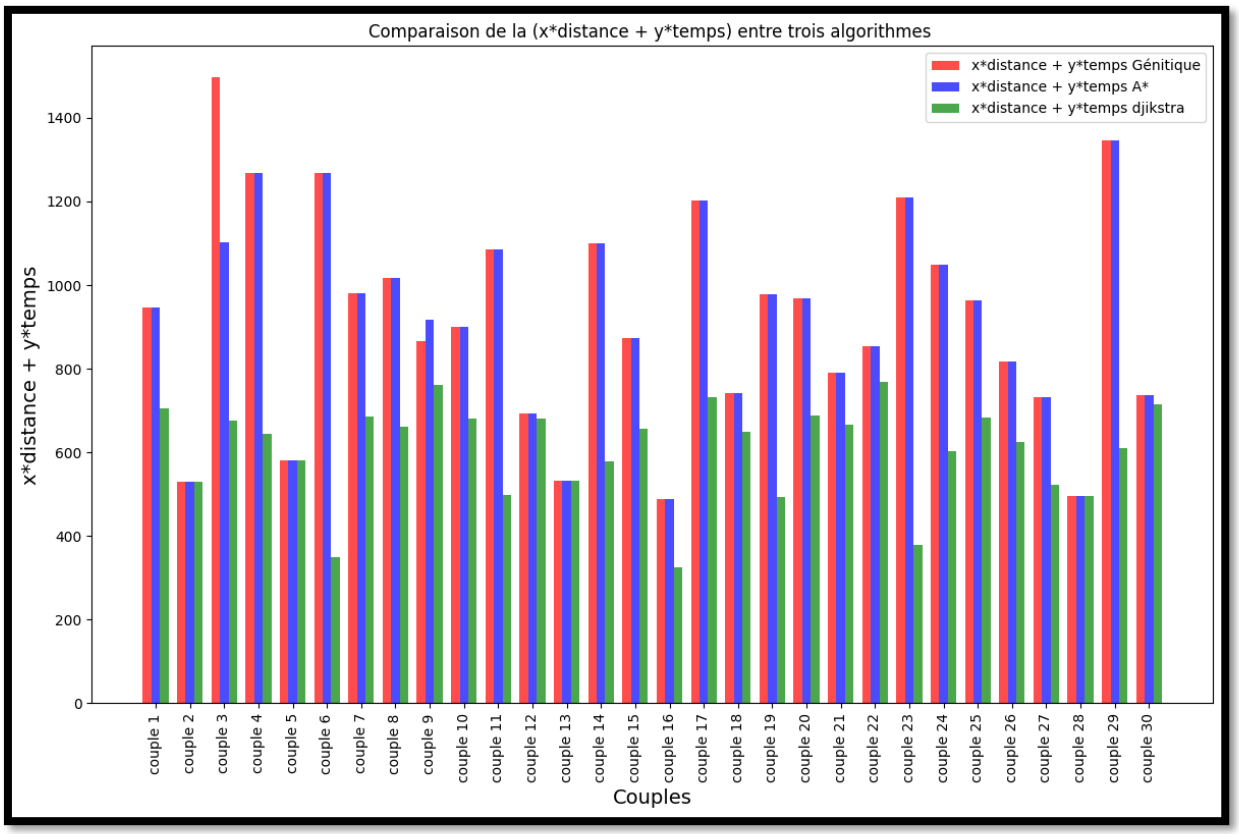


Figure 4.10. Comparaison de la (x*distance + y*temps) entre trois algorithmes BDD3.

Le graphique montre le résultat de la méthode (x*distance + y*temps) calculée par trois algorithmes différents (Génétique, A* et Dijkstra) pour 30 couples. Chaque ensemble de trois barres représente un résultat trouvé pour une paire donnée par les trois algorithmes.

- Dijkstra (vert) : Trouve généralement les distances les plus courtes.
- A (bleu) : Offre des distances optimales, parfois légèrement supérieures à celles de Dijkstra.
- Génétique (rouge) : Souvent donne des distances plus élevées, indiquant une efficacité moindre pour minimiser la distance dans ces instances spécifiques.
- Pour certains couples (comme le couple 4 et le couple 20), la différence entre les algorithmes est plus prononcée. Pour d'autres couples, les distances entre A* et

Dijkstra sont très similaires, ce qui montre une bonne heuristique pour A* dans ces cas, Dijkstra est souvent le plus performant pour minimiser la distance.

Ce tableau (4.7) représente les résultats après ajustement des paramètres génétiques en utilisant la méthode (A+B) BDD3.

Paramètre	Partie 1	Partie 2	Partie 3	Partie 4
Taille population	1000	100	500	500
Nb itération	100	100	100	500
Moyenne des solutions existe	357.49	453.72	367.6	362.02
Temps d'exécution (s)	1.463	0.142	0.709	3.190

Tableau 4.7. Les résultats après ajustement des paramètres génétiques en utilisant la méthode (A + B) BDD3.

Ce tableau (4.7) présente les résultats obtenus après l'ajustement des paramètres génétiques en utilisant la méthode (A + B) BDD3. Voici quelques conclusions clés à tirer de ces résultats :

- Les solutions moyennes obtenues sont les plus petites pour la partie 1 (357.49), avec la taille de la population (1000). Les solutions obtenues avec une population de 100 (partie 2) sont légèrement moins bonnes que celles obtenues.
- Le temps d'exécution est le plus bas pour la Partie 2 (0.142 secondes), qui utilise une taille de population de 100. Le temps d'exécution le plus élevé est observé dans la Partie 4 (3.190 secondes), malgré une taille de population et un nombre d'itérations relativement élevés.

En conclusion, pour cette méthode spécifique, la partie 1 semble offrir la meilleure solution. Les résultats indiquent que l'optimisation des paramètres doit prendre en compte la qualité de la solution et l'efficacité du temps.

4.4.2. Résultats de base de données 4 (Chess Game Dataset (Lichess))

1) La méthode A+B

- A : Arc1 (mouvement).
- B : Arc2 (score).

Après avoir appliqué des algorithmes à 100 couples, nous avons sélectionné 5 couples, comme indiqué dans ce tableau (4.8).

Algorithmes	Génétiques		Djikstra		A*	
N. Couples	**Arc 1+ Arc2**	**TE (s)**	**Arc 1+ Arc2**	**TE (s)**	**Arc 1+ Arc2**	**TE (s)**
Couple 1	22	0.983	22	0,045	22	0,046
Couple 2	NAN	0,955	25	0,105	25	0,102
Couple 3	11	0,904	11	0,007	11	0,007
Couple 4	22	0,832	22	0,078	22	0,075
Couple 5	NAN	1,045	23	0,104	23	0,088
AVG	18.33	0.9439	20.60	0.0678	**20.60**	0.0636

Tableau 4.8. Comparaison des 3 algorithmes sur les couples choisis en utilisant la méthode (A+B) BDD4.

Le tableau (4.8) compare les performances de trois algorithmes de recherche de chemin : Génétique, Dijkstra et A*, sur une sélection de couples choisis aléatoirement à partir de la base de données BDD4. Et avons utilisé la méthode A + B, où A représenté Arc1 (mouvement) et B représente Arc 2 (score). Notre objectif est de trouver un chemin entre une source et une destination où la somme (Arc 1+Arc 2) est maximale le long du chemin

La figure (4.11) représente un graphique comparant les Temps d'exécution entre les trois algorithmes : Dijkstra, A* et Génétiques, utilisé la méthode (A+ B).

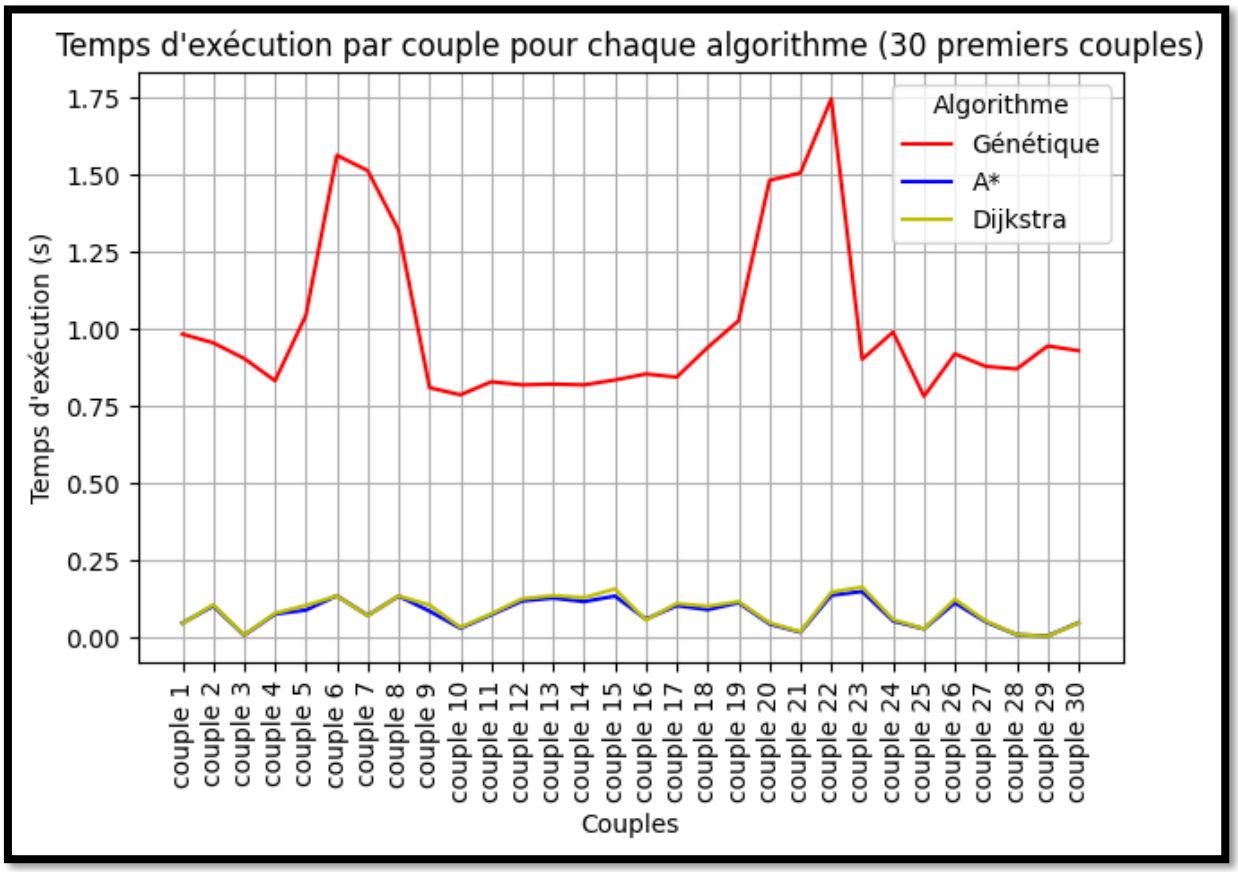


Figure 4.11. Temps d'exécution par couple pour chaque algorithme (30 premier couples) en utilisant la méthode (A+B) BDD4.

Le graphique montre une comparaison des temps d'exécution de trois algorithmes (Génétique, A*, Dijkstra) sur les 30 premiers couples. Les résultats indiquent que l'algorithme Génétique (en rouge) a des temps d'exécution significativement plus élevés et plus variables, tandis que les algorithmes A* (en bleu) et Dijkstra (en jaune) ont des temps d'exécution beaucoup plus courts et constants, démontrant une meilleure efficacité pour ces couples.

- Algorithme Génétique : Temps d'exécution élevé et variable.
- Algorithme A* : Temps d'exécution faible.
- Algorithme Dijkstra : Temps d'exécution faible.

Pour des performances optimales et constantes, les algorithmes A* et Dijkstra sont préférables.

La figure (4.12) représente un graphique comparant le résultat entre les trois algorithmes : Dijkstra, A* et Génétique ont utilisé la méthode A + B.

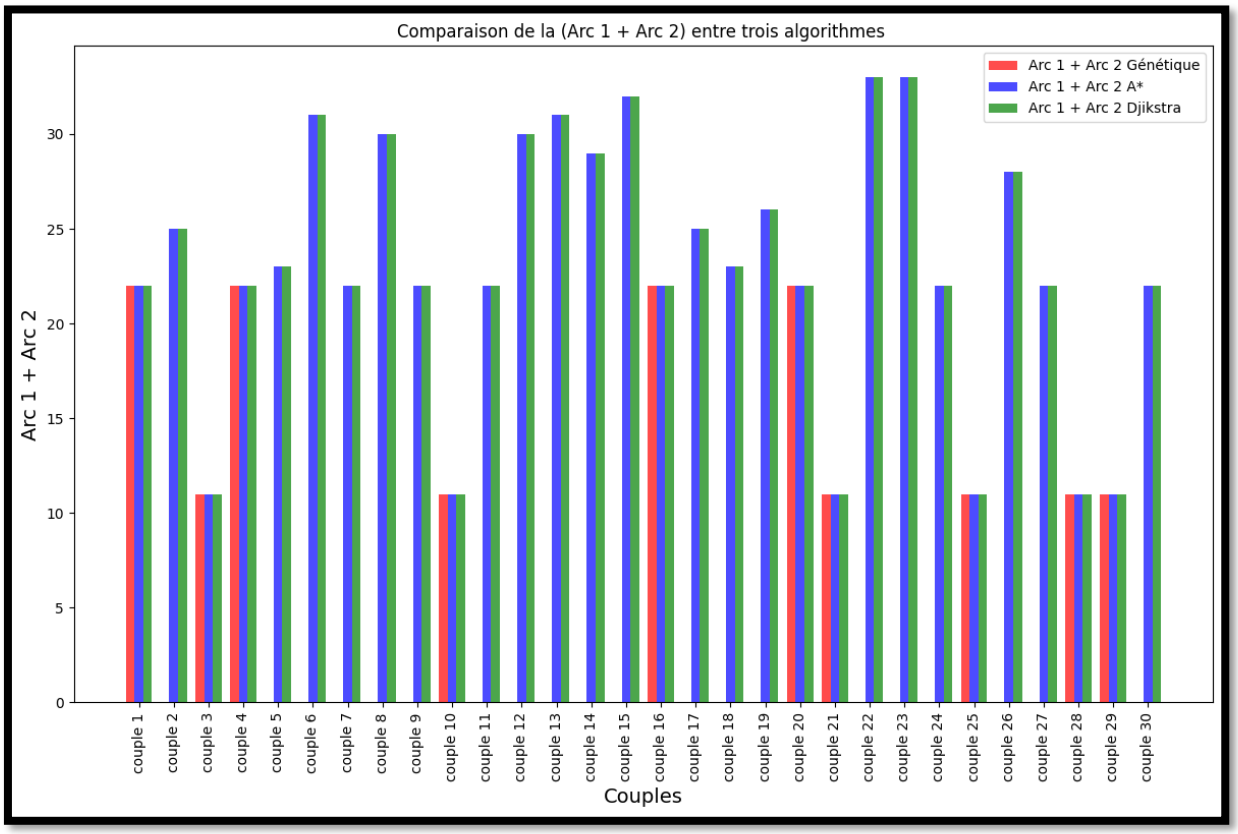


Figure 4.12. Comparaison de la (arc1+arc2) entre trois algorithmes BDD4.

Le graphique compare la somme des (arc1 + arc2) pour trois algorithmes (Génétique, A*, Dijkstra) sur plusieurs couples.

Notre objectif est de rechercher un chemin dans lequel la somme des (arc1 + arc2) est maximale

- Algorithme Génétique : la somme des (arc1 + arc2) est légèrement inférieure pour certains couples mais sont globalement similaires aux autres algorithmes.
- Algorithme A* : Montre la somme des (arc1 + arc2) cohérentes et souvent égales à celles de Dijkstra, indiquant une similarité dans l'optimisation des chemins.
- Algorithme Dijkstra :la somme des (arc1 + arc2) sont en grande partie similaires à celles de l'algorithme A*, confirmant l'efficacité de ces deux algorithmes pour l'optimisation des chemins.

Dans l'ensemble, les algorithmes A * et Dijkstra montrent des performances presque identiques en termes la somme des (arc1 + arc2) calculées, tandis que l'algorithme génétique montre de légères différences.

2) La méthode A*B

- A : Arc1 (mouvement).
- B : Arc2 (score).

Après avoir appliqué des algorithmes à 100 couples, nous avons sélectionné 5 couples, comme indiqué dans ce tableau (4.9).

Algorithmes	**Génétiques**		**Djikstra**		**A***	
N. Couples	**Arc 1*Arc2**	**TE (s)**	**Arc 1* Arc2**	**TE (s)**	**Arc 1* Arc2**	**TE (s)**
Couple 1	20	0.848	20	0,046	20	0،081
Couple 3	10	0,779	10	0,007	10	0,010
Couple 7	20	1،603	20	0،085	20	0،105
Couple 10	10	0،805	10	0،039	10	0،027
Couple 11	20	0،861	20	0،095	20	0،084
AVG	16	0.9792	**16**	0.0544	16	0.0614

Tableau 4.9. Comparaison des 3 algorithmes sur les couples choisis en utilisant la méthode (A*B) BDD4.

Le tableau (4.9) compare les performances de trois algorithmes de recherche de chemin : Génétique, Dijkstra et A*, sur une sélection de couples choisis aléatoirement à partir de la base de données BDD4. Et avons utilisé la méthode A * B, où A représenté Arc1(mouvement) et B représente Arc2(score). Notre objectif est de trouver un chemin entre la source et la destination où (Arc 1 * Arc 2) est un maximum le long du chemin.

La figure (4.13) représente un graphique comparant les Temps d'exécution entre les trois algorithmes : Dijkstra, A* et Génétiques, utilisé la méthode (A*B).

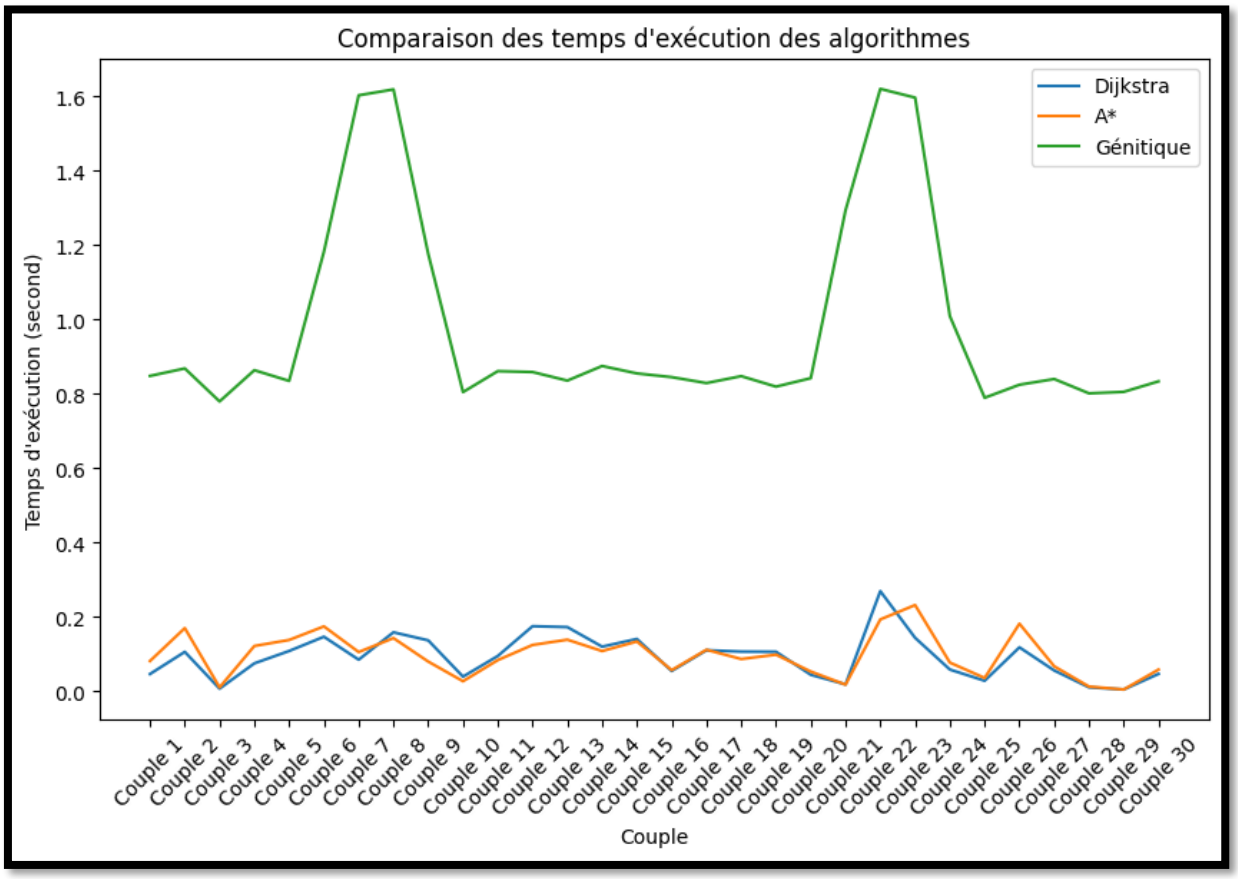


Figure 4.13. Temps d'exécution par couple pour chaque algorithme (30 premier couples) en utilisant la méthode (A*B) BDD4.

Le graphique montre une comparaison des temps d'exécution de trois algorithmes (Génétique, A*, Dijkstra) sur les 30 premiers couples. Les résultats indiquent que :

- Dijkstra est le plus rapide et le plus constant, parfait pour les applications nécessitant des performances prévisibles et rapides.
- A* offre un bon compromis entre rapidité et complexité, idéal pour des scénarios où des heuristiques peuvent améliorer l'efficacité.
- Génétique est beaucoup plus lent mais pourrait être utile pour des problèmes complexes où des solutions de qualité supérieure sont requises, malgré les temps d'exécution plus longs.

En résumé, l'algorithme Djikstra est le plus rapide, suivi de près par A*. L'algorithme génétique est le plus lent, avec des temps d'exécution significativement plus élevés.

La figure (4.14) représente un graphique comparant les (arc1 * arc2) entre les trois algorithmes : Dijkstra, A* et Génétiques, utilisé la méthode A * B.

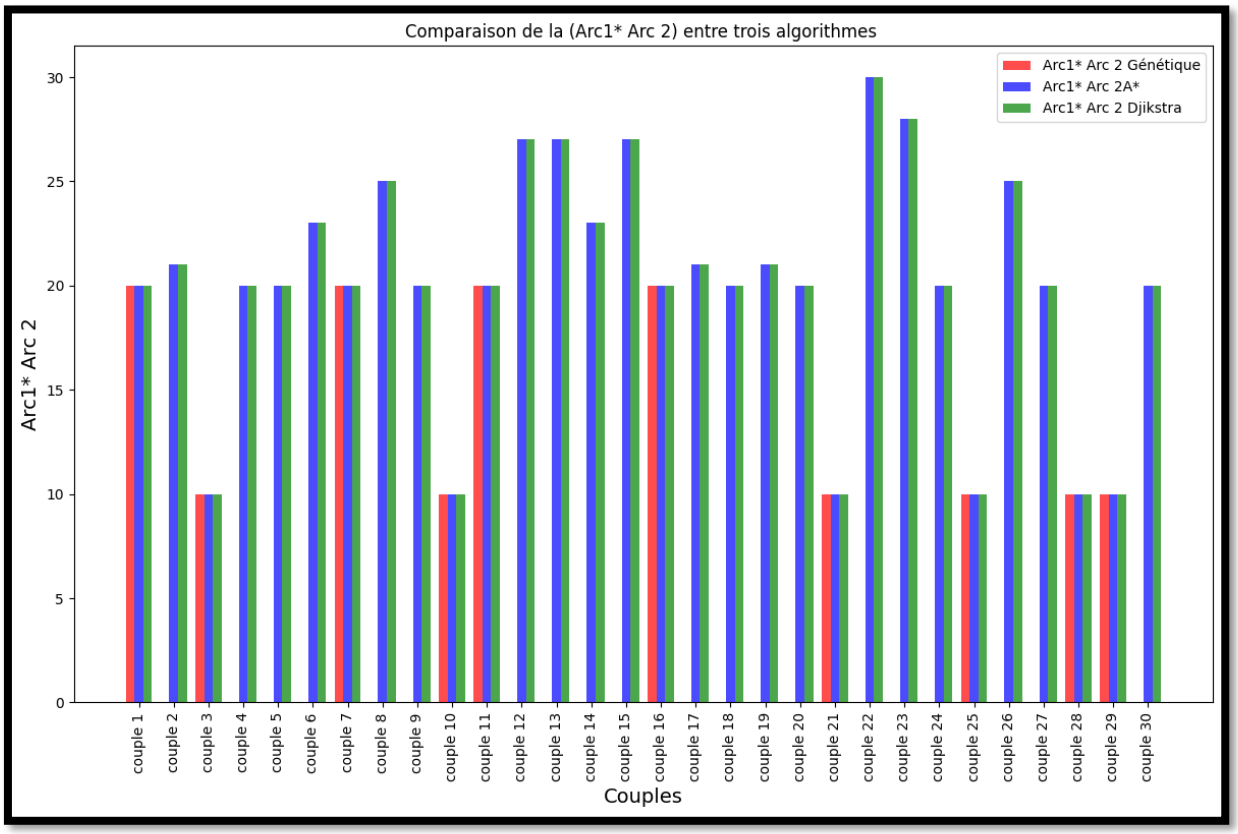


Figure 4.14. Comparaison de la (arc1*arc2) entre trois algorithmes BDD4.

Le graphique compare la multiplication des arcs (arc1 * arc2) à trois algorithmes (génétique, A *, Dijkstra) sur plusieurs paires.

- Algorithme Génétique : Les (arc1 * arc2) sont légèrement inférieures pour certains couples mais sont globalement similaires aux autres algorithmes.
- Algorithme A* : Montre des (arc1 * arc2) cohérentes et souvent égales à celles de Dijkstra, indiquant une similarité dans l'optimisation des chemins.
- Algorithme Dijkstra : Les (arc1 * arc2) sont en grande partie similaires à celles de l'algorithme A*, confirmant l'efficacité de ces deux algorithmes

En général, les algorithmes A* et Dijkstra montrent des performances presque identiques en termes de (arc1 * arc2) calculées, tandis que l'algorithme Génétique montre des variations mineures.

3) La méthode Ax+By

Après avoir appliqué des algorithmes à 100 couples, nous avons sélectionné 5 couples, comme indiqué dans ce tableau (4.10).

Définir des coefficients pour la fonction linéaire : (x * Arc1+ y *Arc2)

- x = 1 , y = 5 , où le score est plus important que le mouvement.
- A : Arc1 (mouvement).
- B : Arc2 (score).

Algorithmes	Génétiques		Djikstra		A*	
N. Couples	Ax +By	TE (s)	Ax+By	TE (s)	Ax+By	TE (s)
Couple 1	102	1,594	102	0,046	102	0,077
Couple 2	51	0,819	109	0,104	109	0,166
Couple 3	102	0,828	51	0,007	51	0,012
Couple 4	51	0,782	102	0,075	102	0,129
Couple 5	102	0,823	103	0,090	103	0,154
AVG	81.60	0.9692	**93.40**	0.0644	93.40	0.1076

Tableau 4.10. Comparaison des 3 algorithmes sur les couples choisis en utilisant la méthode (Ax + By) BDD4.

Le tableau (4.10) compare les performances de trois algorithmes de recherche de chemin : Génétique, Dijkstra et A*, sur une sélection de couples choisis aléatoirement à partir de la base de données BDD4. Et avons utilisé la méthode Ax+By, où A représenté 1 et B représente 5. Notre objectif est de trouver un chemin entre une source et une destination ou la somme (A*Arc 1+B*Arc 2) est maximale le long du chemin.

La figure (4.15) représente un graphique comparant les Temps d'exécution entre les trois algorithmes : Dijkstra, A* et Génétiques, en utilisant la méthode (Ax + By).

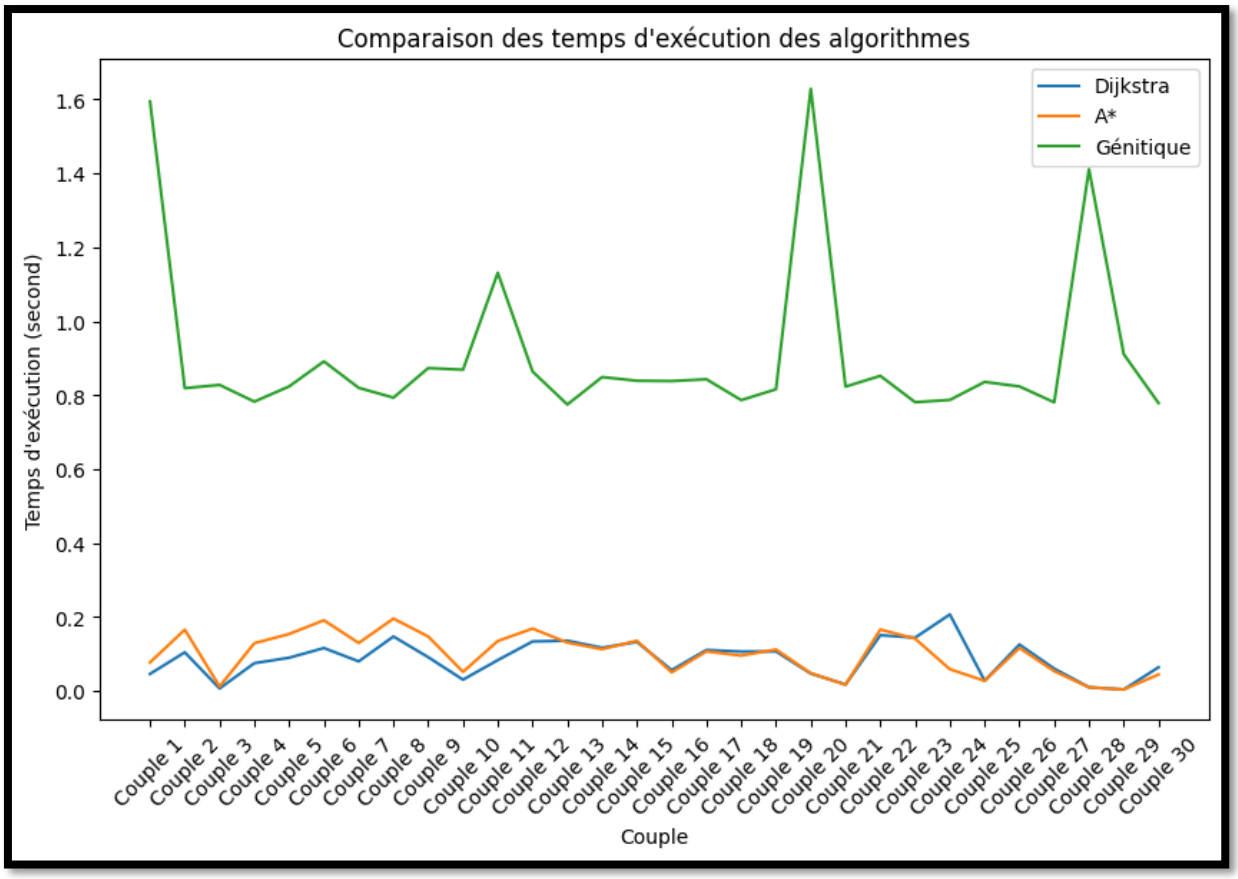


Figure 4.15. Temps d'exécution par couple pour chaque algorithme (30 premiers couples) en utilisant la méthode (Ax + By) BDD4.

Le graphique montre une comparaison des temps d'exécution de trois algorithmes (Génétique, A*, Dijkstra) sur les 30 premiers couples. Les résultats indiquent que l'algorithme Djikstra est le plus rapide, suivi de près par A*. L'algorithme génétique est le plus lent, avec des temps d'exécution significativement plus élevés.

La figure (4.16) représente un graphique comparant les résultats entre les trois algorithmes : Dijkstra, A* et Génétiques, en utilisant la méthode (Ax + By).

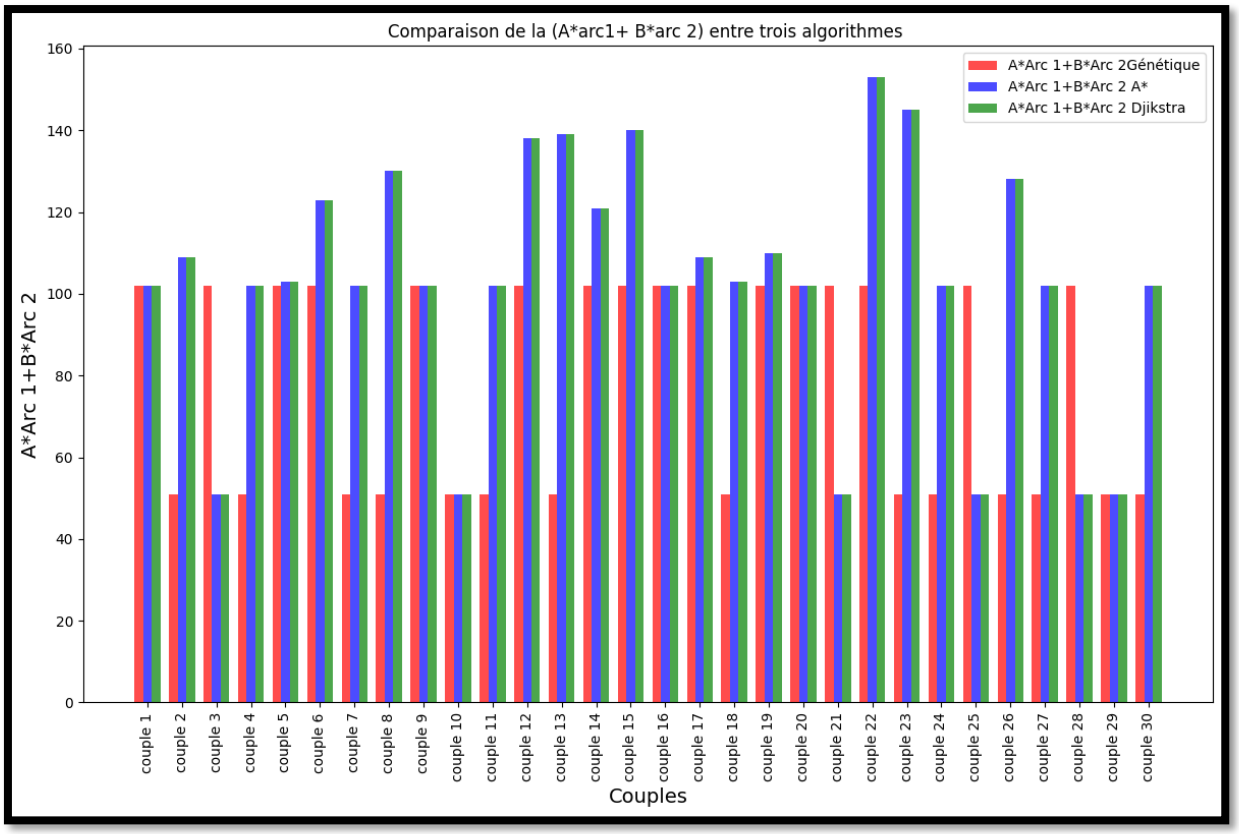


Figure 4.16. Comparaison des résultats (Ax + By) entre trois algorithmes BDD4.

Le graphique compare le total (Ax + By).de trois algorithmes (Génétique, A*, Dijkstra) sur de nombreux couples. En général, les algorithmes A * et Dijkstra montrent des performances presque identiques en matière de résultat calculé, tandis que l'algorithme génétique montre de légères différences.

4.5. Tableaux de comparaison des résultats pour les bases de données

Ce tableau (4.11) compare les performances de différents algorithmes (Génétiques, Djikstra, A*) sur différentes bases de données (BDD1, BDD2, BDD3, BDD4) en termes de temps d'exécution moyen (en secondes) et le AVG :

Algorithme / Base de données		Génétiques		Djikstra		A*	
		TE (s)	AVG	TE (s)	AVG	TE (s)	AVG
BDD1		212.256	11.5	15.592	12.45	10.148	12.45
BDD2		0.308	47.783	0.0035	63.404	0.0169	63.404
BDD3	A+B	0.568	366.79	0.554	234.27	0.014	356.2
	A*B	0.573	6603.83	0.469	1196.38	0.022	6204.71
	Ax + By	0.604	922.29	0.030	879.46	0.557	579.86
BDD4	A+B	1،047	16،5	0.116	22.53	0.0996	22.53
	A*B	0،944	13،793	0،086	19،78	0،066	19،78
	Ax + By	0.971	79.05	0.097	101.69	0.090	101.69

Tableau 4.11. Tableaux de comparaison des résultats pour les bases de données.

Le tableau (4.11) compare les résultats des algorithmes (génétique, Dijkstra, A *) appliqués à plusieurs bases de données (BDD1, BDD2, BDD3, BDD4). Pour chaque algorithme et base de données, il y a deux mesures : « Temps d'exécution » et « AVG ». Voici une analyse détaillée et une comparaison :

- Pour BDD1, l'algorithme A * est le plus rapide en matière de temps d'exécution. Les performances de Dijkstra sont également bonnes, tandis que les algorithmes génétiques sont significativement plus lents, Les valeurs AVG sont similaires pour les algorithmes À * et Dijkstra et l'algorithme génétique sont un peu moins qu'eux.

- Pour BDD2, Dijkstra est extrêmement rapide, suivi des algorithmes A*, L'algorithme génétique est plus lent, Les valeurs AVG similaire aux algorithmes A* et Dijkstra sont supérieures à la valeur de l'algorithme génétique.
- Pour la BDD3, les algorithmes A* sont généralement les plus rapides pour les trois types de calculs ((A+B), (A*B) et (A*x+B*y)). Dijkstra est rapide pour (A*x +B*y), mais moins performant pour(A+B) et(A*B). Les algorithmes génétiques ont un temps d'exécution élevé. Alors que les algorithmes génétiques et les algorithmes A* ont une valeur AVG élevée, l'algorithme de Dijkstra a une AVG inférieure à celle des autres algorithmes.
- Pour BDD4, les algorithmes A* et Djikstra, le temps d'exécution est plus faible, tandis que pour l'algorithme génétique il est légèrement plus élevé. La AVG était élevée dans A * x + B * y.

4.6. Complexité

Depuis la fin des années 90, la « complexité » est devenue un calcul nécessaire pour prouver la qualité et la rapidité d'un algorithme _(Maguire S. a., 2006). Le but de la complexité est de permettre une comparaison directe entre deux ou plusieurs algorithmes selon le temps d'exécution ou plus précisément selon le nombre d'instructions de chaque algorithme. Dans ce qui suit, nous allons calculer la complexité de chacun des algorithmes utilisés dans notre recherche.

Algorithme	GA	A*	Dijkstra
Complexité	O (n * Log(n))	O(n2)	O ((a + n) Log n)

Tableau 4.12. Complexité de trois algorithmes.

Nous remarquons que bien que les algorithmes génétiques puissent trouver des solutions de haute qualité, ils ont une complexité élevée et sont moins pratiques pour les applications nécessitant des résultats rapides. Dijkstra est très rapide et efficace pour trouver des chemins optimaux, tandis que A* offre un bon compromis entre rapidité et qualité de solution grâce à l'utilisation d'une heuristique. Le choix de l'algorithme dépend donc des exigences spécifiques de l'application, notamment en termes de rapidité d'exécution et de qualité des solutions obtenues.

4.7. Différentes versions des BDDs utilisées

Après une analyse approfondie, nous avons résumé nos résultats ainsi que les couples et les bases de données dans quatre versions appropriées : (xml, csv, txt, Excel (xls)).

Cela permet une intégration et une manipulation faciles pour diverses applications et analyses futures. Par exemple :

La figure (4.17) représente le fichier Excel de la quatrième base de données (BDD4.xls).

	B	C	D	E
1	**Source**	**Destination**	**arc 1**	**arc 2**
2	1	2132	1	10
3	2132	713	1	10
4	713	1022	1	10
5	1022	2319	1	10
6	2319	1102	1	10
7	1102	352	1	10
8	352	2264	1	10
9	2264	2023	1	10
10	2023	402	1	10

Figure 4.17. Capture de la BDD 4.xls.

La figure (4.18) représente le fichier texte pour les Couple des Nœuds de BDD2.

Source	Destination
9920	9934
12045	12047
5777	5778
9975	9960
5539	5536
11842	12409
8269	10769
10534	391
11208	11206
6156	10532

Figure 4.18. Couple des Nœuds de BDD2(couple_noeuds.txt).

La figure (4.19) représente le fichier csv les résultats des algorithmes génétiques utilisant la méthode A+B pour bdd3.

	A	B	C	D	E	F	G
1	id	source	destination	sum_distance_time	path_length	path	execution_time
2	0	271	682	433	3	271 -> 494 -> 682	0.9350955486297607
3	1	710	823	257	2	710 -> 823	0.8059124946594238
4	2	807	379	468	3	807 -> 910 -> 379	0.5995216369628906
5	3	190	454	549	3	190 -> 130 -> 454	0.4938726425170898
6	4	436	283	263	3	436 -> 791 -> 283	0.4984486103057861
7	5	469	11	485	3	469 -> 591 -> 11	0.4843549728393555
8	6	858	457	371	4	858 -> 323 -> 240 -> 457	0.4911086559295654
9	7	265	983	405	3	265 -> 941 -> 983	0.5100736618041992

Figure 4.19. results_genetic_algorithm (A+B).csv.

La figure (4.20) représente le fichier xml pour les Couple des Nœuds de BDD3

```
<Root><Row><Source>271</Source><Destination>682</Destination>
</Row><Row><Source>710</Source><Destination>823</Destination>
</Row><Row><Source>807</Source><Destination>379</Destination>
</Row><Row><Source>190</Source><Destination>454</Destination>
</Row><Row><Source>436</Source><Destination>283</Destination>
</Row><Row><Source>469</Source><Destination>11</Destination>
</Row><Row><Source>858</Source><Destination>457</Destination>
</Row><Row><Source>265</Source><Destination>983</Destination>
</Row><Row><Source>302</Source><Destination>288</Destination>
```

Figure 4.20. Couple des Nœuds de BDD3(couple_noeuds.xml).

4.8. Résultat et discussion

Pour Les bases de données BDD1 et BDD2 ont pour objectif d'optimiser la recherche du plus court chemin dans un graphe orienté uni-critère. Les résultats indiquent que l'algorithme génétique ne fournit pas les meilleurs résultats. Cela peut être attribué au fait que le graphe n'est pas complet, ce qui signifie que des informations cruciales peuvent manquer pour guider l'algorithme vers une solution optimale. Dans ce contexte, En termes de temps d'exécution, l'algorithme Dijkstra est généralement très performant pour des tâches simples ou de petites bases de données, comme le montre BDD2. Pour des tâches plus complexes, les algorithmes A* se révèlent très rapides, en particulier avec BDD1.

Les algorithmes de Dijkstra et A* ont affiché des performances similaires. En ce qui concerne les valeurs moyennes (AVG), les plus faibles ont souvent été observées avec Dijkstra et A*, ce qui indique une meilleure performance moyenne pour ces algorithmes.

En revanche, dans BDD3, où l'optimisation cherche à minimiser les critères dans un graphe complet non orienté, la fonction (A+ B) apparaît avec les temps d'exécution moyens (AVG)

Les plus bas dans les deux cas. Cette observation indique que pour cette configuration de graphe et de critères, l'approche par somme de critères est plus appropriée pour atteindre l'objectif d'optimisation minimum. Ensuite, vient la méthode (Ax + By), puis (A* B) dans les algorithmes Dijkstra et A*.

Dans une base de données BDD4, où l'optimisation vise à maximiser les paramètres dans un graphe orienté, la fonction (Ax + By) semble être la plus efficace en termes de (AVG). Les algorithmes (Dijkstra et A*) ont donné le même résultat (AVG). Ce résultat indique qu'une approche linéaire telle que l'ajout de critères est plus efficace pour atteindre l'objectif d'optimisation maximal dans cette configuration spécifique.

En conclusion, pour les bases de données unidimensionnelles, les algorithmes de Dijkstra et A* sont préférables à l'algorithme génétique en raison de la nature incomplète du graphe. Pour les bases de données multidimensionnelles, le choix de la fonction dépendra des caractéristiques spécifiques du graphe et des critères d'optimisation.

4.9. Conclusion

Le chapitre 4 présenté se concentre sur l'implémentation et la comparaison de trois algorithmes d'optimisation, à savoir l'algorithme de Dijkstra, l'algorithme génétique et l'algorithme A*. L'objectif principal était d'analyser, d'évaluer et de comparer les performances de ces algorithmes sur quatre bases de données différentes, tant uni-critères que multicritères.

Les paramètres de l'algorithme génétique sont détaillés, notamment la taille de la population, le nombre d'itérations et les différents opérateurs génétiques appliqués pour générer un nouvel ensemble de solutions.

Nous avons respectivement étudié les performances des algorithmes sur des bases de données uni-critères et multicritères. Pour les bases de données uni-critères, les performances d'exécution ont été comparées, montrant que les algorithmes de A* et de Dijkstra étaient globalement les plus rapides, tandis que l'algorithme génétique était nettement plus lent.

Pour les bases de données multicritères, les résultats ont été analysés selon trois méthodes différentes montrant que les algorithmes de A* et de Dijkstra étaient globalement les plus rapides, tandis que l'algorithme génétique était nettement plus lent., comme décrit dans des travaux antérieurs. Enfin, une conclusion générale du chapitre est présentée, basée sur les résultats et les analyses précédentes.

Ce chapitre offre ainsi une compréhension approfondie des performances relatives de ces algorithmes d'optimisation dans différentes situations, ce qui peut être utile pour guider la sélection d'algorithmes dans divers contextes d'application.

Conclusion Générale

Conclusion Générale

L'étude a porté sur l'implémentation et la comparaison de trois algorithmes d'optimisation : l'algorithme de Dijkstra, l'algorithme génétique et l'algorithme A*. L'objectif principal était d'analyser, d'évaluer et de comparer les performances de ces algorithmes sur quatre bases de données différentes, couvrant des cas uni-critères et multicritères, afin de fournir des informations précieuses sur leur efficacité et leurs applications possibles.

Les résultats montrent que les algorithmes de Dijkstra et A* sont particulièrement performants en termes de rapidité dans des bases de données uni-critères et compétitifs dans des contextes multicritères, surpassant souvent l'algorithme génétique en termes de vitesse d'exécution. Cependant, l'algorithme génétique, bien que plus lent, se distingue dans des scénarios nécessitant une exploration approfondie de l'espace des solutions. Pour les bases de données multicritères, la méthode par somme de critères (A+B) a souvent été la plus efficace pour minimiser les critères dans des graphes non orientés, tandis que l'algorithme A* et la méthode (Ax + By) ont démontré des performances supérieures dans des graphes orientés. Ainsi, pour des tâches nécessitant des temps d'exécution rapides et une efficacité éprouvée, les algorithmes de Dijkstra et A* sont recommandés, alors que l'algorithme génétique est plus approprié pour des problèmes nécessitant une exploration extensive. Ces insights fournissent un guide précieux pour la sélection de l'algorithme approprié selon les spécificités des problèmes, influençant des domaines tels que l'optimisation des réseaux de transport, la gestion de l'énergie, et la conception de structures. Les recherches futures pourraient se concentrer sur l'intégration de ces algorithmes pour tirer parti de leurs forces respectives et explorer leur application à des problèmes encore plus complexes et diversifiés. En conclusion, cette étude met en évidence l'importance de l'optimisation algorithmique, démontrant que le choix de l'algorithme adéquat est crucial pour atteindre des performances optimales et résoudre efficacement les problèmes complexes du monde réel.

Les références

Bibliographie

Bibliographie

Anwar, G. A. (2024). Deep reinforcement learning for intelligent risk optimization of buildings under hazard. *Reliability Engineering & System Safety*, 110118.

Bedboudi, A. a. (2018). Amélioration de l'algorithme de chauve-souris par modification de règles d'évolution et introduction de mécanisme de croisement. *Synthèse: Revue des Sciences et de la Technologie, 36*, 1--8.

Bellman. (1958). On a routing problem. *Quarterly of applied mathematics, 16*(1), 87--90.

Berkani, L. a. (2020). Extraction optimisée de règles d'association.

Bernard, F. W. (1959). *Transitivit́e et connexit́e. Comptes rendus.*

Boisson, J.-C. (2008). *Modélisation et résolution par métaheuristiques coopératives: de l'atome à la séquence protéique.* Doctoral dissertation, Université Lille 1.

Cerrone, C. a. (2017). Carousel greedy: A generalized greedy algorithm with applications in optimization. *Computers & Operations Research, 85*, 97--112.

Deneubourg, J. -L. (1990). The self-organizing exploratory pattern of the argentine ant. *Journal of insect behavior*.

F. Glover, M. L. (1986). tabu search. *Book in Informs Journal on Computing., 29*.

Fadel, A. (2019). *Etude comparative des algorithmes évolutionnaires dédier à l'optimisation multi-objectif.*

Finkelstein, A. (2020). *Recherche de plus court chemin multimodal de point à point dépendant du temps.* Doctoral dissertation, Université Côte d'Azur.

Grabener, T. (2010). *Calcul d'itinéraire multimodal et multiobjectif en milieu urbain.* Doctoral dissertation, Université des Sciences Sociales-Toulouse I.

Gupta, A. a. (2023). Determining chess piece values using machine learning. *Journal of Student Research, 12*(1).

H. Yusnita, M. Z. (2020). Performance of A Star with Dynamic Programming Algorithms in.

Hamed, A. Y. (2010). A genetic algorithm for finding the k shortest paths in a network. *Egyptian Informatics Journal, 11*(2), 75--79.

HAYI, Y. M. (2023). *Vers une gestion intelligente des servicessensibles au contexte : Approche centr´ee utilisateur.* thèse de doctorat.

Hussain, K. a. (2019). Metaheuristic research: a comprehensive survey. *Artificial intelligence review, 52*, 2191--2233.

Jin, I. J. (2024). Operating condition optimization of liquid metal heat pipe using deep learning based genetic algorithm: Heat transfer performance. *Nuclear Engineering and Technology*.

Kacem, F. (2012). *Algorithmes Exacts et Approchés pour des problèmes d'Ordonnancement et de Placement.* Doctoral dissertation, Ph. D. thesis.

Ketfi, A. C. (2016). *Sur certaines méthodes d'optimisation globale basées sur l'introduction de fonctions auxiliaires.*

Khetatba, M. a. (2018). Un algorithme hybride pour la résolution du problème de partitionnement matériel/logiciel. *Synthèse: Revue des Sciences et de la Technologie, 37*, 01--13.

Khishe, M. a. (2020). Chimp optimization algorithm. *Expert systems with applications, 149*, 113338.

Kirkpatrick, S. (1983). Optimization by simulated annealing. *science, 220*(4598), 671--680.

Lebbah, F. Z. (2015). *Programmation Par Contraintes Appliquée À Des Problèmes Issus Des Finances.* Laboratoire d'informatique et des technologies de l'information d'Oran (LITIO).

Lv, J. a. (2024). Optimization design of forebody shape of an underwater axisymmetric body for transition delay using a genetic algorithm. *Ocean Engineering, 301*, 117529.

Lyu, C. a. (2024). Inner-outer layer co-optimization of sizing and energy management for renewable energy microgrid with storage. *Applied Energy, 363*, 123066.

Maguire, S. a. (2006). 1.5 complexity science and organization studies. *Sage handbook of organization studies*, 165--214.

Maguire, S. a. (2006). 1.5 complexity science and organization studies. *Sage handbook of organization studies*.

Mobeen, A. a. (2023). Dynamic programming for shortest path problem in a multimodal transportation network comprising intermediate sinks. *AIP Conference Proceedings*, *2485.*

Nasr, M. a. (2020). Benchmarking meta-heuristic optimization. *arXiv preprint arXiv:2007.13476*.

Othmani, I. (1998). *Optimisation multicritère: fondements et concepts.*

Pape, U. (1974). Implementation and efficiency of Moore-algorithms for the shortest route problem. *Mathematical programming, 7*, 212--222.

Pieume, C. O. (2011). *Les techniques d'optimisation multicritère en.*

Polyak, B. T. (2007). Newton's method and its use in optimization. *European Journal of Operational Research, 181*(3), 1086--1096.

Rahmani Hosseinabadi, A. A. (2019). Extended genetic algorithm for solving open-shop scheduling problem. *Soft computing, 23*, 5099--5116.

Scarselli, F., Yong, S. L., Gori, M. (2005). Graph neural networks for ranking web pages. *IEEE*, 666-672.

Scarselli, F., Yong, S. L., Gori, M., Hagenbuchner, M., Tsoi, A. C., & Maggini, M. (September ,2005). Graph neural networks for ranking web pages. *IEEE/WIC/ACM International Conference on Web Intelligence*, 666-672.

Shadmani, A. a. (2024). Advancements in optimizing wave energy converter geometry utilizing metaheuristic algorithms. *Renewable and Sustainable Energy Reviews, 197*, 114398.

Singh, Y. a. (2018). Towards use of Dijkstra algorithm for optimal navigation of an unmanned surface vehicle in a real-time marine environment with results from artificial potential field.

Smairi, N. (2013). *Optimisation par essaim particulaire: adaptation de tribes à l'optimisation multiobjectif* .

Wang, P. a. (2024). A global optimization method for data center air conditioning water systems based on predictive optimization control. *Energy, 295*, 130925.

Xing, L. a. (2019). *Revised floyd-warshall algorithm to find optimal path in similarity-weight network and its application in the analysis of global value chain.*

Zhang, W. a. (2023). Multiobjective-based decision-making for the optimization of an urban passenger traffic system structure. *Sustainability, 15*(18), 13644.