REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie
Département d'informatique

N° d'ordre : IA07/M2/2022

Mémoire

Présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : **Intelligence Artificielle (IA)**

Une approche basée sur l'optimisation des colonies de fourmis pour la recherche binaire

---

Par :

AOUAS YASMINA

Soutenu le 28/06/2022 devant le jury composé de :

Merizig Abdelhak	M.A.B	Président
Berghida Meryem	M	Rapporteur
Belaala Abir	M	Examinateur

Année universitaire 2021-2022

Remerciement

الحمد والشكر لله

أما بعد

Je tiens à exprimer mes profonds remerciements A mon encadreur

Mme .Meryem Berghida, pour son encadrement, sa grande disponibilité,

sa confiance et pour le soutien qu'il a su m'accorder durant toute cette

année. Ses compétences scientifiques ont toujours été sources

d'enrichissement me permettant de mener à bien ce travail.

Merci pour tous vos conseils et votre patience.

Je voudrais aussi présenter mes remerciements à ma famille,

à mes amis et à tous ceux qui ont contribué de près ou de loin

à l'accomplissement de ce modeste travail.

Dédicace

Avec tout respect et amour je dédie ce modeste travail :

À deux belles perles et mes yeux alla et iyad

A ma mère et mon père, qui n'ont pas hésité de m'encourager et m'aider.

À mes frères, mes sœurs et à toute la famille.

À mes amies À mes collègues de travail dans la direction

des services universitaire Biskra

Résumé

La recherche est considérée comme une fonctionnalité importante dans un environnement informatique. Les techniques de recherche sont appliquées dans les récupérations de fichiers et d'indexation. Mais il existe diverses techniques de recherche, la recherche binaire est largement utilisée dans de nombreuses applications en raison de son avantage par rapport aux autres techniques de recherche. La recherche binaire est facile à mettre en œuvre et permet de trouver un élément dans un grand espace de recherche. Ce travail propose un algorithme ACOBS (Ant Colony Optimization for Binary Search) pour trouver un espace de recherche optimal pour la recherche binaire. L'algorithme ACOBS catégorise les espaces de recherche où l'élément clé est recherché uniquement dans une catégorie spécifique réduisant ainsi l'espace de recherche.

Mots clés:

- ✓ Optimisation par colonies de fourmis.
- ✓ Recherche binaire.
- ✓ Espace de recherche optimal.

Abstract

Search is considered to be an important functionality in a computational system. Search techniques are applied in file retrievals and indexing. Though there exists various search techniques, binary search is widely used in many applications due to its advantage over other search techniques namely linear and hash search. Binary search is easy to implement and is used to search for an element in a large search space. The worst case time complexity of binary search is O (log 2 n)

where n is the number of elements (search space) in the array. However, in binary search, searching is performed on the entire search space. The complexity of binary search may be further reduced if the search space is reduced. This paper proposes an Ant ColonyOptimization based Binary Search (ACOBS) algorithm to find an optimal search space for binary search. ACOBS algorithm categorizes the search space and the key element is searched only in a specific category where the key element can exist thereby reducing the search space.

Keywords:

- ✓ **Ant colony optimization.**
- ✓ **Binary search.**
- ✓ **Optimal search space.**

Table des matières

Table de matières …….. I

Liste des figures …… III

Liste des tableaux …… IV

Liste des Algorithmes ……. V

Introduction générale ……. 1

Chapitre 01 : L'optimisation combinatoire

1.1. Introduction ……. 3
1-2-Notions de base ……..... 3
1-2-1-Le voisinage…… 3
1-2.2 complexité ……... 3
1.2.3.1. Notion d'algorithme et Complexité ……..... 3
1.2.3.2 Complexité de problème……. 4
1.3. Les méthodes résolution des problemes d'optimisation……... 4
1.3.1. Les méthodes exactes ……. 5
1.3.2. Les heuristiques……..... 5
1.3.3 métaheuristique……... 5
1.4. Métaheuristique utilisée ……... 8
1.4.1. Intelligence Collectives des Fourmis …… 8
1.4.2. Optimisation par Colonies des Fourmis …… 10
1.5. Conclusion ……... 11

Chapitre 02 : La recherche binaire

2.1. Introduction ……. 12
2.2. La recherche dichotomique (binaire)……..... 12
2.3. L'algorithme de recherche binaire:……... 12
2.4. Complexité algorithmique de l'algorithme de recherche binaire ……... 13
3.5. Conclusion …….. 14

Chapitre 03 : Conception et expérimentations

3.1. Introduction ……. 15
3.2. Modèle global du système : ……. 15
3.3.Conception détaillée……..... 16
3.3.1. Recherche ACOBS :……. 16
3.3.2. Définition des variables :…….. 17
3.4 Implémentation et résultat expérimentaux …….. 18
3.4.1 L'environnement de développement :…… 18

3.4.2Langage de programmation :…….. 18
3.4.3Interface de l'application……. 19
3.4.4. Comparaison entre ACOBS et la recherche binaire pour rechercher des mots dans un fichier texte 20
3.5 .conclusion ……..... 21

Conclusion Générale …….. 22

Bibliographie ……..... VI

Liste des figures

Chapitre 01 : L'optimisation combinatoire

Fig 1.1. Classification des méthodes RPO 4
Fig1.2 colonie de fourmis 8

Chapitre 03: Conception et expérimentations

Fig. 3.1. Modèle du système 15
Fig. 3.2. Interface de RB et ACOBS 19
Fig. 3.3. Comparaison entre le nombre total de recherches utilisant ACOBS et RB 20

Liste des tableaux

Chapitre 02 : La recherche binaire

Tableau 2.1 Complexité temporelle de divers algorithmes de recherche 14

Chapitre 03 : Conception et expérimentations

Tableau 3.1. Résultats de recherche par RB 19
Tableau 3.2. Résultats de recherche par ACOBS 20
Tableau 3.3. Comparaison entre RB et ACOBS 20

Liste des Algorithmes

Chapitre 01 : L'optimisation combinatoire

L'algorithme de Colonies de fourmis 11

Chapitre 02 : La recherche binaire

Algorithme .Pseudo-code 13

Chapitre 03 : Conception et expérimentations

Pseudo code ACOBS 16

Introduction Générale

Introduction générale

Les méthodes classiques pour résoudre certains problèmes n'ont pas donné un résultat efficient, c'est pourquoi les informaticiens se sont orientés vers le monde naturel pour s'en inspirer.

En effet, Certains phénomènes naturels sont capables de mettre en œuvre des heuristiques originales capables de trouver des solutions à des problèmes difficiles à résoudre par des algorithmes classiques.

De plus ces heuristiques sont robustes. Parmi les approches s'inspirant des systèmes biologiques, on peut citer : les algorithmes génétiques, l'intelligence en essaim et les sociétés d'insectes telles que les fourmis qui ont démontré sur terrain des constatations extraordinaires.

Dans ce mémoire, nous allons proposer une amélioration d'une technique de recherche classique à savoir la recherche dichotomique ou la recherche binaire par un algorithme d'optimisation par colonies de fourmis.

Organisation du mémoire

Afin de bien présenter notre travail nous avons choisi la structure suivante :

Chapitre 1 : notions de base sur l'optimisation combinatoire

La première partie de notre travail est consacrée à l'exposé des différentes techniques d'optimisation capables de résoudre les problèmes d'optimisation combinatoire. Deux grandes classes de méthodes sont présentées : les méthodes exactes qui consistent généralement à énumérer, de manière implicite, l'ensemble des solutions de l'espace de recherche et garantissent de trouver une solution optimale, et les méthodes approchées qui traitent généralement des problèmes de grande taille efficacement.

Chapitre 2 : Ce chapitre sera consacré à la présentation de l'algorithme classique de recherche binaire RB. Nous comparons aussi la complexité temporelle de ce dernier avec les autres techniques de recherche

Chapitre 3 : Le dernier chapitre de ce travail présentera la conception globale et détaillée de l'algorithme proposé pour améliorer la recherche binaire. L'amélioration est effectuée en réduisant l'espace de recherche en utilisant l'algorithme de colonies de fourmis. Ce dernier vas permettre de trouver l'espace de recherche optimal.

Nous présentons aussi les résultats de l'approche proposée ainsi qu'une comparaison avec la RB classique.

Chapitre 01

L’optimisation combinatoire

1. INTRODUCTION

L'optimisation combinatoire est un outil indispensable combinant diverses techniques de la mathématique discrète et de l'informatique afin de résoudre des problèmes d'optimisation combinatoire de la vie réelle. Un problème d'optimisation combinatoire consiste à trouver la meilleure solution dans un ensemble discret de solutions appelé ensemble des solutions réalisables. En général, cet ensemble est fini mais de cardinalité très grande Il s'agit, en général, de maximiser (problème de maximisation) ou de minimiser (problème de minimisation) une fonction objectif sous certaines contraintes. Le but est de trouver une bonne solution dans un temps d'exécution raisonnable.

1-2-Notions de base

1-2-1-Le voisinage

Permettent de déterminer l'optimum d'une fonction avec ou sans contraintes. Soit S un ensemble de solutions à un problème d'optimisation et f une fonction objective qui mesure la valeur f(s)avec s ∈ S. Pour un problème de minimisation on cherche à déterminer une solution s qui minimise la fonction objectif. Un minimum est une solution qui fait partie des solutions réalisable, dans le domaine d'optimisation on distingue deux types de minimums :

1. **Minimum Local** : une solution s est minimum local par apport à une structure de voisinage N si ∀ s' ∈N(s), f(s)≤f (s')
2. **Minimum Global** : une solution s est minimum global si ∀s' ∈ S, f(s)≤f (s')
3. **Voisinage [3]** : le voisinage est une fonction notée N qui associe un sous Ensemble de S à toute solution s, les voisins de s sont s' ∈N(s).

Le voisinage représente l'ensemble de modifications (transformations) possibles appliquées à une solution pour obtenir des nouvelles solutions.

1-2.2. Complexité

1.2.3.1. Notion d'algorithme et Complexité

La première définition de la notion d'algorithme remonte au IXème siècle ap. J.-C., Al Khuwarizmi, un mathématicien perse, publie un ouvrage consacré aux algorithmes.

L'étymologie du terme « algorithme» vient du nom de ce mathématicien. Selon AlKuwarizmi Un algorithme est composé d'un nombre fini d'étapes, chaque étape est Composée d'un ou de plusieurs opérations [3].

Un algorithme est une méthode précise utilisable par ordinateur pour déterminer la solution d'un problème p, à condition que l'algorithme soit correct et qu'il converge.

Différents algorithmes ont des coûts différents en termes de :

- Temps d'exécution (nombre d'opérations effectuées par l'algorithme).
- Taille mémoire (Taille nécessaire pour stocker les différentes structures de données pour l'exécution).

Ces deux concepts sont appelés la complexité en temps et en espace de L'algorithme.

La complexité algorithmique est un concept fondamental pour tout informaticien, Elle permet de déterminer si un algorithme A est meilleur qu'un algorithme B et s'il est **Optimal** ou s'il ne doit pas être utilisé.

1.2.3.2. La complexité d'un problème

La complexité d'un problème est équivalente à la complexité du meilleur algorithme résolvant ce problème. Un problème est traitable (ou facile) s'il existe un algorithme polynomial pour le résoudre.

1-4 Les méthodes de résolution des problèmes d'optimisation

Les méthodes d'optimisation peuvent être réparties en deux grandes classes de méthodes pour la résolution des problèmes :

- ✓ Les méthodes exactes,
- ✓ Les méthodes approchées,

Les méthodes de résolution de problème d'optimisation elles sont schématisées comme suit :

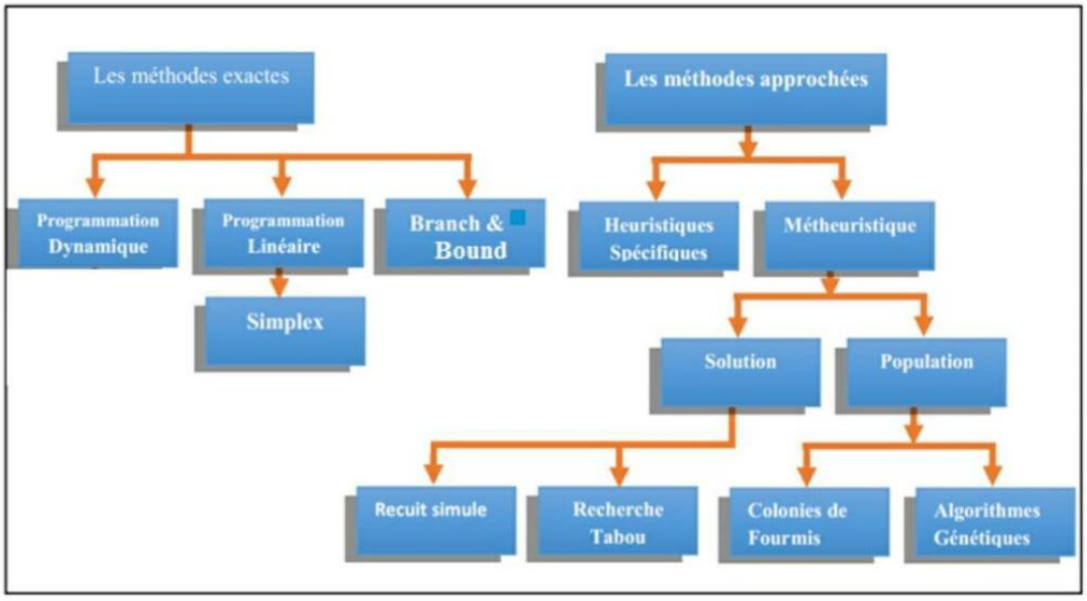

Fig 1.1. Classification des méthodes RPO

1.4.1. Les méthodes exactes

Les méthodes exactes obtiennent des solutions en garantissant leurs optimalités. Pour les problèmes NP-complet, les algorithmes exacts sont des algorithmes non polynômiaux (à moins que P = NP). Dans la classe des méthodes exactes, on peut trouver les algorithmes classiques suivantes : programmation dynamique, la famille des algorithmes de Branch and X (branch and bound,
branch and cut, branch and price), programmation par contraintes, et une famille des algorithmes A* de recherche(A*, IDA* Iterative Deepening Algorithms) développé dans la communauté de l'intelligence artiffcielle **[3].**

Ces méthodes énumératives peuvent être considérés comme des algorithmes de recherche utilisant un arbre. La recherche est effectuée sur l'ensemble de l'espace de recherche, et le problème est résolu en le subdivisant en problèmes plus simples. La programmation dynamique est basée sur la division récursive d'un problème en sous-problèmes plus simples. Cette procédure est basée sur le principe de Bellman qui dit que la sous-stratégie d'une politique optimale est elle-même optimale [4]. Cette méthode d'optimisation par stades est le résultat d'une séquence de décisions partielles. La procédure permet d'éviter une énumération totale de l'espace de recherche en élaguant des séquences partielles de décision qui ne peuvent pas conduire à la solution optimale.

L'algorithme de branch and bound et de A * sont basés sur une énumération implicite de toutes les solutions du problème d'optimisation considéré.

1.4.2. Les heuristiques

Les heuristiques permettent de trouver des solutions "bonnes" sur des instances du problème de grande taille. Elles permettent d'obtenir des performances acceptables à des coûts acceptables sur une grande gamme de problèmes. En général, les heuristiques n'ont pas une garantie de performance sur les solutions obtenues.

Elles peuvent être classées en deux familles : les heuristiques constructives sont principalement dédiés à un problème spécifique et essayent de construire une solution unique avec la meilleure qualité possible en choisissant soigneusement les éléments de solution prometteuses. Les heuristiques de recherche mettent en œuvre une recherche dans l'espace des solutions d'un problème donné pendant lequel ils examinent de nombreuses solutions différentes afin de trouver la meilleure solution possible.

1.4.3 Les métaheuristiques

Les métaheuristiques sont des algorithmes d'usage général qui peuvent être appliqués pour résoudre presque n'importe quel problème d'optimisation. Elles peuvent être considérées comme des méthodes générales de haut niveau qui peuvent être utilisées en tant que stratégie de guidage dans la conception des heuristiques pour résoudre des problèmes d'optimisation spécifiques.

Les métaheuristiques sont utilisées lorsque le problème est difficile et de grande taille, leur objectif est de trouver une solution proche de l'optimal rapidement.

A. Diversification, intensification et apprentissage

La diversification (ou exploration, synonyme utilisé presque indifféremment dans la littérature) désigne les processus visant à explorer différentes zones dans l'espace de recherche du problème à optimiser.

L'intensification (ou exploitation) vise à parcourir une zone de l'espace de recherche pour trouver la meilleure solution. La mémoire est le support de l'apprentissage, qui permet à l'algorithme de ne tenir compte que des zones où l'optimum global est susceptible de se trouver, évitant ainsi les optima locaux.

Les métaheuristiques progressent de façon itérative, en alternant des phases d'intensification, de diversification et d'apprentissage. L'état de départ est souvent choisi aléatoirement, l'algorithme se déroulant ensuite jusqu'à ce qu'un critère d'arrêt soit atteint. Les notions d'intensification et de diversification sont prépondérantes dans la conception des métaheuristiques, qui doivent atteindre un équilibre délicat entre ces deux dynamiques de recherches. Les deux notions ne sont donc pas contradictoires, mais complémentaires, et il existe de nombreuses stratégies les mêlant.

B. Classification des métaheuristiques

Plusieurs classifications des métaheuristiques ont été proposé, parmi ces classifications, nous pouvons citer :

B.1. Première classification : métaheuristiques à base de population ou à trajectoire

Les Métaheuristiques peuvent être classées en fonction du nombre de solutions utilisées en même temps. Les méthodes à trajectoire, sont des algorithmes basés sur une solution unique à n'importe quelle étape dans l'algorithme comme les algorithmes de recherche locales tels que TS (Tabu Search)[5], ILS (Iterated Local Search)[6] [7], VNS (Variable Neighborhood Search) [8], HC (Hill Climbing) [9], SA(Simulated Annealing)[10], GRASP (Greedy Randomized
Adaptive Search Procedure)[11][12]...etc.

Les algorithmes à base de population effectuent la recherche avec plusieurs points de départ (solutions initiales) dans un style parallèle comme par exemple : ACO (Ant Colony Optimization algorithm)[13], PSO (Particle Swarm Optimization) [12], les algorithmes évolutionnistes (Les algorithmes génétiques [15], la programmation génétique [16], les algorithmes mémétique [17],les algorithmes d'évolution di_érentielle [18], la recherche par dispersion
[19]...etc.).

B.2. Deuxième classification : Les métaheuristiques inspirées ou non inspirées d'un phénomène naturel

Les métaheuristiques inspirées d'un phénomène naturel peuvent se baser selon la source d'inspiration sur :

- L'intelligence en essaim comme : PSO, ACO, RFD (River Formation Dynamics)[20], ABC(Arti_cial Bee Colony) [21], BFO (Bacterial Foraging Optimization)[22], FA (Fire_y Algorithm) [24], CS(Cuckoo Search) [24], IWD (Intelligent Water Drops)[25], BA (Bat Algorithm) etc).
- Les systèmes biologiques comme : AG, ACO, CSA (Clonal Selection Algorithm)[26],...etc.
- la physique ou la chimie comme : Le recuit simulé (SA),HS(Harmony Search) [27], EO (Extremal Optimization)[28]...etc.
- Tandis que la méthode de descente (HC), ou la recherche Tabou (TS), vont dans la seconde classe.

B.3. Troisième classification : Les métaheuristiques avec fonction objectif statique ou dynamique
Les métaheuristiques peuvent également être classées en fonction de la manière d'utilisation de la fonction objectif. Bien que certains algorithmes maintiennent la fonction objectif donnée dans la représentation du problème telle qu'elle, d'autres, comme GLS (Guided Local Search) [29], la modifient lors de la recherche.

L'idée derrière cette approche est de s'échapper des minimas locaux en modifiant le paysage de la recherche. En conséquence, lors de la recherche, la fonction objectif est modifiée en essayant d'intégrer les informations recueillies au cours du processus de recherche.

B.4. Quatrième classification : Les métaheuristiques avec une ou plusieurs structures de voisinage
La plupart des métaheuristiques travaillent sur une structure de voisinage unique. D'autres métaheuristiques, comme la recherche à voisinage variable (VNS), utilisent un ensemble de structures de voisinage qui donne la possibilité de diversifier la recherche et d'explorer mieux l'espace de recherche.

B.5. Cinquième classification : Les métaheuristiques avec ou sans mémoire
Une caractéristique très importante pour classer les métaheuristiques est l'utilisation de l'historique de recherche. Les algorithmes qui n'utilisent pas la mémoire (historique) effectuent un processus de Markov, car l'information qu'ils utilisent pour déterminer la prochaine action est l'état actuel du processus de recherche. Il existe plusieurs façons pour utiliser la mémoire. Habituellement, nous distinguons entre l'utilisation de la mémoire à long terme et court terme. La première garde généralement la trace de mouvements effectués récemment, des solutions visitées ou, en général, les décisions prises. La seconde est généralement une accumulation de paramètres synthétiques sur la recherche.

1.5. Métaheuristique utilisée : l'optimisation par colonies de fourmis (ACO : Ant Colony Optimization)

1.5.1. Intelligence Collectives des Fourmis

Les fourmis offrent une grande diversité de comportements et de morphologies, l'étude précise de leurs comportements est souvent limitée aux espèces les moins populeuses pour des raisons pratiques. Cette diversité est une mine d'inspiration fascinante pour les systèmes informatiques, c'est ainsi que les capacités des fourmis en matière de coopération, de communication, de compétition et d'apprentissage peuvent être mise à profit pour la conception des algorithmes de résolution des problèmes.

Les principales caractéristiques des fourmis que l'on pourra retrouver dans les Systèmes informatiques sont :

A. Communication

Les fourmis ont développé des mécanismes de communication très élaborés, il à été définie plusieurs types de réponse mettant en œuvre une forme de communication :

- L'alarme.
- L'attraction simple.
- Le recrutement.
- L'échange d'aliments solides. …etc.

La communication chimique est la plus présente chez les fourmis. Les phéromones, sont des mélanges hydrocarbures, sont à la base de la communication de nombreuses espèces.

Les ouvrières sont capables de déposer des traces chimiques sur le trajet qu'elles empruntent pour ramener de la nourriture. Ils sont capables aussi de déclencher des alarmes quand le nid est attaqué et ainsi mobiliser un grand nombre d'individus pour défendre la fourmilière.

La communication entre les individus peut se faire directement ou indirectement, l'utilisation des phéromones est majoritairement considérée comme une forme indirecte puisque l'échange d'information se fait grâce au support du sol, ce principe est appelé principe de la **stigmergie**.

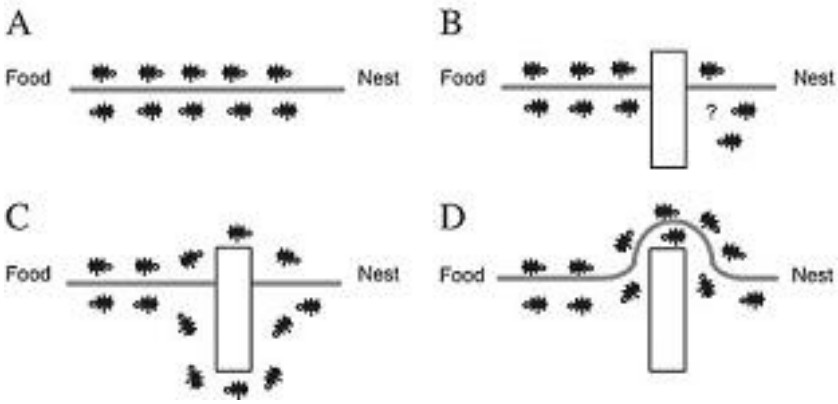

Figure 2 colonie de fourmis

B. Principe de la Stigmergie

Un principe fondamental des comportements émergents à travers des interactions locales est la stigmergie. Le biologiste Grassé est le premier qui fait introduire ce concept dans les années cinquante [32], devenue maintenant une voie de recherche et de conception dans les systèmes d'agents artificiel. Grassé fut découvrir le principe en étudiant les comportements des insectes sociaux. La stigmergie fournit le concept général permettant la coordination entre les individus et le comportement global au niveau de la colonie.

C. Phéromones

La phéromone est une substance chimique qui jeu un rôle très important dans la réalisation des tâches définies. Il permet de refléter une caractéristique des systèmes complexes. La présence de la phéromone dans une piste jeu le rôle d'un moyen permettant l'attraction des fourmis à la piste renforcée. La phéromone est soumise à l'évaporation, due aux contraintes d'environnement. Le rôle inverse est joué à travers le processus de propagation et de sécrétion de la phéromone.

L'évaporation permet de réduire les amplifications et de créer des points de fluctuations, pour lesquels des nouvelles situations prometteuses peuvent être visité. Ce mécanisme conduit à la découverte des nouvelles régions riches en nourriture.

D. Propagation

La propagation de la phéromone est un processus qui permet de laisser passer une quantité de la phéromone d'un emplacement à un autre emplacement voisin. Si le taux d'évaporation est de ½ par exemple, la moitié de la quantité de la phéromone de l'emplacement actuel est propagée vers les emplacements voisins.

E. Evaporation

L'évaporation est un processus, qui permet de réduire la quantité de la phéromone dans un emplacement par un taux défini. Cette évaporation est due aux contraintes de l'environnement.

F. Le Fourragement

La recherche de la nourriture, appelée aussi « le fourragement », est une activité souvent plus dispersée spatialement que la construction du nid et qui peut aussi être mise en œuvre de façon très différente suivant les espèces de fourmis. La communication peut avoir un impact important, en particulier pour les mécanismes de recrutement, dont le principal intérêt collectif est de rassembler les ouvrières sur les sources de nourriture rentables.

1.5.2. Optimisation par Colonies des Fourmis

La capacité des fourmis à trouver le plus court chemin entre une source de nourriture et leur nid, a été utilisée pour résoudre des problèmes d'optimisation combinatoire. Les traces de la phéromone,

représentent une attirance pour un arc du graphe modélisant le problème. Chaque fourmi construit une solution pour le problème et l'évaluation de chaque solution est utilisée pour mettre à jour les traces de la phéromone. Ces principes ont été appliqués en premier au problème de voyageur de commerce [33][34]. Après des variations de la méthode ont été proposées par[35][36] La même technique est appelée pour d'autres problèmes combinatoires comme l'affectation quadratique et autres problèmes [37][38].

A. Principe de l'algorithme ACO

La méthode ACO « Ant Colony Optimisation » est une métaheuristique inspirée du comportement des fourmis dans la recherche des nourritures. La Première version d'algorithme a été développée par Dorigo [39].

Lorsqu'une fourmi doit prendre de décision sur la direction à prendre, elle doit Choisir le chemin ayant la plus forte concentration en phéromone, c'est-à-dire la **d**écision dépend de la probabilité de transition d'un emplacement à une autre. Cette **p**robabilité dépend de la concentration en phéromone. C'est exactement le principe **u**tilisé par les algorithmes d'optimisation à base des fourmis. Chaque fourmi est considérée comme un agent capable de générer des **s**olutions

B. L'algorithme de Colonies de fourmis

L'Algorithme de colonies de fourmis

Début

Initialiser une population de m fourmis ;
Evaluer les m fourmis ;
Tant que la condition d'arrêt n'est pas satisfaite **faire**
Pour i=1 à m **faire**
Construire le trajet de la fourmi i;
Déposer des phéromones sur le trajet de la fourmi i;
Fin pour
Evaluer les m fourmis;
Evaporer les pistes de phéromones;
Fin Tant que
Retourner la ou les meilleures solutions ;

Fin

1.5. Conclusion

Dans ce chapitre, nous avons passé en revue quelques notions sur l'optimisation. Nous avons introduit quelques notions sur la complexité à savoir, la complexité des algorithmes et la complexité des problèmes. Nous avons aussi cité quelques méthodes de résolution de problèmes d'optimisation. Nous avons parlé sur les méthodes exactes, les heuristiques et les métaheuristiques.

Bibliographie

1) *, La programmation orientée objet, ÉDITIONS EYROLLES , Paris , 2009.*
2) *LEMOUARI Ali Introduction Aux Métaheuristiques 2014 université de Jijel BP 98 Ouled Aissa. 18000, JIJEL, ALGERIE.*
3) *Stuart Russell et Peter Norvig. Arti_cial intelligence - a modern approach. Prentice-Hall International Limited, London, UK, 1995.*
4) *Richard Bellman. Dynamic programming. Princeton University Press, Princeton, NJ, 1957.*
5) *Fred Glover et Manuel Laguna. Tabu Search. Kluwer Academic Publishers, Norwell, MA, USA, 1997.*
6) *Helena R Lourenço, Olivier C Martin et Thomas Stützle. Iterated local search. In Handbook of Metaheuristics, volume 57 of International Series in Operations Research and Management Science, pages 321_353. Kluwer Academic Publishers, 2002.*
7) *Helena R Lourenço, Olivier C Martin et Thomas Stützle. Iterated Local Search : Framework and Applications Handbook of Metaheuristics. volume 146 of International Series in Operations Research & Management Science, chapitre 12, pages 363397. Springer US, Boston, MA, 2010.*
8) *N Mladenovi¢ et P Hansen. Variable neighborhood search. Computers & Operations Research, vol. 24, no. 11, pages 10971100, 1997.*
9) *S J Russell et P Norvig. Articial Intelligence : A Modern Approach. Prentice Hall, 3rd édition, 2009.*
10) *S Kirkpatrick, C D Gelatt et M P Vecchi. Optimization by simulated annealing. Science, vol. 220, pages 671680, 1983.*
11) *Thomas A Feo et Mauricio G C Resende. A probabilistic heuristic for a computationally di-cult set covering problem. Operations Research Letters, vol. 8, no. 2, pages 6771, 1989.*
12) *Thomas A Feo et Mauricio G C Resende. Greedy Randomized Adaptive Search Procedures. Journal of Global Optimization, vol. 6, pages 109133, 1995.*
13) *Alberto Colorni, Marco Dorigo et Vittorio Maniezzo. Distributed Optimization by Ant Colonies. In European Conference on Arti cial Life, pages 134142, 1991.*
14) *James Kennedy et Russell C. Eberhart. Particle swarm optimization. In Proceedings of the IEEE International Conference on Neural Networks, pages 19421948, 1995.*
15) *Melanie Mitchell. An introduction to genetic algorithms. MIT Press, Cambridge, MA, USA, 1998.*
16) *Wolfgang Banzhaf, Frank D. Francone, Robert E. Keller et Peter Nordin. Genetic programming : An introduction : on the automatic evolution of computer programs and its applications. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.*
17) *Xianshun Chen, Yew-Soon Ong, Meng-Hiot Lim et Kay Chen Tan. A Multi-Facet Survey on Memetic Computation. IEEE Trans. Evolutionary Computation, vol. 15, no. 5, pages 591607, 2011.*
18) *Rainer Storn et Kenneth Price. Dierential Evolution &Ndash ; A Simple and E-cient Heuristic for Global Optimization over Continuous Spaces. J. of Global Optimization, vol. 11, no. 4, pages 341359, Décembre 1997.*

19) *Rafael Martí, Manuel Laguna et Fred Glover. Principles of scatter search. European Journal of Operational Research, vol. 169, no. 2, pages 359372, 2006.*

20) *Pablo Rabanal, Ismael Rodríguez et Fernando Rubio. Using River Formation Dynamics to Design Heuristic Algorithms. In Selim G. Akl, Cristian S. Calude, Michael J. Dinneen, Grzegorz Rozenberg et Todd Wareham, editeurs, UC, volume 4618 of Lecture Notes in Computer Science, pages 163177. Springer, 2007.*

21) *Duc Truong Pham, Marco Castellani et Le Thi Hoai An. Nature-Inspired Intelligent Optimisation Using the Bees Algorithm. T. Computational Collective Intelligence, vol. 13, pages 3869, 2014.*

22) *Swagatam Das, Arijit Biswas, Sambarta Dasgupta et Ajith Abraham. Bacterial Foraging Optimization Algorithm : Theoretical Foundations, Analysis, and Applications. In Ajith Abraham, Aboul-Ella*

23) *Hassanien, Patrick Siarry et Andries Engelbrecht, editeurs, Foundations of Computational Intelligence Volume 3 SE - 2, volume 203 of Studies in Computational Intelligence, pages 2355. Springer Berlin Heidelberg, 2009.*

24) *Xin-She Yang. Nature-inspired metaheuristic algorithms : Second edition. Luniver Press, 2010.*

25) *Hamed Shah ;Hosseini. The Intelligent Water Drops Algorithm a Nature Inspired Swarm ;Based Optimization Algorithm. Int. J. Bio-Inspired Comput., vol. 1, no. 1/2, pages 7179, Janvier 2009.*

26) *L. N. de Castro et F. J. Von Zuben. Learning and Optimization Using the Clonal Selection Principle. Trans. Evol. Comp, vol. 6, no. 3, pages 239251, Juin 2002.*

27) *Zong Woo Geem, Joong Hoon Kim et G V Loganathan. A new heuristic optimization algorithm : harmony search. Simulation, vol. 76, no. 2, pages 6068, 2001.*

28) *Stefan Boettcher et Allon G. Percus. Extremal Optimization : Methods derived from Co-Evolution. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela et Robert E. Smith, editeurs, Proceedings of the Genetic and Evolutionary Computation Conference, volume 1, pages 825832, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.*

29) *Chris Voudouris et Edward Tsang. Guided Local Search. Rapport technique, European Journal of Operational Research, 1995.*

30) *François Morain, « Introduction à la complexité des algorithmes : 7.3.2 Recherche dichotomique » [archive], sur École polytechnique (France), 2004.*

31) *https://edelalon.com/blog/2013/09/zipcode-to-city-state-excel-spreadsheet/10/06/2022*

32) *) P. Grassé. "La reconstruction du nid et les coordinations interindividuelles chez bellicositermes et cubitermes. La théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs". Insectes Sociaux , 6, pp.41–81.*

33) *A. Colorni, M. Dorigo, V. Maniezzo, "Distributed Optimization by Ant Colonies" . In Varela and Bourgine, pp. 134–142.*

34) *A. Colorni, M. Dorigo, V. Maniezzo, "Investigations of some properties of an "Ant algorithm"". In Männer and Manderick, pp. 509-520.*

35) *M. Dorigo, V. Maniezzo, and A. Colorni. "The ant system:*

Optimization by a colony of cooperating ants". IEEE, Transactions on Systems, Management and Cybernetics, 26, pp.1–13.

36) L. Gambardella, M. Dorigo. "Solving Symmetric and TSPs by Ant Colonies". Proceeding of the IEEE, Transactions on Evolutionary Computation, Japan.

37) L. M. Gambardella, E. Taillard, and M. Dorigo. "Ant Colonies for the QAP". Journal of Operational Research Societies, Vol. 50(2), pp.167-176.

38) V. Maniezzo , A. Colorni, "The Ant System Applied to the nQuadratic Assignment problem". In IEEE Transactions on Knowledge and Data Engineering. Vol. 11(5), pp.769-778.

39)J. Delgado and R. Sole. "Collective-induced computation". Physical Review , Vol. 55, pp.2338-2344.

Chapitre 02

La recherche binaire

2.1. Introduction :

Les techniques de recherche sont appliquées dans les récupérations de fichiers, l'indexation, etc. Bien qu'il existe diverses techniques de recherche, la recherche binaire est largement utilisée dans de nombreuses applications en raison de son avantage par rapport à d'autres techniques de recherche, à savoir la recherche linéaire et par hachage. La recherche binaire est facile à implémenter et utilisé pour rechercher un élément dans un grand espace de recherche. Le pire des cas la complexité temporelle de la recherche binaire est O (log 2 n) où n est le nombre d'éléments dans le tableau (espace de recherche). Cependant, en recherche binaire, la recherche est effectuée sur l'ensemble d'espace de recherche. La complexité de la recherche binaire peut être encore réduite si l'espace de recherche est réduit

2.2. La recherche dichotomique (binaire) :

La recherche dichotomique, ou recherche par dichotomie1 (en anglais : binary search), est un algorithme de recherche pour trouver la position d'un élément dans un tableau trié. Le principe est le suivant : comparer l'élément avec la valeur de la case au milieu du tableau ; si les valeurs sont égales, la tâche est accomplie, sinon on recommence dans la moitié du tableau pertinente.

Le nombre d'itérations de la procédure, c'est-à-dire le nombre de comparaisons, est logarithmique en la taille du tableau. Il y a de nombreuses structures spécialisées (comme les tables de hachage) qui peuvent être recherchées plus rapidement, mais la recherche dichotomique s'applique à plus de problème

2.3. L'algorithme de recherche binaire :

1. Trouver la position la plus centrale du tableau (si le tableau est vide, sortir).
2. Comparer la valeur de cette case à l'élément recherché :
   - Si la valeur est égale à l'élément, alors retourner la position,
   - Sinon reprendre la procédure dans la moitié de tableau pertinente.

On peut toujours se ramener à une moitié de tableau sur un tableau trié en ordre croissant. Si la valeur de la case est plus petite que l'élément, on continuera sur la moitié droite, c'est-à-dire sur la partie du tableau qui contient des nombres plus grands que la valeur de la case. Sinon, on continuera sur la moitié gauche.

Algorithme .Pseudo-code

```
1. VARIABLE
2. t : tableau d'entiers trié
3. mil : nombre entier
4. fin : nombre entier
5. deb : nombre entier
6. x : nombre entier // x : l'entier recherché
7. tr : booléen
8. DEBUT
9. tr ← FAUX
10. deb ← 1
11. fin ← longueur(t)
12. tant que tr == FAUX et que deb ⩽ fin :
13. mil ← partie_entière((deb+fin)/2)
14. si t[mil] == x :
15. tr = vrai
16. sinon :
17. si x > t[mil] :
18. deb ← mil+1
19. sinon :
20. fin ← mil-1
21. fin si
22. fin si
23. fin tant que
24. renvoyer la valeur de tr
25. FIN
```

2.4. Complexité algorithmique de l'algorithme de recherche binaire :

La dichotomie possède une complexité algorithmique logarithmique en le nombre d'éléments composant le tableau dans lequel s'effectue la recherche On considère dans un premier temps le nombre de comparaisons comme étant la mesure de complexité. On appelle T(n) le nombre de comparaisons effectuées pour une instance de taille n. Alors le nombre de comparaisons T

satisfait la récurrence suivante : T(n)=1+T(n/2). D'après le master théorème [30] cette récurrence a une solution de la forme T(n)=O(log(n)), avec la notation de Landau. Enfin le nombre de comparaisons est linéaire en nombre d'opérations effectuées ; l'algorithme a donc une complexité logarithmique.

Le tableau (**Tableau 2.1**) compare la complexité temporelle de différents algorithmes de recherche.

Algorithme de recherche[30]	**Taille d'entrée**	**Complexité temporelle**	
		Pire des cas	**Cas moyen**
Recherche exponentielle[30]	N	O(logn) -	O(logn)
Recherche de Fibonacci[30]	N	O(logn)	O(logn)
Recherche binaire omniprésente[30]	N	O(logn)+1	O(logn)
Recherche de hachage[30]	N	O(n)	O(1)
Recherche ternaire[30]	N	O(2log3n)	-
Recherche binaire[30]	N	O(log(n))	-

Tableau 1. Complexité temporelle de divers algorithmes de recherche

3.5. Conclusion :

La recherche binaire est préférée pour de nombreuses applications en raison de sa faible complexité de calcul. Cependant, l'algorithme fonctionne sur l'ensemble du tableau (espace de recherche) pour rechercher un élément clé.

Dans le chapitre suivant, nous allons proposer un algorithme amélioré de la recherche binaire nommé ACOBS (Ant Colony Optimization based Binary Search).

L'amélioration de cet algorithme a pour but de trouver l'espace de recherche optimal pour la recherche binaire.

Chapitre 03

Conception et expérimentations

1. Introduction :

Après avoir mis le point sur les concepts théoriques liés à ce travail, à savoir, la recherche dichotomique et l'algorithme de colonie de fourmis. Ce chapitre sera consacré à la conception de l'algorithme de recherche binaire amélioré par les algorithmes de colonies de fourmis nommé ACOBS (**A**nt **C**olony **O**ptimization algorithm for **B**inary **S**earch). Nous présentons ensuite nos résultats expérimentaux.

2. Modèle global du système :

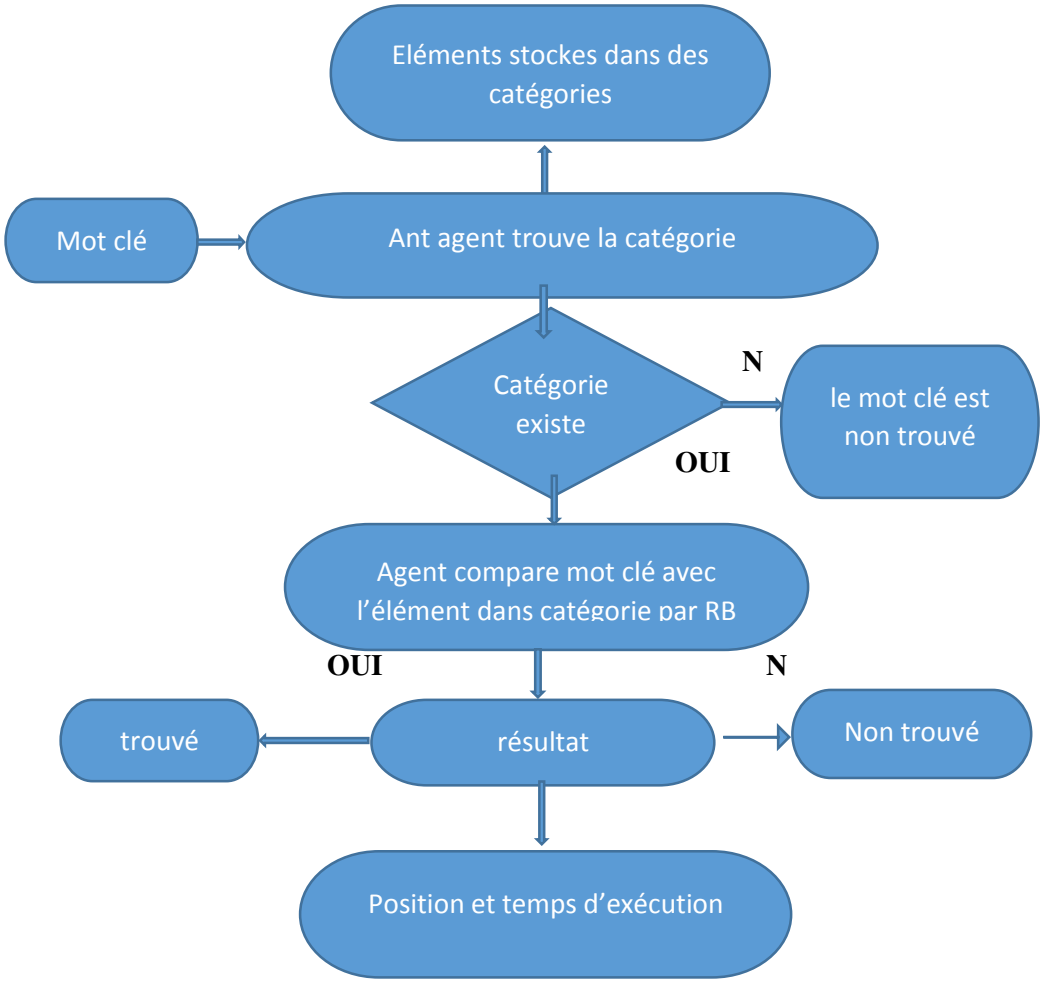

Fig. 3.1. Modèle du système

La figure 3.1 montre le schéma global de l'algorithme proposé (ACOBS). L'approche proposée est une amélioration de l'algorithme de recherche binaire. Le principe est de réduire en premier lieu l'espace de recherche par un algorithme de colonie de fourmis, puis appliquer la recherche binaire sur l'espace de recherche réduit, ce qui vas réduire la complexité temporelle de l'algorithme de recherche.

Dans cette proposition, l'ensemble des éléments triés par ordre croissant est catégorisé et l'agent fourmi trouve l'espace de recherche réduit en calculant le nombre et les dernières positions d'occurrence des éléments appartenant à chaque catégorie.

3.Conception détaillée

3.1. Recherche binaire :

Le principe de l'algorithme de recherche binaire a été détaillé dans le chapitre 2. Il sera utilisé une fois l'espace de recherche est réduit par l'algorithme de colonies de fourmis.

3.2. Recherche ACOBS :

1. Les éléments triés par ordre croissant sont classés en fonction de la nature des éléments. Par exemple :
   - Si les éléments sont des nombres et sont de longueur variable, la catégorisation est basée sur le nombre de chiffres.
   - Si les éléments sont des chaînes et sont de longueur variable, la catégorisation est basée sur le premier chiffre variable.
   - Si tous les éléments du tableau ont la même longueur, les éléments sont classés en fonction du premier chiffre variable.
2. L'agent fourmi stocke le nombre d'éléments appartenant à chaque catégorie et la position de la dernière occurrence des éléments de chaque catégorie dans sa liste mémoire.
3. Pour rechercher la valeur de la clé, l'agent fourmi trouve la catégorie de la clé à rechercher.
4. L'agent fourmi dépose la phéromone au début de la catégorie trouvée et prend la position de la dernière occurrence des éléments de cette catégorie ainsi que le nombre des éléments de cette catégorie à partir de la liste mémoire de la catégorie correspondante. Donc, l'espace de recherche réduit est trouvé.
5. Une recherche binaire est effectuée dans cet espace de recherche réduit. Si l'espace de recherche n'existe pas, la valeur de la clé n'est pas présente et le processus est arrêté.

Le pseudo code de l'approche proposée est le suivant :

Pseudo code ACOBS
Procedure ACOBS(ky , $x[n]$, c , p) # ky est la valeur clé à rechercher dans un tableau trié $C = \{c_1, c_2, c_3, \dots, c_k\}$ # c_i: représente le nombre d'éléments de la catégorie i $P = \{p_1, p_2, p_3, \dots, p_k\}$ # p_i : représente la position de la dernière occurrence des éléments de la catégorie i $lp = 0$; $hp = 0$; # $[lp, hp]$: dénote l'intervalle de l'espace de recherche réduit trouvé

$i = trouver\_Catégorie\ (Ky)$ # l'agent fourmi trouve la catégorie i de la clé Ky

$p_i = Deposer\_phéromone\ (i)$ # cette procédure permet de positionner l'agent fourmi au début de la catégorie i

Si $(p_i \neq 0)$ **alors**

$hp = p_i$

$lp = p_i - c_i + 1;$

Tantque $(lp \leq hp)$ **faire**

$val\_moy = int(lp + hp/2)$;

$phéromone = val\_moy$;

Si $(Ky = X[val\_moy])$**alors**

$énergie(agent\_fourmi) = +1$

$hp = val\_moy\ - 1$

Sinon Si $(Ky\ \geq X[val\_moy])$ **alors**

$énergie(agent\_fourmi) = -1$

$lp = val\_moy + 1$

Sinon Si $(Ky = X[val\_moy])$ **alors**

$énergie(agent\_fourmi) = 0$

FinSi

Si $(énergie(agent\_fourmi) = 0\ )$ **alors**

Retourner (val_moy)

Sinon

Retourner ("$élément\ non\ trouvé$! ")

FinSi

FinTantque

FinSi

Fin ACOBS()

3.3. Définition des variables :

L'agent fourmi a une liste « mémoire » $\{C, P\}$ qui contient les catégories du fichier en entrée (la catégorisation a été expliquée dans la section 3.1).

Pour chaque catégorie i, on a deux variables :

c_i : qui représente le nombre des éléments de cette catégorie et

p_i : qui représente la position de la dernière occurrence des éléments de cette catégorie.

L'ensemble C contient le nombre d'éléments de chaque catégorie :

$$C = \{c_1, c_2, c_3, \dots, c_n\}$$

L'agent fourmi indique la position de la dernière occurrence des éléments de la catégorie i selon l'équation suivante :

$$p_i = \begin{cases} \sum_{j=1}^{i} C_j & si \quad C_j \neq 0 \\ 0 & Sinon \end{cases}$$

4. Implémentation et résultat expérimentaux

4.1. L'environnement de développement :

L'algorithme de recherche binaire et l'algorithme ACOBS ont été développé sous l'environnement suivant :

- Micro-portable acer i3 (CPU 1.7GHz, RAM 4Go).
- Système d'exploitation Microsoft Windows 7.

4.2.Langage de programmation :

Notre application est implémentée en utilisant le langage JAVA. Le langage Java est un langage généraliste de programmation synthétisant les principaux langages existants lors de sa création en 1995 par Sun Microsystems [10]

4.3.Interface de l'application

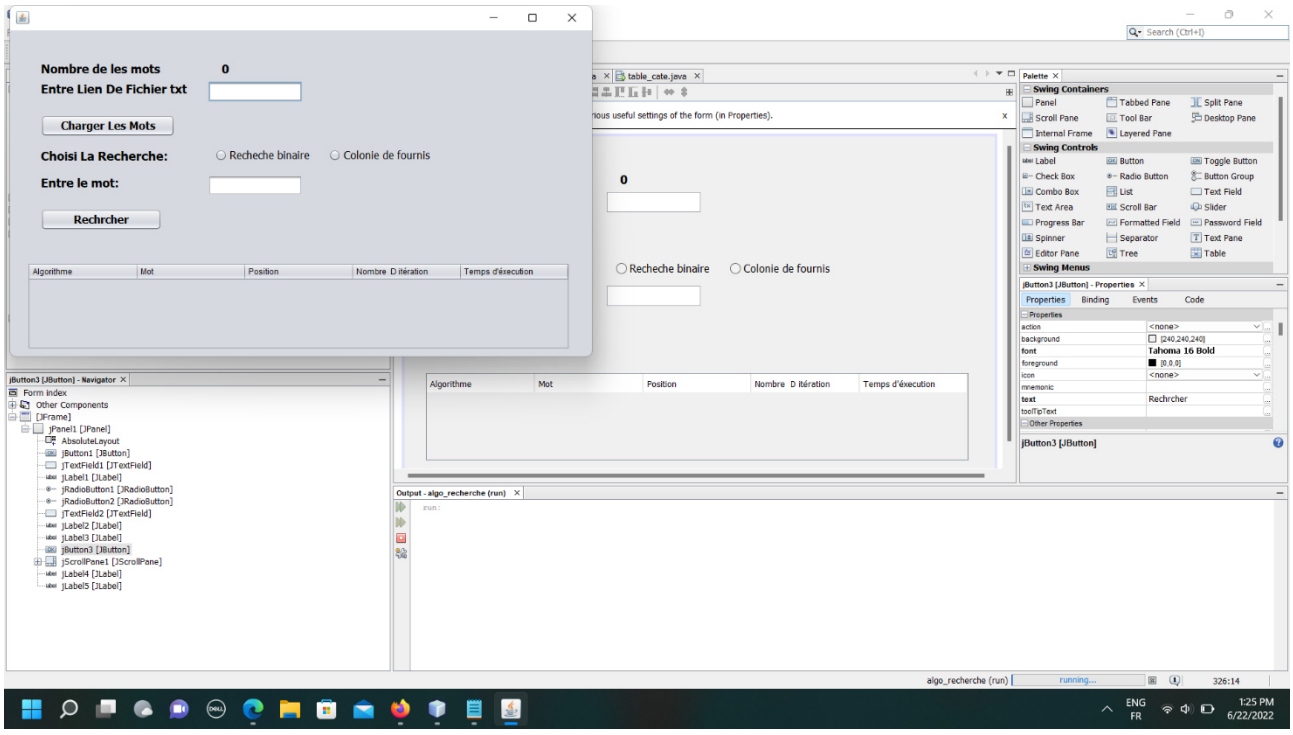

Fig. 3.2. Interface de RB et ACOBS

A partir cette interface (**Fig. 3.2),** on commence par introduire le fichier texte, puis le mot à rechercher. En choisissant la méthode de recherche (RB ou ACOBS), le programme va afficher la position du mot à rechercher ainsi que le temps d'exécution du programme.

Les tableau 3.1 et 3.2 montrent quelques exemples d'exécution du programme en utilisant l'algorithme de RB et ACOBS respectivement.

Mot à rechercher	**La position**	**Temps d'exécution**
z	25	0.011ms
zwittrionic	315125	0.012ms
zoouews	1	0.02ms
yasmine	149158	0.01ms

Tableau 3.1. Résultats de recherche par RB

Agent fourmis	**Taille mot**	**Position basse (lp)**	**Position haute (hp)**	**Phéromone**	**valeur**
1	1	0	1	1	+1
1	2	1	15	14	+1
1	3	15	119	104	-1
1	4	403	1073	678	-1
1	5	1073	2245	1175	-1
1	6	2245	3857	1612	+1

Tableau 3.2. Résultats de recherche par ACOBS

4.4. Comparaison entre ACOBS et la recherche binaire pour rechercher des mots dans un fichier texte

Algorithme	**Espace de recherche**	**Mot à rechercher**	**Position**	**Temps d'exécution (sec)**
ACOBS	13143	yasmine	3857	0.0001ms
RB	13143	yasmine	3857	0.0122ms
ACOBS	13143	Iyad	1073	0.004ms
RB	13143	Iyad	1073	0.0231ms

Tableau 3.3. Comparaison entre RB et ACOBS

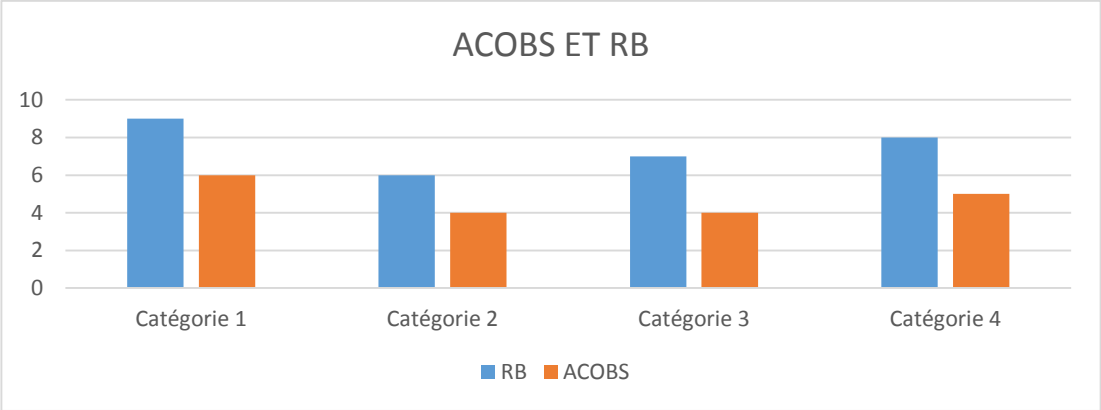

Fig. 3.3. Comparaison entre le nombre total de recherches utilisant ACOBS et RB

Le tableau 3.3 montre les résultats de comparaison entre les deux méthodes RB et la RB amélioré par l'algorithme de colonies de fourmis (ACOBS). Les résultats montrent que sur toutes les instances testées, le temps d'exécution de l'algorithme ACOBS est inférieur au temps d'exécution de l'algorithme de RB. Cela est dû au fait que l'algorithme ACOBS a effectué la recherche dans un espace de recherche réduit.

5. Conclusion

La complexité temporelle de l'algorithme de recherche binaire dans les pires des cas est **O ($\log_2$ n)** où **n** représente le nombre des éléments dans l'espace de recherche. L'idée de l'approche proposée est de réduire l'espace de recherche pour réduire la complexité temporelle de l'algorithme de recherche.

La complexité temporelle de l'algorithme ACOBS est de **O ($\log_2$ c)** où **c** représente le nombre des éléments dans l'espace de recherche réduit ($\boldsymbol{c < n}$). De ce fait on peut dire que l'algorithme ACOBS convient mieux aux applications dans lesquelles la recherche est effectuée sur un très grand espace de recherche.

Conclusion générale

Dans ce travail, nous avons effectué une étude expérimentale des méthodes de recherche dans un tableau (ou espace de recherche). En particulier, nous avons présenté un algorithme amélioré de la recherche binaire par l'algorithme de colonies de fourmis nommé ACOBS.

Ce mémoire a été organisé comme suit :

Le premier chapitre a été consacré à la présentation des notions de base sur l'optimisation combinatoire. Dans le deuxième chapitre, nous avons passé en revue quelques algorithmes de recherche, puis nous avons détaillé l'algorithme de recherche binaire. Dans le troisième chapitre, nous avons présenté l'approche amélioré de la recherche binaire.

Des expérimentations comparatives des deux algorithmes (RB et ACOBS) ont été réalisées. Les résultats ont montré les points suivants :

- La complexité temporelle de l'algorithme de recherche binaire dans les pires des cas est O (log2 n) où n représente le nombre des éléments dans l'espace de recherche
- L'idée de l'approche proposée est de réduire l'espace de recherche pour réduire la complexité temporelle de l'algorithme de recherche.
- La complexité temporelle de l'algorithme ACOBS est de O (log2 c) où c représente le nombre des éléments dans l'espace de recherche réduit ($\boldsymbol{c}<\boldsymbol{n}$).

De ce fait, on peut dire que l'algorithme ACOBS convient mieux aux applications dans lesquelles la recherche est effectuée sur un très grand espace de recherche.