République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université A.MIRA-BEJAIA

جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

Faculté de Technologie
Département d'Automatique,
De Télécommunication et d'Electronique

Projet de Fin d'étude

Pour l'obtention du diplôme de Master en Automatique et Systèmes

Thème

Réduction des capacités de calcul requises par un algorithme bio-inspiré

Préparé par :
M[r]. BERBAR Salah
M[lle]. MEDJAHED Sonia

Examiné par :
M[r]. N.NAIT MOHAND
M[me]. M. GAGAOUA

Dirigé par :
Dr Lyes TIGHZERT
Dr Hocine Lehouche

Année Universitaire : **2018/2019**

Remerciements

On remercie Dieu le tout-puissant, qui nous a donné la force et la patience pour l'accomplissement de ce travail.

Nous remercions en particulier Dr. Lyes TIGHZERT pour l'honneur qu'il nous a fait de bien vouloir nous encadrer, et pour les conseils donnés lors de la réalisation de ce travail.

On adresse nos remerciements au membre de jury d'avoir accepter de nous prêter de leur attention et évaluer notre travail.

Dédicaces

Je dédie ce modeste travail:

- *A toute ma famille*
- *A mes chers parents*
- *A mes sœurs Katia et Nassima*
- *A mon frère Faouzi*
- *A tous mes proches et amis*
- *A ma collègue Sonia ainsi à toute sa famille*

Salah

Dédicaces

Toutes les lettres ne sauraient trouver les mots qu'il faut…
Toutes les mots ne sauraient exprimer la gratitude, le respect, la reconnaissance…
Aussi, c'est tout simplement que
Je dédie ce mémoire

A celle qui m'a donnée la vie, le symbole de tendresse, qui s'est sacrifiée pour mon bonheur, et qui a été l'école de mon enfance, à ma mère.

A mon père, qui a été mon ombre durant toutes les années d'études et qui a veillé tout au long de ma vie à m'encourager et à me protéger.

A mes quatre sœurs Sabrina et son mari Yacine , Yamina et son mari Razik, Kahina, Samia, je les remercie du fond du cœur pour leurs soutients

A mes grands-parents maternels

A la mémoire de mes grands-parents paternels qui ont été une joie qui nous manque énormément aujourd'hui.

A mes oncles et cousins

A mon très cher oncle nadji

A tous les membres de ma famille qui m'ont soutenu

A mon ami et collègue Salah

A tous mes amis

A ceux qui ont été toujours à mes côtés avec leurs aides morales et leurs encouragements

Sonia

Sommaires

CHAPITRE III : Algorithme d'essaim de blattes compact

LISTE DES TABLEAUX

LISTE DES FIGURES

CHAPITRE I : Algorithmes inspirés de la nature

CHAPITRE II : Algorithme d'essaim de blattes

CHAPITRE III : Algorithme d'essaim de blattes compact

Liste des Acronymes

AE : Algorithme évolutionnaire.
AG : Algorithme génétique.
SE : Stratégie d'évolution.
PE : Programmation évolutionnaire.
AIS : système immunitaire artificiel.
MA : Monkey Algorithm.
BB – BC : Big-BagBig-Crunch.
CRO : Chemical Reaction Optimization.
BSA : Backtracking Optimization Search.
SI : Swarm Intelligence.
CSO : Cockroach Swarm Optimization.
RIO : Roach Infestation Optimization**.**
BCSO : Binary Cockroach Swarm Optimization.
ICSO : improved cockroach swarm optimization.
cSI : compact swarm Intelligence.
rcGA : Algorithme génétique réel compacté.
FA : Firefly algorithm.
TLBO : teaching-learning based optimization
DE : diffirentiel evolutionaire
PSO : particle swarm optimization
PV : vecteur de probabilité

INTRODUCTION GÉNÉRALE

Contexte

La résolution d'un problème d'optimisation consiste à explorer un espace de recherche afin de maximiser ou minimiser une fonction donnée. Les algorithmes évolutionnaires sont des procédures très robustes pour résoudre un problème d'optimisation. Néanmoins elles présentent certaines limites et difficultés qui influent fortement sur la convergence de ce type d'algorithme.

Lorsqu'un nouveau problème se pose en ingénierie, il faut parfois définir de nouvelles méthodes de résolution, car les techniques existantes ne sont pas toujours précisément adaptées au cas traité. Lorsque l'on veut inventer une nouvelle méthode de résolution de problème, il faut souvent une source d'inspiration.

La biologie moderne a fait faire un saut extraordinaire dans la compréhension des mécanismes biologiques intervenant dans le monde du vivant. Cette connaissance ouvre actuellement un champ d'investigation très intéressant qui permet de mettre en œuvre les méthodes expérimentales et théoriques pour l'observation et la modélisation des différents comportements espèces biologiques, de l'échelle microscopique aux systèmes beaucoup plus complexes, cette approche été une source d'inspiration très riche pour les scientifiques. Ce mémoire est consacré aux algorithmes inspirés des systèmes biologiques et évolutionnaires. L'approche que nous proposons s'inspire des comportements des essaims de blattes.

Problématique

Ces dernières années, de nombreuses méthodes d'optimisations basées sur des populations de candidats sont largement étudiées par les chercheurs et les scientifiques. L'un des développements récents dans la swarm intelligence (SI), ou intelligence des groupes, est l'optimisation basée sur les comportements des blattes. Cette nouvelle méthode est utilisée pour résoudre des problèmes complexes, le Cockroach Swarm Optimization Algorithm (CSOA) est un algorithme, méta-heuristique, basé sur un essaim de candidats en interaction entre eux et leur environnement. Par conséquent, cette approche nécessite des capacités des calculs élevées puisque la taille de la population grandit en fonction de la dimension du problème à résoudre. L'implémentation de ce genre d'algorithmes est souvent difficile, et voire même impossible, quand les capacités de calcul et de mémoire ne sont pas disponibles. Ceci constitue alors un point faible dans leur implémentation sur des microcontrôleurs. Dans cette étude, on présentera

une solution à ce problème. L'objectif principal dans ce mémoire est la réduction des capacités de calcul et du stockage en mémoire de l'algorithme d'essaim de blattes (CSOA).

La solution proposée

Le succès des algorithmes compact et de l'intelligence des essaims compactés, nous conduit à étudier le cas du CSOA. Dans le cadre de ce mémoire, nous proposons d'exploiter les notions de la représentation compacte afin d'aboutir à un algorithme CSOA qui ne nécessite qu'un strict minimum de mémoire et de capacités de calcul. L'idée de base consiste à représenter l'essaim du CSOA par des densités de probabilités. L'algorithme obtenu est appelé compact Cockroach Swarm Optimization Algorithm (cCSOA). Ce dernier donne des résultats très compétitifs, sur des problèmes d'optimisation difficile, comparé à la version originale du CSOA et d'autres algorithmes tels que le PSO et les algorithmes évolutionnaires.

La méthodologie

Afin d'évaluer, de valoriser et d'étudier les limites de l'algorithme cCSOA, nous proposons une large étude comparative. Le black box de IEEE, le CEC2014, est utilisé ici pour fournir un ensemble de tests issus des problèmes mathématiques d'optimisation les plus difficiles, reconnus par les chercheurs au niveau mondial. Les résultats obtenus révèlent l'efficacité de notre approche.

L'organisation de ce mémoire

Ce mémoire est divisé en trois chapitres, outre cette introduction et la conclusion, nous présentons dans le chapitre 1 les techniques bio-inspirés tels que les algorithmes évolutionnaires, l'intelligence des essaims ainsi que d'autres approches inspirées de la physique ou de la chimie. Nous mettons aussi l'accent sur les différents aspects du vivant ayant inspiré les ingénieurs.

Dans le chapitre 2, nous nous intéressons à l'intelligence des essaims et particulièrement à l'algorithme CSOA. Nous présentons son fondement et son développement.

Dans le chapitre 3, nous présentons l'algorithme que nous avons développé le compact Cockroach Swarm Optimization Algorithm (cCSOA). Une large validation expérimentale est présentée. Les résultats obtenus sont très compétitifs.

Chapitre I

Algorithmes Inspirés de la Nature

Chapitre I

Algorithmes inspirés de la nature

1. Introduction

La nature est le réservoir des technologies de demain. Le biomémitisme nous a permis d'en prendre exemple et de s'en inspirer. Ceci a donné des fruits dans plusieurs domaines. Les pionniers en la matière ont observé l'incroyable diversité des espèces vivantes et leurs comportements face aux changements de l'environnement afin de les imiter et de créer des procédés destinés à l'industrie. En effet, les avions sont une imitation des oiseaux, les sous-marins sont une imitation des poissons et les ordinateurs cherchent à imiter le cerveau de l'humain, etc.

Plusieurs questions préoccupent les biologistes : dans une colonie d'insectes sociaux, telles fourmis ou les abeilles, pourquoi le groupe est-il souvent cohérent alors que chaque individu semble autonome? Comment les activités de tous les individus sont-elles coordonnées sans supervision? Par exemple, les éthologistes qui étudient le comportement des insectes sociaux ont observé que la coopération au sein des colonies de fourmis est auto organisée, cette dernière résulte d'interactions entre les individus. Bien que ces interactions puissent être simples, elles permettent à la collectivité de résoudre des problèmes complexes, tels que la recherche du chemin le plus court entre le nid et une source de nourriture parmi d'innombrables voies possibles [1]. Toutes ces observations et interrogations ont permis l'émergence d'un nouveau paradigme de calcul intelligent inspiré de la biologie « Bio-inspired Computing» ou calcul bio-inspiré, pour traiter les problèmes complexes et dynamiques du monde réel. Ce nouveau paradigme a connu un grand succès. Il a contribué à la compréhension des différents phénomènes naturels, ainsi qu'au développement de nouvelles techniques de calcul en réduisant considérablement la complexité algorithmique [2].

2. Algorithmes Evolutionnaires

Les algorithmes évolutionnaires (AE) sont des algorithmes pseudo-aléatoires qui prennent leur inspiration de la théorie darwinienne de l'évolution naturelle et de la génétique afin de trouver des solutions aux problèmes d'optimisation complexes.

Les (AE) fondés sur les principes de la survie des individus les mieux adaptés selon la diversité génétique de l'évolution issue du croisement, mutation, sélection, etc. L'idée de base consiste à réaliser des croisements de la chaine chromosomique, des mutations et des sélections. Ce qui permet à l'algorithme de créer la génération suivante d'individus. Ce processus de

diversification aide l'algorithme à éviter la convergence de la population vers une solution qui n'est pas optimale.

L'organigramme ci-dessous représente un algorithme évolutionnaire, il s'agit de générer et de simuler un groupe d'individus tirés aléatoirement et de suivre leur évolution en leur appliquant les différents opérateurs (reproduction, mutations…) et que l'on soumet à une sélection. Si la sélection s'opère à partir de la fonction d'adaptation, alors la population tend automatiquement à s'améliorer. [3] On note que ceci ne garantit pas la convergence.

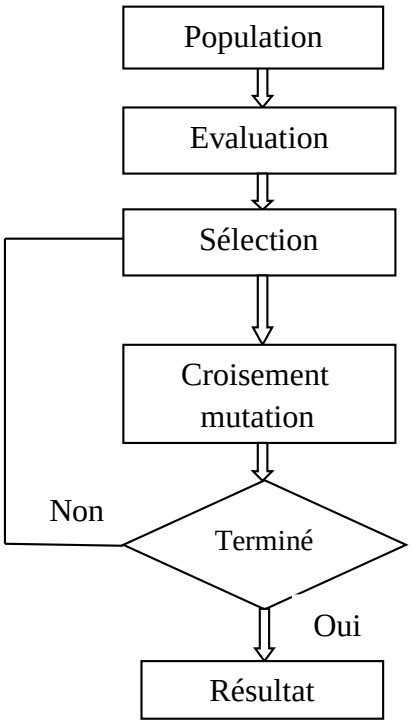


Figure I.1. L'organigramme de l'algorithme évolutionnaire

2.1.Les étapes d'un algorithme évolutionnaire

Les algorithmes évolutionnaires suivent généralement les étapes suivantes :

- Création aléatoire d'une population de base d'un groupe de N qui représentation des solutions possibles au problème à résoudre.
- **Évaluation** des N individus composants la population par la fonction objective.
- **Sélection** de M individus dans la population pour le croisement.
- **Croisement** des M individus sélectionnés.
- **Mutation** d'une partie d'individus M issus du croisement.
- **Évaluation** par la fonction objective des individus résultants des croisements et des mutations.

- **Sélection** de N individus parmi la population (parents et enfants) c'est l'étape de remplacement (domination darwinienne). Si un critère d'arrêt est atteint, l'algorithme s'arrête et le ou les meilleurs individus sont sélectionnés. Dans le cas contraire, on recommence le processus à l'étape de sélection de M individus dans la population pour le croisement.

2.2. Types d'algorithmes évolutionnaires

Il y a plusieurs types d'algorithmes évolutionnaires, ils sont souvent classifiés pour différentes raisons, par le type d'opération de mutation. On distingue quatre types (AE) comme indiqué dans la **figure I.2** [5,6].

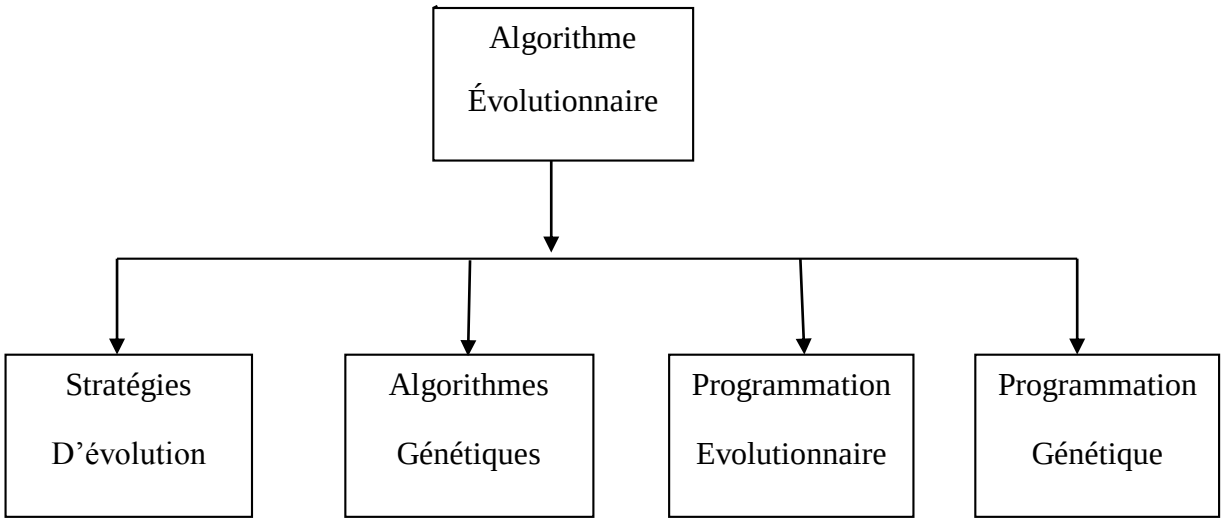


Figure I.2. Types d'algorithmes évolutionnaires

- **Algorithmes génétiques** (AG), ce sont les algorithmes les plus populaires en optimisation et sont aussi utiles pour comprendre les mécanismes de l'adaptation et de l'évolution naturelle. Ils ont été développés à l'Université de Michigan (USA) par J.Holland (1975), Goldberg (1989), L. Davis (1991) et Michalewicz (1992). Ils s'appuient sur les opérateurs de variation, le croisement et la mutation, pour créer une large variété de solutions. La représentation des génotypes des individus, qui est à l'origine de type binaire, a été par la suite développée à de nombreuses autres formes de représentation [5,6].
- **Stratégies d'évolutions** développées par Rechenberg et H.P. Schwefel, (1965). Elles ont été développées pour résoudre des problèmes d'optimisation à variables réelles posés au milieu industriels et pour les quels n'existe pas de fonction objective analytique; le contexte étant l'optimisation paramétrique. Ce sont les meilleurs algorithmes pour les

problèmes purement numériques. Ce modèle de stratégies d'évolution utilise le principe de mutation sur les réels du modèle de la programmation évolutive avec un taux de mutation plus grand. Cette augmentation peut être interprétée par le fait que si la proportion de mutation réussie est élevée, l'espace de recherche exploré est limité autour d'un optimum local; il faut donc diversifier la population en augmentant le taux de mutation. Ces approches utilisent un opérateur de sélection de type déterministe, les solutions dont la fonction d'adaptation est mauvaise sont éliminées de la population. En outre, dans le modèle original, les populations des parents et de leurs descendants sont généralement de taille différente

- **Programmation évolutionnaire** : (PE), ce sont des algorithmes destinés à évoluer des automates, développés par [L.J. Fogel, 1964 et D.B. Fogel, 1991, 1995, Californie, USA], ce modèle est souvent appliqué à la résolution des problèmes d' optimisation à variables réelles dans un espaces de recherche très variés. Ils utilisent des mutations et des croisements qui permettent de donner à la fin du processus un programme exécutable. [5]
- **Programmation Génétique :** (PG), ce sont des algorithmes qui permettent de générer des fonctions informatiques à partir des principes évolutionnaires, la population est un ensemble de codes de base de programmes informatiques, cette approche a été développer par [J. Koza, 1990, Californie, USA]. [6]

3. Algorithmes Génétiques

Les algorithmes génétiques sont des méthodes stochastiques d'optimisation, initiés dans les années 1970 par John Holland, ce dernier a introduit le premier modèle formel de ces algorithmes (the canonical genetic algorithm AGC) dans son livre (Adaptation in Natural and Artificial Systems). [7] Ce modèle nous amène à comprendre l'utilisation de l'intelligence dans un programme informatique avec la mutation et le croisement. Depuis l'apparition de cet algorithme, plusieurs chercheurs ont proposé d'autres algorithmes. En 1989 Goldberg a développé ce modèle et rajouta que l'individu est lié à son environnement par son ADN ainsi que la solution est liée à un problème par son indice de qualité [7].

3.1. Principe général du fonctionnement d'un algorithme génétique

Le but d'un (AG) est de trouver la chaine qui minimise une fonction objective, ces algorithmes sont basé sur les principales phases suivantes : [6].

- **Initialisation :** Une population initiale de N individus est tirée aléatoirement.
- **Evaluation :** Chaque individu est décodé, puis évalué.

- **Sélection :** Une nouvelle population est créée par l'utilisation de l'une des méthodes de la sélection naturelle.
- **Reproduction :** Possibilité de croisement et de mutation au sein de la nouvelle population.
- Retour à la phase d'évaluation jusqu'à la vérification du critère d'arrêt de l'algorithme.

3.2.Les domaines d'utilisation des Algorithmes génétiques

Ces Algorithmes sont utilisée dans plusieurs domaines comme :

- Optimisation : optimisation de fonctions, planifications … etc.
- Apprentissage : classification, prédiction, robotique … etc.
- Programmation automatique : automates cellulaires … etc.
- Etude du vivant, du monde réel : marchés économiques, comportements sociaux.

4. Systèmes immunitaires artificiels

Le système immunitaire est une collection de cellules, des molécules et des organes qui sont capable de protéger le corps contre les maladies et les infections. On se basant sur ce système les chercheures ont pu développer dans ces dernières années le modèle immunitaire artificiel pour la résolution de divers problèmes. [7]

4.1.Définition du système immunitaire artificielle

Un système immunitaire artificiel (AIS) est un système informatique basé sur les métaphores du système immunitaire naturel [8].Les algorithmes exploitent typiquement les caractéristiques du système immunitaire d'étude et de mémoire pour résoudre un problème (autodéfense). Ils sont couplés à l'intelligence artificielle et quelques algorithmes d'AIS sont étroitement liés aux algorithmes génétiques [9].

4.2.Les types de systèmes immunitaires [10]

La méthode des systèmes immunitaires artificiels est appliquée dans plusieurs secteurs pour des buts différents comme la sécurité des ordinateurs, l'optimisation, la robotique, la détection et l'élimination des virus informatiques … etc.

- **Sélection négative :** La sélection négative (ou détection négative) est une abstraction des mécanismes qui permettent aux systèmes immunitaires naturels de distinguer entre le soi et le non soi. Elle se concentre sur la génération de détecteurs de changements, ces détecteurs sont censés détecter qu'un élément d'un ensemble de chaînes (le soi) a changé. Dans cet algorithme, le soi correspond aux données à protéger et le non-soi aux données indésirables. La génération de détecteurs proposée par Forrest [10] est la suivante :

1. Définir les données du soi.

2. Générer les détecteurs aléatoirement.

3. Comparer chaque détecteur généré avec les données du soi. S'il en détecte une, il l'a garde, sinon il la supprime.

- **Algorithme de la sélection clonale**

La sélection clonale artificielle est une abstraction des mécanismes de mémorisation des systèmes immunitaires. Les algorithmes développés sont généralement dédiés à l'optimisation ou à la recherche, l'algorithme à la forme suivante :

1. Produire un ensemble de solutions (répertoire d'anticorps) de N candidats qui sont définis par le problème à étudier.
2. Choisir les *n*1 cellules qui ont la plus grande affinité à l'antigène.
3. Produire des copies identiques de ces cellules choisies. Le nombre de copies est proportionnel aux affinités : plus l'affinité est haute, plus le nombre de clones est grand.
4. Changer la structure des cellules choisies (hyper mutation). Le taux de changement est proportionnel à leurs affinités : plus l'affinité est haute, plus le taux de Changement est petit.
5. Sélectionner les *n*2 cellules (du résultat de l'étape 4) qui ont la plus grande affinité à l'antigène pour composer le nouveau répertoire.
6. Remplacer quelques cellules qui possèdent des valeurs d'affinité faible par les nouvelles cellules.
7. Répéter les étapes 2 à 6 jusqu'à ce qu'un critère d'arrêt soit satisfait.

- **Réseaux immunitaires**

La théorie immunitaire de réseau semble très attrayante sur l'intelligence artificielle. Car premièrement, elle présente un système dynamique capable d'exécuter des interactions entre ses propres constituants et entre ses constituants et l'environnement externe. Deuxièmement, elle offre les possibilités d'ajuster la structure du système (réseau) et les paramètres du système sur l'environnement. Son algorithme général est :

1. **Initialisation**: initialiser un réseau de cellules immunisées.
2. **Boucle de population :** pour chaque antigène :
   - **2.1. Identification antigénique :** calculez les affinités des anticorps face à l'antigène ;
   - **2.2. Interactions du réseau :** calculez les affinités entre tous les paires d'anticorps ;
   - **2.3. Méta dynamique :** ajoutez des nouveaux anticorps aux réseaux et supprimez ceux sont inutile (le choix est basé sur un certain critère) ;

2.4. Niveau de stimulation : évaluez le niveau de stimulation de chaque cellule du réseau tenant compte des résultats des étapes précédentes ;

2.5. Dynamique du réseau : mettre à jour la structure et les paramètres du réseau selon le niveau de stimulation de différentes cellules ;

3. Cycle : répéter l'étape 2 jusqu'à ce qu'un critère donné d'arrêt soit satisfait;

5. Intelligence des essaims

Il y a longtemps, les gens ont découvert la diversité des comportements intéressants des insectes ou des animaux qui vivent en groupes. Une bande d'oiseaux balaie le ciel. Un groupe de fourmis cherche de la nourriture. Un banc de poissons nage, se retourne, s'enfuit ensemble, etc.… Nous appelons ce type de comportement global des essaims. Récemment, des biologistes et des informaticiens du monde ont étudié comment modéliser des essaims biologiques pour comprendre comment de tels animaux interagissent, atteignent des objectifs et évoluent. En outre, les ingénieurs s'intéressent de plus en plus à ce type de comportement.

L'intelligence des essaims (SI) apporte des solutions aux problèmes d'optimisation motivés par le comportement collectif des animaux sociaux. Les animaux qui essaiment ont été un domaine d'intérêt attirant de nombreux chercheurs. Les mécanismes qui contrôlent les insectes sociaux et le comportement des animaux sont restés étonnants; les animaux se déplacent en groupe, tournent ensemble, évitent des obstacles, ils sont très intelligents dans leurs mouvements [11]. En coopérant, le groupe est en mesure d'accomplir des tâches complexes, malgré le fait que les individus dans ces colonies ne sont pas sophistiqués. Des exemples d'algorithmes d'essaims d'insectes et d'animaux sont représentés dans la section suivante.

5.1. Quelques exemples

Toute créature vivante peut survivre et se protéger de sa propre manière, telle que la recherche de nourriture, la défense contre les prédateurs. Les chercheurs se sont inspirés pour résoudre une variété de problèmes.

5.1.1. Monkey Algorithm (MA) :

(MA) est un algorithme de l'intelligence en essaim. Il a été proposé par Ruiqing et Wansheng, cette algorithme est utilisé pour résoudre des problèmes d'optimisations multimodaux, Le processus obtient de la simulation d'escalade des singes. Il se compose de trois processus : le processus de montée, le processus de surveillance, et le processus de saut

périlleux. Le but du processus de montée est de rechercher des solutions locales les plus favorables.

Le but du processus de saut périlleux est de faire découvrir aux singes une nouvelle zone de recherche et cette action évite de se retrouver dans la recherche de quartier [12].

5.1.2. Fich Swarm

Les poissons peuvent découvrir la région avec plus de nourriture dans l'eau par l'exploration individuelle ou en suivant d'autres poissons.

Par conséquent, la région de l'eau où la quantité de poissons est la plus concentrée est généralement la plus nutritive. Selon cette qualité, le modèle Poisson artificiel est présenté les trois types de comportements : le comportement de proie, le comportement d'essaimage et le comportement de poursuite. Le comportement de proie est un comportement biologique de base qui tend à la nourriture. Dans le comportement d'essaimage, les poissons se rassembleront en groupes naturellement en mouvement. Pendant qu'ils grouillent. Le comportement de poursuite : Dans le processus de déplacement de l'essaim de poissons, lorsqu'un seul poisson ou plusieurs d'entre eux trouvent de la nourriture, les agents de l'essaim suivront et se rendront rapidement à la nourriture [13].

5.1.3 Cat Swarm

Cat Swarm est un algorithme d'optimisation qui décrit le comportement commun des chats. Les chats ont la concentration sur les objets en action et ont une grande compétence de suivi [14]. Il pourrait penser que les chats passent la plupart du temps au repos, mais en fait ils sont toujours attentifs et se déplaçant graduellement. Cette action correspond au mode de recherche.

En outre, lorsque les chats remarquent une proie, ils gaspillent beaucoup d'énergie en raison de leurs mouvements rapides. Cette action correspond au mode de traçage [15].

6. Algorithmes Inspirés de la physique et de la chimie

Tous les algorithmes métaheuristiques ne sont pas tous obligatoirement bio-inspirés. Il existe d'autres approches dont la source d'inspiration provient de la physique et de la chimie. Pour les algorithmes qui ne sont pas bio-inspirés, la plupart ont été développés en imitant certaines lois physiques ou chimiques qui régissent notre univers, notamment les charges électriques, la gravité, les systèmes fluviaux, etc. Même si la physique et la chimie sont deux matières différentes, il est toutefois inutile de les mettre dans deux catégories différentes. Après tout, beaucoup de lois fondamentales sont les mêmes, donc nous les regroupons simplement dans des algorithmes basés sur la physique et la chimie.

6.1 Big-Bang Big-Crunch (BB-BC) :

La méthode BB-BC inventée par Erol et Eksin se compose de deux phases : une phase Big Bang et une phase Big Crunch. Dans la phase Big Bang, les solutions des candidats sont réparties arbitrairement au cours de la place. Erol et Eksin ont lié la nature aléatoire du Big Bang à la dissipation de l'énergie ou à la remise d'une solution convergente à un nouvel ensemble de candidats à la solution. La phase Big Ban est suivie par le Big Crunch qui a de nombreuses entrées, mais une seule sortie connue sous le nom de centre de masse. Le seul résultat a été atteint en calculant le centre de masse [16].

Algorithme de La méthode BB-BC

Big Bang phase (solutions construction):

***Step 1*:** Generate population .

Big Crunch phase (Local Search move):

Repeat

Step2: Generate some neighbors for all solutions in the population and replace the parent withits best off- spring for each solution in the population;

Step 3*:* Find the centre of mass;

Step 4: Apply local search to the centre of mass;

Step 5: Update the elite pool and the best found solution;

Step 6: Eliminating some poor quality solutions;

Until population size isreduced to a single solution;

Step 7: Return to Step 1 if stopping criterion is not met;

Step 8: Return the best found solution.

Figure I.3. Le pseudo code de BB-BC algorithm

6.2. Chemical Reaction Optimization algorithm (CRO)

L'optimisation des réactions chimiques (CRO) est un algorithme méta-heuristique inspiré des réactions chimiques. Cela commence avec les molécules initiales, et en effectuant une séquence de collisions, le produit final devient à l'état stable.

La principale différence entre l'algorithme CRO et toute autre technique évolutive est que la taille de la population dans le CRO peut changer après chaque itération de l'algorithme en cours d'exécution, alors que dans toutes les autres techniques, la taille de la population est généralement fixe et inchangée pendant l'exécution [17].

7. Autres algorithmes

Parfois, il n'est pas facile de mettre des algorithmes dans les groupes ci-dessus, car ces algorithmes ont été développés selon différentes caractéristiques issues de différentes sources, sociales, émotionnelles, etc.

7.1. Backtracking Optimization Search :

BSA, un nouvel algorithme évolutif (EA) utilisé pour résoudre des problèmes d'optimisation numérique à valeurs réelles. BSA peut être expliquée en divisant ses fonctions en cinq processus comme cela se fait dans d'autres: initialisation EAs, sélection, mutation, croisement et sélection. [18].

8. Conclusion

Dans ce chapitre nous avons présenté différents types de systèmes dits bio-inspirés, nous avons donné pour chacun les caractéristiques qui le définissent et pour en conclure nous pouvons dire que **:**

- Les algorithmes évolutionnaires sont un outil puissant pour l'optimisation. En plus du fait de s'inspirer de l'évolution des espèces et d'avoir une « légitimité » biologique, ils disposent d'une base théorique solide qui permet leur mise en application dans divers domaines. Nous avons vu par ailleurs qu'il y a de très nombreuses variantes d'algorithmes évolutionnaires.
- Les Systèmes Immunitaires Artificiels constituent une nouvelle approche d'optimisation robuste et flexible qui peut être appliquée à une grande variété des problèmes.
- De nombreux constats montrent que les êtres vivants présentent des caractéristiques telles que l'autoréparation, l'adaptation, l'apprentissage, l'évolution et le développement. Ces concepts sont maintenant bien utilisés et bien maîtrisés dans le domaine de l'ingénierie.
- Tous les algorithmes métaheuristiques ne sont pas tous obligatoirement bio-inspirés, car leurs sources d'inspiration proviennent aussi de la physique et de la chimie.
- La nature reste un mystère extraordinaire utile pour les chercheures qui s'inspirent pour résoudre différents problèmes.

Dans le chapitre suivant, nous allons présenter l'algorithme d'essaim de blattes (CSOA) qui est l'objet de ce mémoire. Cet algorithme, comme ceux présentés dans ce chapitre, utilise une population de candidat pour résoudre un problème d'optimisation.

Chapitre II

Algorithme d’Essaim de Blattes

Chapitre II

Algorithme d'essaim de blattes

1. Introduction

L'optimisation des systèmes joue un rôle important dans divers domaines, notamment les mathématiques, l'économie, l'ingénierie et les sciences de la vie.

De nombreuses approches conventionnelles n'ont pas réussi à résoudre efficacement les problèmes complexes en raison de l'espace de solution de plus en plus grand. Cela a conduit au développement d'algorithmes évolutionnaires qui s'inspirent du processus d'évolution naturelle. On croit que la nature fournit des inspirations qui peuvent conduire à des modèles innovants ou des techniques pour résoudre des problèmes d'optimisation complexes.

Parmi la classe de modèle basée sur cette inspiration**,** on trouve la Swarm Intelligence (SI). Un paradigme SI est composé d'algorithmes inspirés par le comportement social des animaux et des insectes. Les algorithmes basés sur le SI ont attiré l'intérêt, gagné la popularité et l'attention en raison de leur souplesse et polyvalence. Algorithmes SI sont efficace dans la résolution des problèmes d'optimisation du monde réel. Exemples d'algorithmes du SI: comme l'optimisation des essaims des oiseaux, qui imite le comportement social des oiseaux [19]; l'optimisation des colonies de fourmis, imite le comportement social des fourmis [20]…Etc. Dans ce chapitre on s'intéresse à l'étude d'algorithme des blattes qui imite le comportement social d'un essaim de blattes.

2. Swarm intelligence

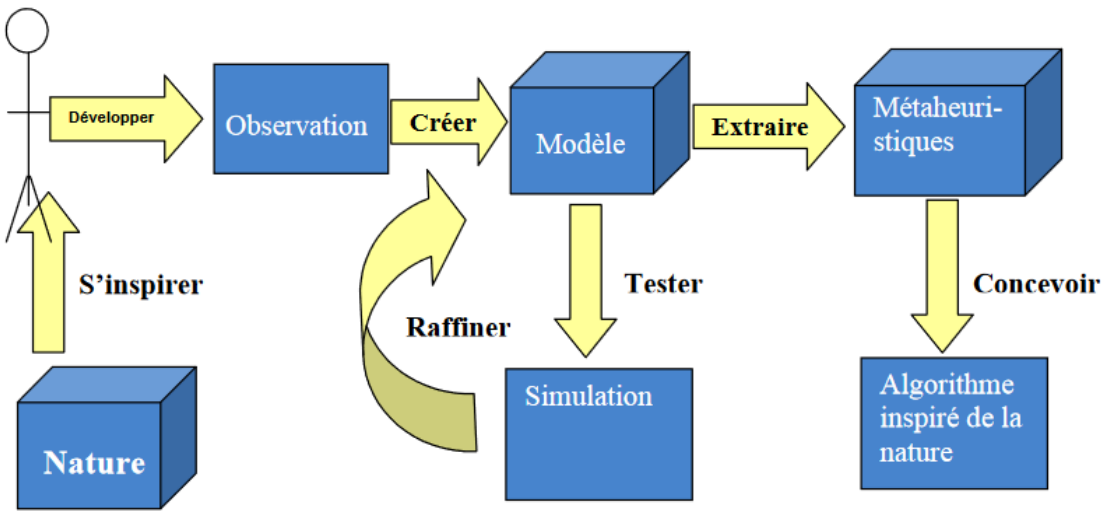


Figure II.1 - Processus d'inspiration d'un phénomène naturel

SI fournit des solutions aux problèmes d'optimisations qui s'inspirent généralement des comportements collectifs de certaines espèces dans la résolution de leurs problèmes [21].

Les mécanismes qui contrôlent le comportement des insectes et des animaux sociaux sont restés étonnants; les animaux se déplacent dans un groupe, se tournent ensemble, évitent des obstacles, ils sont très intelligents dans leurs mouvements [22]. En coopérant, le groupe est capable d'accomplir des tâches complexes, malgré le fait que les individus dans ces colonies ne sont pas sophistiqués.

3. Algorithme d'optimisation d'essaim de blattes

3.1. Inspiration

L'un des développements récents dans le SI est l'optimisation des blattes [23] L'algorithme de l'optimisation de l'essaim de blattes (CSOA) en anglais Cockroach Swarm Optimization Algorithm est un algorithme méta-heuristique simple et efficace, il a été introduit par Zhao Hui et HaiYan [26] en se basant sur l'observation du comportement des blattes. Initialement, il a été appliqué pour résoudre des problèmes d'optimisation continus [25].

Figure II.2- Essaim des blattes

Les blattes sont des insectes qui ont vécu plus de 0,35 milliards d'années, soit plus de 300 millions d'années que les dinosaures. Les blattes recherchent naturellement des abris sombres qui sont riches en ressources nutritionnelles. Ils se reposent en groupes dans ces abris

et se nourrissent la nuit. Ils restent fidèles à leur refuge tant qu'il y a assez de ressources riches dans leur environnement [27]. Ils sont extraordinairement capricieux, ce qui explique leur activité physique surtout la nuit. Il s'avère que, face au risque ou au danger, non seulement ils réagissent d'une manière assez impulsive, mais aussi leur comportement a un sens profond pour l'ensemble du groupe.

Les blattes ne possèdent pas une organisation sociale hiérarchique, ce qui leur permettrait de définir un dirigeant ou des leaders, même pendant un certain temps. Mais grâce aux interactions individuelles au sein du groupe, elles sont capables de résoudre des problèmes de décision complexes. Lorsqu'elles sentent le danger, elles se dispersent rapidement et explorent la région à la recherche d'un nouvel abri.

3.2. Fondement

L'algorithme d'optimisation de l'essaim de blattes (CSOA) a récemment été proposé par Chen et Tang [Chen et al. 2010]. Il est basé sur la population ou chaque individu est une solution potentielle au problème ciblé. Pour résumer les étapes de l'algorithme avec une simplicité suffisante, nous pouvons le diviser en trois étapes [31] :

- **Comportement de chasse-essaim** (aller en essaim) : une blatte solitaire *x*(i) va se déplacer vers la blatte la plus apte (optimum local) p(i) dans son champ visuel. Une blatte solitaire (dans son champ de visibilité) est un optimum local pour elle-même.

  Les blattes les plus fortes (pi) forment de petits essaims et suivent l'optimum global pg.

  Le comportement est modélisé par l'équation suivante

$$x(i)= \begin{cases} x(i) + step.rand.[p(i) - x(i)] & if\ x(i) \neq p(i) \\ . & \\ x(i) + step.rand.[p_g - x(i)] & if\ \ x(i) = p(i) \end{cases} \qquad \text{(II.1)}$$

  Où $x(i)$ est la position de blatte

  step est une valeur fixe

  rand est un nombre aléatoire dans [0,1]

  p(i)=opt$_j${x(j) $\|$ x(i)-x(j)$\|$ ≤ visual (rayon de vision) ,i=1,…N et j=1,…N**}** est la meilleure position locale

  pg=**opt$_i${x(i), i=1,…N}** est la meilleure position globale.

- **Comportement dispersant:** chaque blatte sera dispersée aléatoirement dans le but de maintenir la diversité de l'essaim actuel. Ce comportement est modélisé par l'équation suivante :

$$x(i) = x(i) + rand(1, D) \qquad \text{(II.2)}$$

  D est la dimension du problème.

- **Comportement brutal:** De temps en temps, nous devons également faire face à un comportement sans pitié, lorsque la meilleure blatte actuelle remplace une blatte choisie au hasard. Et en cas de manque de nourriture, la plus forte mange la plus faible (si elle est beaucoup plus «forte»)

$$x(k) = p_g \quad \text{(II.3)}$$

Où *k* est un entier aléatoire au sein de [1, N], *Pg* est la meilleure position globale.

3.3. Organigramme [29]

L'algorithme d'optimisation de l'essaim de blattes peut être présenté comme suit :

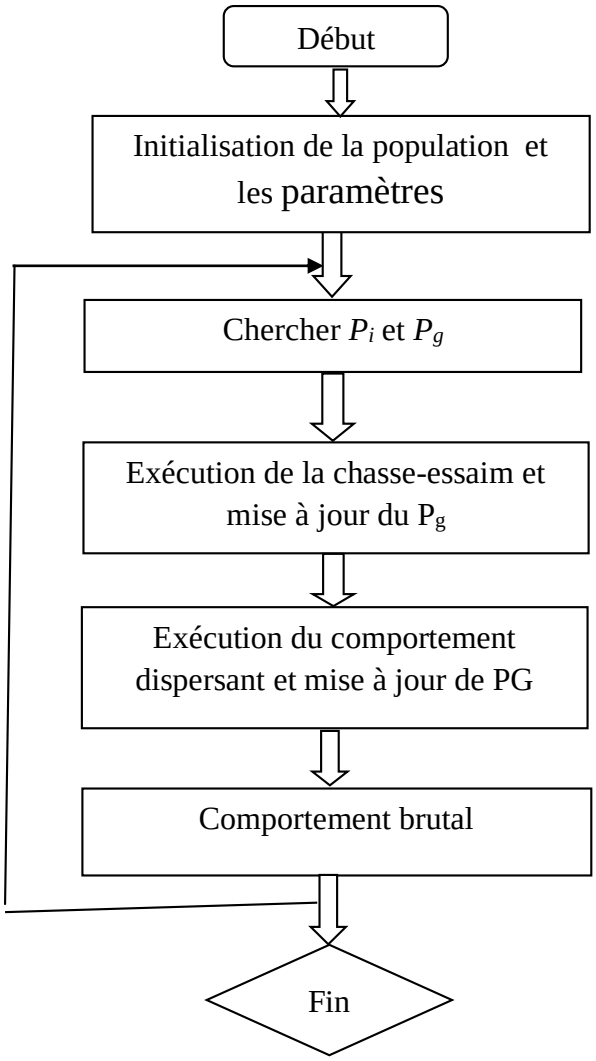


Figure II.3- Organigramme d'un Essaim de blattes

3.4.Pseudocode

Algorithm : COCKROACH SWARM OPTIMIZATION

```
1.  Objective function : f(x),x∈ℝ^D
2.  Set p_g=x_1
3.  For  i=2 to N do
4.          If  f(x_i)<f(p_g) then
5.                  p_g=x_i
6.          end if
7.  end for
8.  for t=1 to N  do
9.      for i=1 to N do
10.         for j=1 to N do
11.             if abs(x_i-x_j)<visual ; f(x_j)<f(x_i )then
12.                     p_i=x_i
13.             end if
14.         end for
15.         if  p_i==x_i then
16.             x_i=w.x_i+step*rand*(p_g-x_i)
17.         else
18.             x_i=w.x_i+step*rand*(p_i-x_i)
19.         end if
20.         if f(x_i)<f(p_g)  then
21.             p_g=x_i
22.         end if
23.     end for
24.     for i=1 to N do
25.         x_i=x_i+rand(1,D)
26.         if f(x_i)<f(p_g) then
27.         p_g=x_i
28.         end if
29.     end for
30. k=randint([1,N])
31. x_k=p_g
32. end for
```

Figure II. 4-pseudo-code de l'essaim des blattes [29]

L'algorithme CSOA est résumé par les étapes ci-dessous :

- **Étape 1**: initialiser les paramètres de l'algorithme et générer aléatoirement la population (step, rayon de vision, *D* est dimension spatiale, *N* est nombre d'individus).
- **Etape 2**: recherche du $p(i)$ (l'optimum local) dans le champ de vision de$x(i)$, et trouver Pg (l'optimum global).
- **Étape 3:** aller en essaim
  - Si une blatte $x(i)$ est localement la plus puissante (optimum local), la blatte va à l'optimum global Pg, qui est la blatte la plus puissante:

$$x(i) + step.rand.[pg - x(i)] \qquad Si\ x(i) = p(i)$$

  - Sinon, la blatte va à un optimum local de *p(i)*, qui est la blatte la plus puissante localement:

$$x(i) + step.rand.[p(i) - x(i)] \qquad Si\ x(i) \neq p(i)$$

  Mettre à jour le Pg.
- **Etape 4**:éffectuer le comportement dispersion :

$$x(i) = x(i) + rand(1, D)$$

  Mise à jour Pg.
- **Etape 5**: comportement brutal

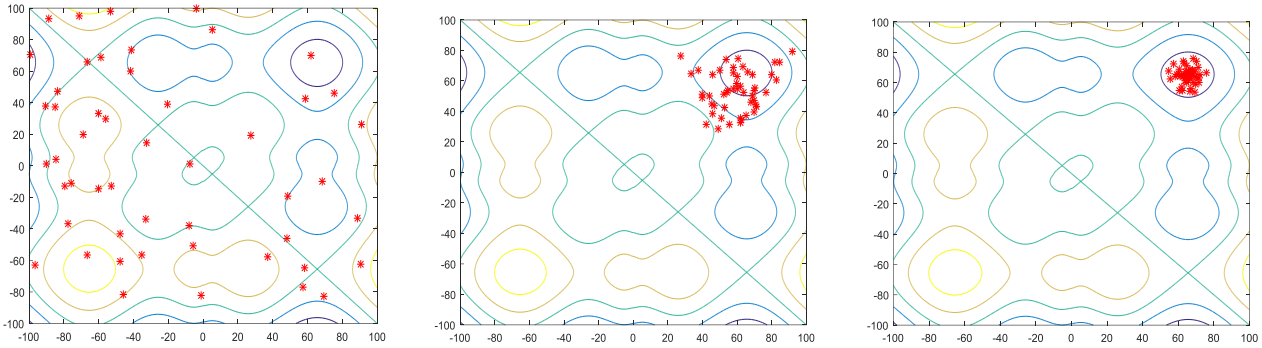
$$x(k) = Pg$$

- **Etape 6 :** Si un critère d'arrêt prédéfini est atteint, on affiche les résultats, sinon on revient à l'étape 2.

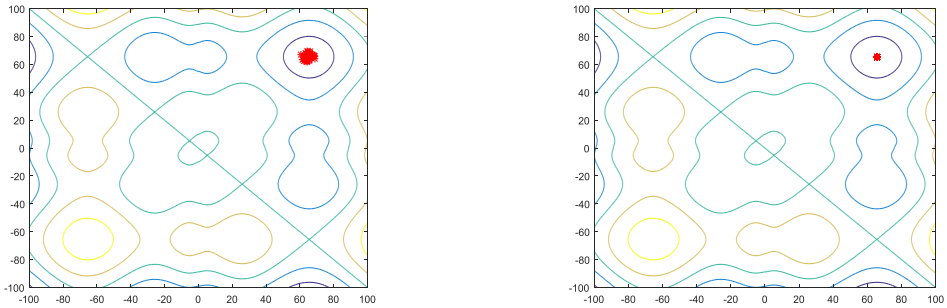
3.5.Représentation de la population

Les figures ci-dessous représentent une simulation du mouvement d'essaim des blattes dans un espace à deux dimensions. Le contour de la fonction objective est donné. On voit bien la convergence de l'essaim vers la région optimal de la fonction cout.

D'après les résultats obtenus dans la figure ci-dessous, on remarque qu'au début de la simulation nous trouvons tous les agents dispersés dans tout l'espace. Après 100 itérations, les agents sont toujours dispersés, mais sur une région restreinte. À l'itération 500, les agents commencent à se regrouper. Après 1000 itérations, les agents se regroupent de plus en plus dans une région précise (minimum global). A 10000 itérations, tous les agents sont presque superposés sur un seul point qui représente la solution au problème à résoudre.



Pour une itération | Pour 100 itérations | Pour 500 itérations



Pour 1000 itérations | Pour 10000 itérations

Figure II. 5- Simulation du mouvement d'essaim de blattes

3.6. Variantes

3.6.1. Roach Infestation Optimization :

RIO a été introduit à l'origine par Haven et al. [30] en tant qu'algorithme inspiré des blattes. RIO a été adapté à partir de l'algorithme PSO traditionnel et comporte donc des éléments similaires à PSO. Lorsqu'un agent de blattes rencontre un autre agent, il s'arrête, socialise et partage des informations sur l'emplacement le plus sombre connu [31]. Quand une blatte a faim, elle laisse ses camarades et son abri et part à la recherche de la nourriture [32]. Un compteur de faim est défini pour chaque agent de blatte, une fois atteint, la blatte est conduite vers une source de nourriture qui est placée au hasard dans la région de recherche. Le comportement de recherche d'aliments provoque la diversité de la population [30].

RIO est décrit avec trois comportements [30]:

- **Trouvé une zone sombre** : «Les blattes recherchent la zone la plus sombre dans l'espace de recherche.

$$v_i = c_0 v_i + c_{max} R_1 (p_i - x_i) \quad \text{(II.4)}$$

Où v_i est la vitesse du i^{ème} agent, x_i est l'emplacement actuel trouvé par le i^{ème} agent, p_i est le meilleur emplacement trouvé par i^{ème} agent, c_0; c_{max} sont des paramètres de blattes, R_1 est un vecteur de nombres aléatoires uniformes et (p_i - x_i) est un changement de vitesse dans la direction de la position la plus sombre connue de cet agent.

- **Recherche d'informations:** Si une blatte arrive dans un rayon de détection d'un autre agent, elle va s'arrêter, socialiser et partager des informations sur l'endroit le plus sombre connu en définissant son emplacement local *l* :

$$li = lj = arg_k min\{F(pk)\} \ , \ k = \{i,j\} \quad \text{(II.5)}$$

Où i, j sont les indices des deux blattes en train de socialiser et P_k est l'endroit le plus sombre.

$$v_i = c_0 \, v_i c_{max} \, . R1(p_i - x_i) + c_{max} \, .\mathrm{R2}(l_i - x_i) \quad \text{(II.6)}$$

$$x_i = x_i + v_i \quad \text{(II.7)}$$

- **Trouver de la nourriture**: Lorsqu'une blatte ressent la faim, elle quitte son abri se met à chercher de la nourriture. Elle est ensuite conduite vers une source de nourriture b placée au hasard dans la région de recherche.

$$x_i = b \quad \text{(II.8)}$$

3.6.2. Binary Cockroach Swarm Optimization :

L'optimisation binaire d'essaim de blattes (BCSO) a été conçue pour prendre en charge certains problèmes nécessitant une décision binaire. La probabilité de décision de la blatte, telle que, oui ou non, vrai ou faux, est prise en compte. Dans l'algorithme BCSO, la position de la blatte est définie en termes de probabilité qu'un bit soit à l'état zéro ou un. Les positions x_i et best personnel (p_i) sont des entiers de zéros ou de uns, $x_i \in \{0,1\}^D$, $p_i \in \{0,1\}^D$, où *D* est la taille de la dimension de la population de blattes.[33]

3.6.3. Improved Cockroach Swarm : [34]

Le CSOA original a été modélisé avec trois composants: aller en essaim, dispersion et le comportement brutal. Dans cet algorithme une nouvelle composante « faim » est introduite dans l'algorithme afin d'améliorer ses capacités de recherche et la diversité de sa population.

- **Comportement de chasse-essaimage:**

En utilisant l'équation (II.1), où step est une valeur fixe, rand est un nombre aléatoire compris entre [0, 1], p_i la meilleure position personnelle et p_g la meilleure position globale.

$$\mathbf{p(i)} = \mathrm{opt}_j\{x(j) \,\|\, x(i)-x(j) \,\| \leq \mathrm{visual}\} \quad \text{(II.9)}$$

Où la distance visuelle de perception est une constante, *j*= 1, 2,. . . ,N *i*= 1, 2,. . . N,

$$\mathbf{pg}=\text{opt}_i\{x(i)\} \quad \text{(II.10)}$$

- **Comportement de la faim** : Si hunger==t_{hunger} :

$$x(i) = x(i) + (x(i) - ct) + x_{food} \quad \text{(II.11)}$$

Où x_i indique la position de la blatte, (x_i-c_t) indique la migration de la blatte par rapport à sa position actuelle, c'est une constante qui contrôle la vitesse de migration dans le temps t, x_{food} indique l'emplacement de la nourriture, t_{hinger} indique le seuil de la faim, et la faim est un nombre aléatoire compris entre [0, 1].

- **Comportement de dispersion :**

$$x(i) = x(i) + rand(1, D), \quad i=1,2\ldots.. \quad \text{(II.12)}$$

Où rand (1, D) est un vecteur aléatoire à D-dimensions pouvant être défini dans une certaine plage.

- **Comportement brutal :**

$$x_k=p_g \quad \text{(II.13)}$$

Où *K* est un entier aléatoire dans [1,N], et p_g représente la meilleure position globale.

L'algorithme ICSO présenter dans la figure se résume en ces étapes ci-dessous :

- Initialiser les essaims de blattes avec des nombres aléatoires uniformes et définir tous les paramètres.
- Trouver p_i et p_g en utilisant (II.9) et (II.10).
- Effectuer une chasse en essaim en utilisant (II.1).
- Effectuer le comportement de la faim en utilisant (II.11).
- Effectuer un comportement de dispersion en utilisant (II.12).
- Effectuer un comportement brutal en utilisant (II.13).
- Répétez la boucle jusqu'à ce que le critère d'arrêt soit atteint.

Algorithm : IMPROVED COCKROACH SWARM OPTIMIZATION

1. INPUT ; Fitness function : $f(x), x \in \mathbb{R}^D$
2. *Set parameters and generate an initial population of cockroach*
3. Set $p_g = x_1$
4. **For** $i=2$ *to* N **do**
5. **If** $f(x_i) < f(p_g)$ **then**
6. $p_g = x_i$
7. **end if**
8. **end for**
9. **for** $t=1$ *to* T_{max} **do**
10. **for** $i=1$ *to* N **do**
11. **for** $j=1$ *to* N **do**
12. **if** $abs(x_i - x_j) < visual$; $f(x_j) < f(x_i)$ **then**
13. $p_i = x_i$
14. **end if**
15. **end for**
16. **if** $p_i == x_i$ **then**
17. $x_i = w.x_i + step * rand * (p_g - x_i)$
18. **else**
19. $x_i = w.x_i + step * rand * (p_i - x_i)$
20. **end if**
21. **if** $f(x_i) < f(p_g)$ **then**
22. $p_g = x_i$
23. **end if**
24. **end for**
25. **if** Hunger==t_{hunger} **then**
26. $x_i = x_i + (x_i - ct) + x_{food}$
27. $hunger_i = 0$
28. *Increment hunger counters*
29. **end if**
30. **for** $i=1$ *to* N **do**
31. $x_i = x_i + rand(1,D)$
32. **if** $f(x_i) < f(p_g)$ **then**
33. $p_g = x_i$
34. **end if**
35. **end for**
36. $k = randint([1,N])$
37. $x_k = p_g$
38. **end for**

Figure II. 6 -An improved cockroach swarm optimization algorithm [34]

3.7. Application

L'application des CSO dans la littérature, mentionnons: la résolution des fonctions de référence [26]; Cheng et coll. ont appliqué le CSO au problème de vendeur itinérant (TSP) [27]; et ZhaoHui et HaiYan ont appliqué le CSO au problème de routage des véhicules [28]. En outre, Obagbuwa et coll. ont amélioré la capacité des CSO à résoudre des problèmes de fonction spatiale par l'introduction du facteur de constriction stochastique, ce qui permet l'algorithme à éviter l'explosion dans les régions en dehors de l'espace de recherche.

3.7.1. Problème du vendeur itinérant [35]

L'algorithme d'optimisation de l'essaim de blattes (CSOA) est utilisé pour résoudre le problème du vendeur itinérant (TSP). Dans CSOA, une série de comportements biologiques de blatte sont simulés tels que le regroupement vivant et la recherche de nourriture, le déplacement-nid et ainsi de suite. Les blattes rampent et recherchent une solution optimale dans l'espace de solution, ils ont supposé que la solution est recherchée comme la nourriture. Quand ils ont utilisé l'algorithme de CSOA pour optimiser le problème de TSP, premièrement, toutes les blattes sont initialisées avec les solutions aléatoires, puis prennent les formules pour l'évolution jusqu'à ce que la règle terminale arrive. L'ensemble de la procédure du CSOA optimisant le problème TSP peut être décrit comme suit:

Étape 1: initialiser l'essaim et les paramètres de CSOA; la taille de la population est définie comme m; la taille de la nourriture est définie comme *n*. *LOF* représente l'aliment optimal.

Étape 2: toutes les blattes rampent vers les aliments

```
FOR (int i = 1; i < = n; i + +)
FOR (int i = 1; i < = m; i + +)
```

{Évaluez la solution qui est générée quand la blatte rampe vers la nourriture}

Si (solution est meilleure que LOF)

{Mettre à jour LOF avec la solution}

-La blatte se déplace vers la meilleure position

Étape 3: toutes les blattes rampent vers LOF

```
FOR (int i = 1; i < = n; i + +)
```

{Évaluez la solution qui est générée quand la blatte rampe à LOF}

Si (solution est meilleure que LOF)

{Mettre à jour LOF avec la solution}

- La blatte se déplace vers la meilleure position

Étape 4: si la règle de terminal est satisfaite, arrêtez l'itération et sortez les résultats, sinon passez à l'étape 2.

4. Conclusion

Au cours des dernières années, diverses méthodes d'optimisation heuristique ont été développées. Certains de ces algorithmes sont inspirés par les comportements d'essaims dans la nature. Dans ce chapitre nous avons exposé un algorithme d'optimisation nommé essaim des blattes (CSOA). CSOA est inspiré du comportement des blattes à la recherche de la nourriture, mais ce dernier présente l'inconvenant du stockage d'informations élevées. Dans le chapitre suivant, nous allons présenter une nouvelle variante d'algorithme CSOA basé sur la notion du compactage de l'information. Ce nouvel algorithme est appelé algorithme d'essaim de blattes compact (compact cockroach swarm optimisation algorithm).

Chapitre III

Algorithme d’Essaim de Blattes Compact

Chapitre III

Algorithme d'essaim de blattes compact

1. Introduction

Nous avons présenté dans les chapitres précédents les algorithmes évolutionnaires et les algorithmes de la swarm intelligence. Ces approches nécessitent un calculateur puissant pour résoudre un problème donné. Le principal problème avec ces algorithmes est la taille de la population qui grandit avec le problème à résoudre. Ceci est le cas de l'algorithme CSOA. Dans ce chapitre, nous allons présenter une solution à ce problème dans le cadre de l'algorithme CSOA. L'algorithme obtenu fait partie des algorithmes de l'Intelligence des Essaims Compactés ou compact swarm intelligence (cSI) en anglais. Dans ce qui suit, nous allons présenter la stratégie compacte et notre algorithme compact Cockroach Swarm Algorithm (cCSOA). Le Toolbox de IEEE issu de congrès international pour le calcul évolutionnaire de 2014 (CEC2014), largement utilisé par la communauté scientifique, est utilisé dans ce chapitre afin d'évaluer les performances de notre approche. Les résultats obtenus montrent que le cCSOA est très compétitif comparé à la version originale du CSOA.

2. Représentation compacte

Les algorithmes compacts sont des algorithmes d'optimisation appartenant à la classe des algorithmes à estimation de la distribution (EDAs) [36]. Les algorithmes compacts emploient la même logique (ou modèle) de recherche des algorithmes basés sur la population, mais ils ne stockent pas et ne traitent pas une population entière. Ils font usage d'une représentation probabiliste de la population afin d'effectuer le processus d'optimisation. Cette représentation probabiliste simule le comportement de la population. Elle explore abondamment l'espace décisionnel au début du processus d'optimisation et concentre progressivement la recherche sur les candidats les plus prometteurs et rétrécit le rayon de recherche. De cette façon, une quantité beaucoup plus faible de paramètres est utilisée et stockée dans la mémoire. Ainsi, ces algorithmes nécessitent des périphériques de mémoire beaucoup plus limités par rapport aux algorithmes basés sur la population [37].

2.1. Algorithmes génétiques compacts

La toute première implémentation d'algorithme compact a été l'algorithme génétique compact (cGA) proposé par Harik et al. Le cGA simule le comportement d'un algorithme

génétique codé en binaire (GA). Le cGA présente des performances aussi bonnes que celles de GA. Et le plus important c'est que le cGA exige peu de mémoire et beaucoup moins de capacité [38]. Il est alors question d'une réduction significative de la complexité de l'algorithme.

Le cGA fait évoluer un vecteur de probabilité (PV) qui décrit la distribution virtuelle d'une population de solutions dans l'espace de recherche. Le cGA traite itérativement le PV avec des mécanismes de mise à jour qui imitent les opérations typiques de sélection et de recombinaison effectuées dans un GA classique (sGA) jusqu'à ce qu'un critère d'arrêt soit satisfait.

Les résultats obtenus par le cGA sur un certain nombre de problèmes de test ont montré que l'approche est presque équivalente à GA binaire utilisant une sélection par tournois et un croisement uniforme. La principale force du cGA est la réduction significative des besoins en mémoire, car il doit stocker seulement le PV au lieu d'une population entière de solutions. Ce dernier contient la probabilité de chaque bit d'être égal à zéro ou à un. Par conséquent, on dit que la population est compactée et représentée par le vecteur de probabilité (PV). Cette fonctionnalité le rend particulièrement adapté aux applications réelles où la mémoire est limitée [39] ou pour résoudre des problèmes combinatoires complexes [40].

À chaque génération, deux individus sont échantillonnés à l'aide de **PV**. La compétition entre ces deux individus déterminera la manière dont **PV** sera corrigé.

2.1.1. Algorithme génétique binaire compacté

Le cGA se compose d'un vecteur de probabilité qui représente la population (PV), celui-ci contient la probabilité de chaque bit d'être égal à zéro ou à un, on dit alors que la population est compactée et représenter par PV.

Au moyen du PV deux individus sont échantillonnés à chaque génération, la compétition entre ces deux individus déterminera la manière dont PV sera corrigé. Si la solution gagnante dans la correspondance avec son $j^{ème}$ gène affiche un 1 tandis que la solution de perdant affiche un 0, la composante $j^{ème}$ de PV sera augmentée d'une valeur de $1/ Np$. Où Np est la taille hypothétique de la population simulée. Au contraire, si la solution gagnante en correspondance avec son $j^{ème}$ gène affiche à 0 alors que la solution du perdant affiche à 1, la $j^{ème}$ composante de PV est réduite d'une quantité de $1 / Np$. Lorsque les gènes en position $j^{ème}$ affichent la même valeur pour les solutions gagnantes et perdantes, le PV ne sera pas modifié [37]. lepseudo-code décrivant les principes de fonctionnement de bcGA est affiché dans l'algorithme suivant :

Algorithme génétique binaire compact (bcGA)

```
1. N : the population size
2. L : chromosome length
3. Initialize the probability vector
4. For  i=1 : L
5.        P[i] =0.5
6. end  for
7. While (0 <  P[i] < 1  for i=1 : L)  do
8.            /* generate two individual a end  b */
9.            a=generate (p), b=generate (p)
10.          /* let them compete */
11.           [winner, loser]=compete(a,b)
12.           /* update the probabilityvector to ward the best one */
13.       For i=1 : L  do
14.                if winner[i] ≠ loser[i]  then
15.                   if winner[i] ==1 then
16.                      P[i]=P[i] +1/N
17.                else
18.                      P[i]=P[i] -1/N
19.                   end if
20.                end if
21.        end for
22. end while
23.  represent the final solution
```

Figure III.1-Algorithme génétique binaire compact (bcGA)

2.1.2. Algorithme Génétique Compact Réel [41]

Noté rcGA, cet algorithme est inspiré du cGA. Cette approche étend la représentation compacte au-delà du binaire et permet d'utiliser des valeurs réelles obtenant ainsi un algorithme d'optimisation avec des performances élevées malgré la quantité limitée de calcul et de mémoire employées.

Dans rcGA le PV n'est pas un vecteur, mais une matrice n × 2, où n est la dimension du problème.

$$PV^{\,t} = [\mu^t , \sigma^t] \qquad \text{(III.1)}$$

ici μ et σ sont, respectivement, des vecteurs contenant, pour chaque variable de décision, la moyenne et d'écart-type d'une fonction de distribution gaussienne (PDF, probability of density function) tronquée dans l'intervalle [− 1, 1]. La hauteur de la PDF est normalisée afin de maintenir sa superficie égale à 1.

Au début du processus d'optimisation, pour chaque variable de décision i, μ [i] = 0 et σ[i] = ʎ où ʎ est une grande constante positive (les scientifiques conseillent d'initialiser ʎ à 10). Cette initialisation des valeurs σ [i] est effectuée afin de simuler une distribution uniforme. Autrement dit, elle permet d'éviter de favoriser une sous-région de l'espace de recherche.

Par la suite, un individu est échantillonné comme élite. Un nouvel individu est généré et comparé à l'élite. Plus précisément, le mécanisme d'échantillonnage d'une variable de décision x [i], associée à une solution candidate x, de PV se fait en trois étapes. Comme mentionné ci-dessus, pour chaque variable de conception indexée par i, une PDF gaussienne tronquée caractérisée par une valeur moyenne μ [i] et un écart-type est associée. La formule mathématique de la PDF est donnée par l'équation suivante :

$$PDF(\mu[i],\sigma[i]) = \frac{e^{\frac{(x-\mu[i])^2}{2\sigma[i]^2}}\sqrt{\frac{2}{\pi}}}{\sigma[i]\left(\mathbf{erf}\left(\frac{\mu[i]+1}{\sqrt{2\sigma[i]}}\right)-\mathbf{erf}\left(\frac{\mu[i]-1}{\sqrt{2\sigma[i]}}\right)\right)} \quad \text{(III. 2)}$$

En utilisant cette représentation, la population est réduite à un vecteur de probabilité **PV** défini par l'équation(III.1)

Où erf représente la fonction erreur [42]. (La primitive de e^{-x^2})

Pour générer un individu de PV, nous générons de chaque fonction PV [i], une variable notée x[i]. On calcule d'abord à partir de la PDF, la fonction de distribution cumulative correspondante (CDF), le domaine de la CDF est [0,1]. Afin d'échantillonner la variable de conception x [i] de PV, un nombre aléatoire rand (0,1) est prélevé à partir d'une distribution uniforme [51]. La fonction inverse de la CDF, dans la correspondance de rand (0,1), est alors calculée. Cette dernière valeur est xr[i], doit être convertie à l'intervalle [a, b] par :

$$xr[i] = (b-a)\frac{xr(i)+1}{2} + a \quad \text{(III.3)}$$

A l'état initial, les moyennes de la densité gaussienne sont mises à zéro et les écarts-types sont mis à 10. Il reste maintenant à trouver un moyen d'ajuster PV en fonction des itérations. Pour ce faire, deux individus sont générés. Puis, un tournoi est effectué entre eux. Par conséquent, par rapport au critère d'optimisation (la fonction coût), un gagnant et un perdant sont obtenus. Selon leurs allèles (variables de décision) respectifs, une règle de mise à jour, qui

est obtenue en supposant que le perdant a été remplacé par le gagnant dans la population virtuelle représentée par PV, est ensuite effectuée en utilisant les formules suivantes [41]:
La règle de mise à jour de la moyenne μ est donnée par:

$$\mu^{t+1}[i] = \mu^{t}[i] + \frac{1}{Np}(w[i] - L[i]) \quad \text{(III.4)}$$

Où Np est la taille de la population virtuelle, w représente le gagnant et L le perdant.
La règle de mise à jour de σ est donnée par :

$$\sigma^{t+1}[i] = \sqrt{\sigma^{t}[i]^2 + \sigma^{t}[i]^2 - \sigma^{t+1}[i]^2 + \frac{1}{Np}(w[i]^2 - L[i]^2)} \quad \text{(III.5)}$$

3. Intelligence des essaims compactés

Depuis l'apparition de rcGA, plusieurs auteurs ont proposé la version compacte d'autres algorithmes d'optimisation à valeur réelle. En 2011, l'évolution différentielle compacte (cDE) a été proposée [43]. En 2012, la version compacte de PSO (cPSO) a également été proposée [44]. En 2013, Yang et al ont proposé un algorithme compact basé sur l'enseignement et l'apprentissage des élèves en classe appelé cTLBO [45]. En 2014, A.S. Soares *et al* ont proposé un algorithme génétique compact basé sur la mutation [46]. En 2016, B. V. Ha *et al* [47] ont proposé un nouvel algorithme génétique compact modifié qui fonctionne avec plusieurs vecteurs de probabilité.

L'intelligence des essaims compactés a été proposée par L. Tighzert *et al* pour réduire les capacités de calcul et de mémoire requises par les algorithmes basés sur des essaims [54]. Le principe consiste à faire interagir des candidats issus d'une densité de probabilité. Avant de présenter le principe de l'intelligence des essaims compactés, nous allons présenter une variante appelée campact Firefly Algorithm (cFA). D'autres algorithmes ont été proposés par L Tighzert et al [59] [60].

3.1. Algorithmes compacts inspirés des lucioles (cFAs)

Les lucioles (en anglais firefly) sont de petits coléoptères ailés capables de produire une lumière clignotante pour une attraction mutuelle. L'échange de ces signaux dans l'essaim permet au groupe de retrouver la nourriture, d'échapper aux prédateurs et de se reproduire (attraction sexuelle).

L'algorithme de luciole (FA) a été proposé par Yang (2008) [48], sa source d'inspiration est basée sur le comportement attractif mutuel entre les lucioles. Initialement, il a été développé pour résoudre les problèmes d'optimisation continus, mais plus tard il a été utilisé pour résoudre des problèmes discrets. Il a également été utilisé dans le domaine du traitement d'image numérique et leur compression [49].

Le FA admet les hypothèses suivantes [49], [50], [51] :

- Toutes les lucioles sont du même sexe, chaque luciole peut être attirée par l'autre indépendamment de son sexe.
- L'attraction d'une luciole est proportionnelle à sa luminosité. Les lucioles les moins lumineuses tendent vers le plus lumineux. Si le plus lumineux n'est pas proche, la luciole se déplace au hasard.
- La luminosité d'une luciole est déterminée par la fonction cout. Pour les problèmes de maximisation, la luminosité peut être supposée proportionnelle à la valeur de la fonction objective.

L'attractivité de la luciole est un facteur qui détermine la puissance de l'attraction des lucioles. Cela dépend de la distance entre les lucioles. L'attractivité est donnée par : [52].

$$\beta(r) = \beta_0 \, e^{-\gamma r^2} \tag{III.6}$$

Où β_0 dénote l'attractivité maximale (à r = 0) et γ est le coefficient d'absorption de la lumière, qui contrôle la diminution de l'intensité lumineuse dans l'environnement.

Luminosité: selon la distance entre deux lucioles et le coefficient d'absorption atmosphérique, un signal est associé à chacune. Ceci est formulé comme:[52]

$$I(r) = I_0 \, e^{-\gamma r^2} \tag{III.7}$$

Où *I* est l'intensité de la source lumineuse, ɣ est le coefficient d'absorption, r est la distance entre les deux lucioles et I_0 est l'intensité de la source lumineuse lorsque r = 0.

Mouvement : Pour l'ensemble de la population et pour chaque paire de lucioles, la luciole la moins apte x_j, suivant le critère étudié, est déplacée vers une luciole x_i plus apte qu'elle en utilisant le modèle suivant:[52]

$$x_j^{t+1} = x_j^t + \beta_0 \, e^{-\gamma r^2} \left(x_i^t - x_j^t\right) + \alpha \, randn(-0.5,0.5) \tag{III.8}$$

Où t est l'indice des itérations, α est le coefficient de mutation qui est généralement un paramètre adaptatif décroissant en fonction des itérations et randn (-0,5, 0,5) est un nombre aléatoire normal entre [-0,5, 0.5]. Le pseudo-code de FA est représenté sur la **figure (III.2).**

Firefly Algorithme

1. ProblemDefinition : Objective function *f(x)*
2. Generate an initial population of *n* fireflies
3. Define light absorption coefficient γ, I0 and β0
4. **while** termination condition == 0 **do**
5. **for** i = 1 : *N* **do /*** all fireflies */
6. **for** j = 1 : N (all n fireflies) (innerloop) **do**
7. **if** $((\boldsymbol{xi})>\boldsymbol{fitn}(\boldsymbol{xj}))$ **then**
8. Move firefly x*j* to wards firefly x*i* according to **Eq. III.8**
9. **end if**
10. Vary attractiveness with distance r using **Eq. III.6**
11. Evaluate new solutions and update light intensity.
12. **end for j**
13. **end for i**
14. Rank the fireflies and find the current global best g.
15. **end while**
16. Post process results and visualization

Figure III.2-Pseudocode de FA.

L'algorithme de luciole compact (cFA) algorithme a été proposé par L. Tighzert pour contrôler un robot humanoïde [53]. Il utilise un essaim de lucioles (insecte lumineuse) compact. L'idée de l'algorithme est d'utiliser une PDF pour représenter l'essaim. Puis de générer des lucioles et les faire interagir comme elles interagissent naturellement.

L'algorithme a 6 étapes [54] :

1- Initialisation :

Initialiser les moyennes des PDF à 0 et sa déviation standard à 10.

Initialiser les paramètres α, I_0, β_0.

2- Adaptation de la PDF

En fonction des gènes (des allèles ou des variables de décision) de ces deux individus, on peut alors adapter la moyenne et la déviation standard de la PDF. Les formules obtenues ont :

$$\mu^{t+1}[i] = \mu^{t}[i] + \frac{1}{Np}(w[i] - L[i]) \quad \text{(III.9)}$$

$$\sigma^{t+1}[i] = \sqrt{\sigma^{t}[i]^2 + \sigma^{t}[i]^2 - \sigma^{t+1}[i]^2 + \frac{1}{Np}(w[i]^2 - L[i]^2)} \quad \text{(III.10)}$$

3- Tournoi :

Pour adapter la PDF, on génère deux individus et on les compare. On obtient un gagnant (winner) et perdant (looser). Le winner est noté **w** et le loser est noté ***l.***

4- Attraction :

Le winner attire le loser par le modèle décrit dans l'équation suivante :

$$x_j^{t+1} = x_j^t + \beta_0\, e^{-\gamma r^2}\left(x_i^t - x_j^t\right) + \alpha\, randn(-0.5,0.5) \qquad \text{(III.11)}$$

t est l'indice d'itération.

Le winner et le loser sont ensuite attirés par la meilleure luciole trouvée. Suivant le même modèle d'attraction suivant l'équation **III.8**

5- Evaluation :

On évalue les nouvelles positions des lucioles. Si une luciole est meilleure que la meilleure luciole elle la remplace.

6- Condition d'arrêt :

Si la condition d'arrêt n'est pas satisfaite, aller à l'étape 2.Sinon : retourner la meilleur solution.

3.2. Pseudocode général du compact swarm intelligence

Un pseudo-code général de cSI est donné dans la **Figure III.3**. Comme l'essaim est compacté, le terme de Virtual Swarm (VS)**,** comme PV dans les cEAs, contient les paramètres de la représentation statistique de l'essaim. Comme dans le SI, un modèle d'interaction bio-inspiré entre les agents doit être défini. Un algorithme de la cSI peut être résumé en cinq étapes. Tout d'abord, nous initialiser VS, définir le modèle d'interaction et d'initialiser ses paramètres. Deuxièmement, un mécanisme d'échantillonnage est utilisé pour générer N_i particules. Troisièmement, les particules sont déplacées à l'aide du modèle d'interaction défini. Quatre, nous effectuons un tournoi entre les particules. Cinq, nous mettons à jour la représentation de VS [53]

Algorithme général de l'intelligence des essaims compactés [59]

1. Initialise parameter of the *virtual swarm (**VS**)*,
2. Define the swarm interaction model and initialise its parameters.
3. while *Termination Condition*
4. Generate **Ni** agents from &***VS***
5. Perform a pre-defined interaction between the generated agents
6. Perform a tournament
7. Update ***VS*** representation.
8. end while
9. return the ***best*** solution

Figure III.3- Pseudocode générale de l'algorithme de l'essaim compacté.

Le grand intérêt pour l'optimisation compacte est dû à trois faits. Tout d'abord, elle donne des résultats satisfaisants. Deuxièmement, elle ne compromet pas les coûts de mémoire et de calculs. Troisièmement, elle est très facile à mettre en œuvre. Par conséquent, les algorithmes d'optimisation compacts conviennent très bien à la mise en œuvre matérielle en raison de leurs besoins en mémoire réduits.

4. Compacte cockroach swarm optimization algorithm (cCSOA)

4.1. Principe

Comme les autres algorithmes basés sur la population, CSOA présente un inconvénient en ce qui concerne ses exigences élevées en capacité de calculs quand la dimension du problème est assez grande.

Notre objectif et de proposer une nouvelle variante de l'algorithme d'optimisation d'essaim des blattes (cockroach swarm optimization algorithm). L'algorithme proposé est appelé *compact cockroach swarm optimization algorithm (*cCSOA*)*. Le premier objectif de ce travail est de réduire les coûts et la taille de la mémoire requise par la population. L'algorithme cCSOA utilise une fonction de distribution pour représenter les agents connus (la population).

L'idée de base consiste à représenter l'essaim des blattes par une densité de probabilité (PDF). Notre proposition (cCSOA) représente chaque variable décisionnelle i du problème de dimension D à résoudre, par une fonction probabiliste. Une densité de probabilité normale, où gaussienne, définie dans l'intervalle [-1,1] est donné par la formule **III.2**

En utilisant cette représentation, la population est réduite à un vecteur de probabilité **PV** défini par l'équation **III.1.**

$$PV^{\,t} = [\mu^t \sigma^t]$$

Où *t* est l'indice de l'itération. La dimension de PV est de 2 ×*D*. Ainsi, à la place d'une population de *N*×*D*, avec *N* individu, nous avons sa représentation compacte dans PV. Nous représentons sur la figure suivante une densité de probabilité gaussienne (PDF) est sa fonction cumulative CDF.

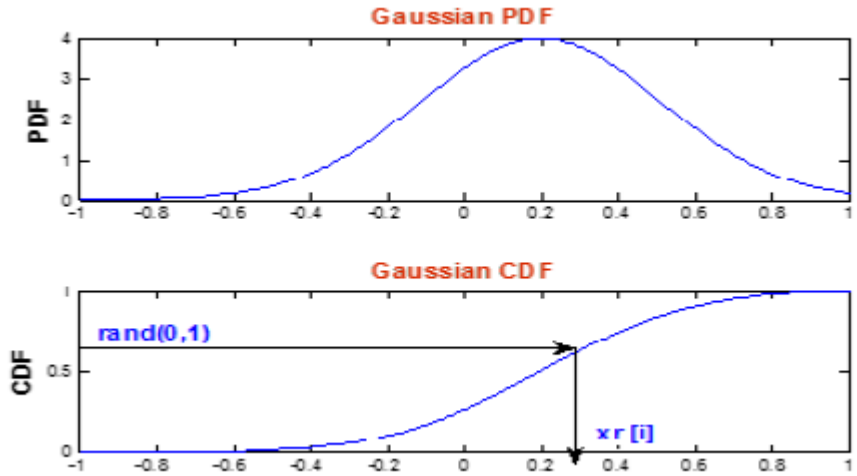


Figure III.4. Représentation graphique d'une PDF normale et de sa CDF

Pour générer un candidat de PV, nous générons de chaque fonction PV[i] la $i^{ème}$ variable notée x[i].Pour ce qui est de ce mécanisme d'échantillonnage (génération d'un individu à partir de PV), on calcule d'abord la fonction de distribution cumulative (CDF) correspondante à la PDF. Comme le domaine de CDF est [0,1], afin d'échantillonner une variable x[i] de PV, un nombre aléatoire uniforme rand (0,1) est généré, la fonction inverse de CDF correspondante à rand (0,1), est ensuite calculée. Cette valeur, notée xr [i], doit être convertir à l'intervalle [a, b] par l'équation **III.3**.

A l'état initial, les moyennes de la densité gaussienne sont mises à zéro et les écart-type sont mis à 10. Il reste maintenant à trouver un moyen d'ajuster PV en fonction des itérations. Pour ce faire, deux candidats sont générés. Puis, un tournoi est effectué entre eux. Par conséquent, par apport au critère d'optimisation, un gagnant et un perdant sont obtenus. Selon leurs allèles respectifs, une règle de mises à jour, qui est obtenue en supposant que le perdant a été remplacé par le gagnant dans la population virtuelle représentée par PV, est ensuite effectuée en utilisant les formules **III.4** et **III.5**.

On peut résumer l'algorithme cCSOA par les étapes suivantes :

1. Génération de la population virtuelle : cette étape permet à l'algorithme de générer une population représentée par une fonction de probabilité.

2. Générer un individu et supposer que c'est le meilleur (Best).
3. L'algorithme génère deux individus (deux agents) durant chaque cycle, évalue la fitness des deux individus et mis à jour le Best, s'il est surpassé.
   $\sigma_i = 10$ et $\mu_i = 0$ pour i=1 : D. D est la dimension du problème à résoudre.
4. Compétition : l'algorithme utilise une sélection par tournoi. Il compare deux individus générés. Le meilleur d'entre ces deux est noté winner (gagnant) et le moins apte est noté loser (perdant).
5. À chaque fois qu'une solution meilleure est trouvée, l'algorithme ajuste la PV comme décrit dans selon les équations **III.4 et III.5.**
6. Mouvement des blattes : après la compétition, l'algorithme déplace le loser (perdant) vers le winner, le loser vers le best et le winner vers le best, par la loi donnée en **II.1** et puis évalue la fitness de ces trois derniers.
7. Mouvement de dispersion : chaque blatte individuelle sera dispersée aléatoirement dans le but de maintenir la diversité de l'essaim actuel.
8. Tournoi entre le gagnant et la blatte déplacée, un gagnant et un perdant sont obtenus puis on ajuste PV.
9. Un tournoi entre le meilleur et le gagnant, un gagnant et un perdant sont obtenus puis et ajuste la PV.
10. Actualiser les paramètres adaptatifs
11. Si le critère d'arrêt n'est pas satisfait, on revient à l'étape 3.

4.2. Pseudo-code

L'algorithme cCSOA peut être représenté comme suit :

Cockraoch compact algorithm

1. N : Virtual population size
2. D:Problem dimension
3. R : ray of vision
4. t=1 /* iteration index */
5. **for** j=1:D **do** /* Initialization of PV */
6. $\mu_i^t = 0$; $\sigma_i^t = \lambda$;
7. **end**
8. *Best = generate* (PV) */* *Assume an elite */*
9. **While** Termination condition **do** /* Main loop */
10. [G1, G2]**=generate** (PV(t))
11. [winner, loser] = *compete* (G1, G2)
12. Update the best
13. **for** i=1 : D **do** /* updating PV */
14. $\boldsymbol{\mu_i^{t+1} = \mu_i^t + \frac{1}{N} * (w_i^t - L_i^t)}$;
15. $\boldsymbol{[\sigma_i^{t+1}]^2=[\sigma_i^t]^2+[\mu_i^t]^2-[\mu_i^{t+1}]^2+1/N*([w_i^t]^2 - [L_i^t]^2)}$;
16. /* where li and wiare the genes of the loser and winner*/
17. **end for**
18. **Move** *the loser toward the winner using* ***(Eq II.1)***
19. **Move** *the loser toward the Best using* ***(Eq II.1)***
20. **Move** *the Winner toward the Best using* ***(Eq II.1)***
21. *Dispersing Behaviour using* ***(Eq II.2)***
22. [winner, loser] = ***compete***(winner, loser)
23. **for** i=1 : D **do** /* updating PV */
24. $\boldsymbol{\mu_i^{t+1} = \mu_i^t + \frac{1}{N} * (w_i^t - L_i^t)}$
25. $\boldsymbol{[\sigma_i^{t+1}]^2=[\sigma_i^t]^2+[\mu_i^t]^2-[\mu_i^{t+1}]^2+1/N*([w_i^t]^2 - [L_i^t]^2)}$;
26. /* where l_i and w_i are the genes of the loser and winner*/
27. **end for**
28. [winner, loser] = ***compete*** (Winner, *Best*)
29. **for** i=1 : D **do** /* updating PV */
30. $\boldsymbol{\mu_i^{t+1} = \mu_i^t + \frac{1}{N} * (w_i^t - L_i^t)}$
31. $\boldsymbol{[\sigma_i^{t+1}]^2=[\sigma_i^t]^2+[\mu_i^t]^2-[\mu_i^{t+1}]^2+1/N*([w_i^t]^2 - [L_i^t]^2)}$;
32. /* where l_i and w*i* are the genes of the loser and winner*/
33. **end for**
34. R=R× Rdump
35. t=t+5;
36. **end while**
37. **return** the best individual

Figure III.5. Pseudocode cCSOA

5. Etude expérimentale des performances du cCSOA proposé

5.1. Fonction de benchmark

Les performances de l'algorithme proposé ont été examinées à l'aide de la suite d'essai CEC'14 lors de la session spéciale et de la concurrence sur l'objectif unique de l'optimisation numérique des paramètres réels .Le Toolbox CEC'14se compose de 30 fonctions de référence, qui comprennent trois fonctions unimodale (f1-f3), treize fonctions multimodales simples (f4-f16), six fonctions hybrides (f17-f22), et huit fonctions composées (f23-f30).

Comme il est recommandé par IEEE, des expériences ont été réalisées pour toutes ces fonctions avec dimension (D) égales à 10, 30**.** Pour chaque fonction, le nombre maximum d'évaluations de fonctions est fixé à 10000*D. L'espace de recherche pour chaque fonction est de $[100\ 100]^D$. Chaque expérience se termine lorsque le MaxFES est atteint ou que la valeur d'erreur est inférieure à 1E-08. Les descriptions détaillées du Toolbox de IEEECEC14 et les critères d'évaluation sont donnés dans [55].

5.2. Algorithmes utilisés dans notre étude comparative

Les performances de notre algorithme proposé, le cCSOA, sont comparées à quatre algorithmes de référence. Ces algorithmes sont GA, DE, PSO et CSOA.

5.2.2. Présentation de chaque algorithme utilisé

Dans cette section, nous allons présenter les algorithmes utilisés dans notre comparative. Comme GA a été présenté au chapitre I, nous orientons le lecteur vers la section **3**. L'algorithme CSO a été aussi présenté au chapitre II. Les deux autres algorithmes sont présentés ci-dessus.

5.2.2.1.L'algorithme DE [56]

Les Algorithmes à évolution différentielle sont des algorithmes inspirés de la théorie de l'évolution des espèces proposée par Charles Darwin il y 150 ans. L'évolution différentielle ou (DE) est proposé pour la première fois en Allemagne par deux scientifiques : par Price et Storn. À l'origine, l'évolution différentielle était conçue pour les problèmes d'optimisation continus et sans contraintes. Cependant, ses extensions actuelles peuvent traiter les problèmes à variables mixtes et gèrent les contraintes non linéaires. Actuellement, un nombre important d'applications industrielles et scientifiques font appel à l'évolution différentielle.

L'algorithme de l'évolution différentielle (DE), est représenté par les étapes suivantes :

1- Initialisation

On initialise les paramètres de l'algorithme

On initialise la population dans l'intervalle de recherche $[min, max]^D$

Où min et max sont les limites de l'intervalle de recherche.

2- Mutation différentielle

Pour chaque individu *X*i de la population *X*, à l'itération t, un vecteur mutant dénoté par *X*mi est créé. Pour avoir ce vecteur *X*mi, il existe différentes formules pour réaliser la mutation en DE sa formule est donné par :

$$X_m^{t+1} = X_i^t + F^*(X_j^t - X_k^t) \quad \text{(III.12)}$$

F est un vecteur de dimension *D* généré aléatoirement, généralement entre 0 et 1.

$X_j^t et\ X_k^t$ Sont deux individus distincts choisis aléatoirement dans la population *X*. *t* est l'indice d'itération.

3- Croisement :

Dans cette étape, chaque individu X_i^t.est croisé avec son vecteur mutant X_{mi}^{t+1}.

Le croisement intervient avec une probabilité Pcr.

4- Evaluation et sélection

L'algorithme évalue la fitness (le cout) de chaque individu X_{mi}^{t+1}de la populationX_{mi}^t

Si la fitness de X_{mi}^{t+1} n'est pas meilleur que celle de X_i^talors l'algorithme garde X_i^tpour la prochaine génération.

5- Condition d'arrêt

L'algorithme vérifie la condition d'arrêt et si elle n'est pas satisfaite il revient l'étape 2.Sinon l'algorithme s'arrête et affiche la meilleure solution qu'il a pu trouver. La condition d'arrêt doit être fixée par l'utilisateur. Elle peut être le nombre de générations.

5.2.2.2. L'algorithme PSO [57]

L'optimisation des essaims de particules (PSO) est une métaheuristique inventée en 1995 par deux scientifiques: Russel Eberhart, ingénieur électricien, et James Kennedy, socio-psychologue. C'est un algorithme basé sur la population qui simule le mouvement d'un groupe d'oiseaux. Comme d'autres algorithmes de la famille de la swarm intelligence, PSO suppose que les individus, c'est-à-dire les particules, peuvent faire émerger des comportements complexes autoorganisés en suivant des règles d'interactions très simples (basiques). L'intelligence obtenue est inconnue au niveau de la particule. Le PSO définit chaque individu (particule ou oiseau) par sa vitesse, sa position et la meilleure position qu'elle connait. Chaque individu i corrige sa vitesse, Vi, et sa position, Xi, qui sont présentés par les formules mathématiques suivantes :

La règle de mise à jour de la vitesse est donnée par:

$$V_i^{k+1} = W\, V_i^k + C_1 R_1 \left(X_i^p - V_i^k\right) + C_2 R_2 \left(X_g - V_i^k\right) \quad \textbf{(III.13)}$$

La règle de mise à jour de la position est donnée par :

$$X_i^{k+1} = X_i^k + V_i^{k+1} \quad \textbf{(III.14)}$$

Les paramètres des algorithmes utilisés sont tirés de la littérature originale. Ils donnés dans le tableau suivant.

L'algorithme	PSO [58]	GA [4]	DE [56]	CSOA [34]
Paramètre	C1=c2=2 W=0.98	Pm=0.4 N=2*D Rm=0.2*(max-min)	Pcr=0.8 F=0.4	Step=2 VD=0.2*(max-min)

Tableau III.1- Paramètres des algorithmes comparatifs

6. Résultats pratiques et discussion

Les algorithmes proposés sont testés sur 30 fonctions, avec 10 puis 30 dimensions (D=10, D=30), tirées du CEC2014 [65], chaque fonction testée 51 fois. La valeur de la fonction d'erreur $(f(x^{best})-f(x^*))$de chaque exécution est enregistrée, où x^* est la solution optimale et x^{best}est la meilleure solution trouvée à la fin de chaque exécution. La moyenne et l'écarts-types des valeurs de la fonction d'erreur $(f(x^{best})-f(x^*))$, pour chaque fonction de test, sont désignées respectivement par «Mean» et «STD» et sont considérés pour évaluer les performances des algorithmes cités.

Toutes les expériences ont été exécutées dans Matlab 2016a sur un Intel® CORE i5 inside-62000U CPU @ 2.3GHz @ 2.4GHz, avec la RAM de 8 Go, système Windows 10, 64bits.

Tableau III.2

Résultats expérimentaux de cCSOA, CSOA, PSO, GA et DE sur 51 essais, sur 30 fonctions de test de dimension 10. "Means" et " St.Dev " indiquent la moyenne et l'écart-type de la valeur de la fonction d'erreur $f(x^{best})$-$f(x^*)$ obtenus, respectivement.

Fonctions		GA	DE	PSO	CSOA	cCOSA
F1	**Means**	2.8430e+09	2.6920e+07	1.9779e+07	**5.0181e+05**	6.1431e+05
	St.Dev.	1.1569e+08	1.5207e+07	6.4038e+06	**2.9792e+05**	5.5442e+05
F2	**Means**	1.2123e+10	5.7071e+05	9.8976e+08	1.6516e+05	**4.8811e+03**
	St.Dev.	3.6860e+08	4.9467e+05	3.1797e+08	3.5734e+04	**8.5071e+03**
F3	**Means**	1.4824e+04	2.3962e+03	5.8253e+03	**1.7585e+02**	2.1786e+03
	St.Dev.	1.1572e+03	1.6523e+03	2.7309e+03	**9.9226e+01**	9.3260e+02
F4	**Means**	7.5595e+03	1.0206e+01	1.7881e+02	2.2118e+01	**7.6717e+00**
	St.Dev.	3.0735e+02	2.2899e+01	7.6451e+01	1.6298e+01	**1.6865e+01**
F5	**Means**	2.0305e+01	2.0560e+01	2.0323e+01	2.0358e+01	**2.0000e+01**
	St.Dev.	1.6229e-01	1.2742e-01	8.6767e-02	6.9083e-02	**3.8121e-04**
F6	**Means**	1.0634e+01	1.0233e+01	7.7449e+00	3.0745e+00	**1.5291e+00**
	St.Dev.	4.1304e-01	1.0053e+00	7.9059e-01	1.6032e+00	**1.2147e+00**
F7	**Means**	2.8107e+02	6.9932e-01	3.3728e+01	8.6222e-01	**2.2991e-01**
	St.Dev.	6.2389e+00	1.2945e-01	1.1531e+01	1.8202e-01	**1.1161e-01**
F8	**Means**	7.1906e+01	2.5273e+01	5.5957e+01	2.3077e+01	**2.2241e+01**
	St.Dev.	7.6013e+00	6.4330e+00	6.3560e+00	9.8630e+00	**7.4910e+00**
F9	**Means**	6.6881e+01	4.8141e+01	4.6933e+01	2.3112e+01	**2.2770e+01**
	St.Dev.	7.6719e+00	9.6977e+00	5.6418e+00	8.3506e+00	**8.3795e+00**
F10	**Means**	1.3135e+03	1.4417e+03	7.0229e+02	8.0343e+02	**6.5022e+02**
	St.Dev.	1.6676e+02	2.3073e+02	1.4813e+02	2.9244e+02	**2.7083e+02**
F11	**Means**	1.6900e+03	2.1492e+03	9.8995e+02	7.8273e+02	**6.4370e+02**
	St.Dev.	2.6583e+02	2.9307e+02	1.8275e+02	3.3970e+02	**2.2548e+02**
F12	**Means**	9.2742e-01	1.6610e+00	9.1621e-01	4.4745e-01	**1.2037e-01**
	St.Dev.	4.4750e-01	4.8439e-01	3.1667e-01	1.7739e-01	**9.8813e-02**
F13	**Means**	6.5958e+00	5.4211e-01	8.3839e-01	**2.0933e-01**	2.7262e-01
	St.Dev.	1.2174e-01	1.2152e-01	2.3445e-01	**6.7904e-02**	1.0539e-01
F14	**Means**	4.5243e+01	3.5781e-01	2.6880e+00	**1.2977e-01**	2.4316e-01
	St.Dev.	1.1426e+00	1.0667e-01	1.4453e+00	**4.4786e-02**	9.5482e-02
F15	**Means**	1.3389e+04	4.5640e+00	7.6515e+00	2.4802e+00	**1.5484e+00**
	St.Dev.	2.5904e+03	9.7089e-01	1.7342e+00	7.5209e-01	**7.8476e-01**
F16	**Means**	4.3897e+00	4.2756e+00	3.4501e+00	**2.6189e+00**	2.8307e+00
	St.Dev.	8.6347e-02	1.6849e-01	2.1004e-01	**4.5407e-01**	2.6870e-01
F17	**Means**	5.6296e+05	7.5873e+04	1.8133e+04	3.2031e+03	**2.8559e+03**
	St.Dev.	4.5756e+04	1.0862e+05	1.5310e+04	2.1704e+03	**3.0250e+03**
F18	**Means**	1.6580e+05	3.1030e+04	9.2680e+03	8.9690e+03	**6.8310e+03**
	St.Dev.	3.3959e+05	4.1677e+04	2.5592e+03	7.9194e+03	**5.4480e+03**
F19	**Means**	2.4101e+02	4.4271e+00	6.0687e+00	2.8629e+00	**2.6905e+00**
	St.Dev.	2.8293e+01	1.8931e+00	8.3020e-01	9.6064e-01	**8.2239e-01**
F20	**Means**	1.1013e+05	1.2844e+03	5.3015e+03	**1.1712e+02**	4.6691e+02
	St.Dev.	4.4425e+05	5.5746e+03	2.7717e+03	**4.3721e+01**	4.0510e+02
F21	**Means**	6.6206e+08	2.2791e+03	6.6238e+03	**1.4753e+03**	2.3631e+03
	St.Dev.	8.3221e+07	1.6683e+03	2.2656e+03	**1.2959e+03**	1.4320e+03
F22	**Means**	5.8403e+02	**4.4276e+01**	1.6281e+02	8.3887e+01	8.1319e+01
	St.Dev.	9.0355e+01	**3.6678e+01**	9.6880e+00	6.3804e+01	5.2683e+01
F23	**Means**	2.4904e+02	3.3339e+02	**2.1117e+02**	3.2949e+02	3.2946e+02
	St.Dev.	1.5485e+01	1.4907e+01	**3.8278e+01**	1.6940e-02	6.8189e-13
F24	**Means**	2.0240e+02	1.5562e+02	1.7824e+02	1.3063e+02	**1.2872e+02**
	St.Dev.	6.7375e-01	8.8883e+00	1.7897e+01	1.0534e+01	**7.4655e+00**
F25	**Means**	2.0022e+02	2.0431e+02	1.9940e+02	1.8316e+02	**1.6210e+02**
	St.Dev.	1.2006e-01	1.9298e+00	1.3137e+00	2.7238e+01	**2.2995e+01**
F26	**Means**	2.0001e+02	1.0065e+02	1.0160e+02	**1.0020e+02**	1.0026e+02
	St.Dev.	5.5818e-03	1.5815e-01	5.0654e-01	**7.2047e-02**	1.0459e-01
F27	**Means**	5.0243e+02	5.6286e+02	4.2401e+02	3.2431e+02	**1.2980e+02**
	St.Dev.	1.6256e+02	5.5484e+01	1.4405e+02	1.4057e+02	**1.8217e+02**
F28	**Means**	4.2171e+02	**3.1584e+02**	8.3057e+02	4.9171e+02	3.2878e+02
	St.Dev.	1.1514e+02	**3.2538e+01**	5.2264e+02	1.0992e+02	4.0901e+01
F29	**Means**	4.3807e+07	2.7531e+02	7.4558e+06	2.4897e+05	**2.0632e+02**
	St.Dev.	1.4936e+07	5.1654e+01	1.4951e+07	7.8831e+05	**4.2970e+00**
F30	**Means**	7.9777e+06	**6.8460e+02**	2.8530e+03	1.5526e+03	6.9328e+02
	St.Dev.	2.4597e+06	**1.517e+02**	4.5287e+02	5.7523e+02	5.7241e+02

Tableau III.3

Résultats expérimentaux de cCSOA, CSOA, PSO, GA et DE sur 51 essais, sur 30 fonctions de test de dimension 30. "Means" et " St.Dev " indiquent la moyenne et l'écart-type de la valeur de la fonction d'erreur $f(x^{best})$-$f(x^*)$ obtenus, respectivement.

Fonctions		GA	DE	PSO	CSOA	cCOSA
F1	**Means**	1.7218e+09	1.7702e+09	5.2161e+08	**7.8373e+05**	1.1276e+07
	St.Dev.	6.7330e+07	5.8774e+08	1.2310e+08	**2.1780e+05**	8.7110e+06
F2	**Means**	8.4930e+10	5.8727e+04	4.2442e+10	4.6904e+04	**2.1774e+04**
	St.Dev.	1.5200e+09	4.5561e+04	3.0515e+09	1.1118e+04	**2.2228e+04**
F3	**Means**	8.2923e+04	4.8331e+04	6.1044e+04	**5.5328e-01**	1.9384e+04
	St.Dev.	3.0110e+03	1.9367e+04	5.7615e+03	**2.6040e-01**	2.0208e+03
F4	**Means**	1.9367e+04	**2.5026e+01**	3.2949e+03	3.7944e+01	3.6567e+01
	St.Dev.	5.5673e+02	**6.3636e-01**	6.2528e+02	3.4981e+01	2.2638e+01
F5	**Means**	2.0812e+01	2.1096e+01	2.0921e+01	2.0067e+01	**1.9999e+01**
	St.Dev.	1.5591e-01	4.5882e-02	6.7040e-02	2.9775e-02	**1.6819e-03**
F6	**Means**	4.3927e+01	4.2404e+01	3.5519e+01	2.0787e+01	**1.5847e+01**
	St.Dev.	7.0505e-01	1.5235e+00	1.6524e+00	3.9526e+00	**3.3037e+00**
F7	**Means**	8.4210e+02	**7.3188e-03**	3.8517e+02	9.0199e-02	1.3392e-02
	St.Dev.	1.3092e+01	**1.4565e-02**	2.8910e+01	1.8898e-02	1.1113e-02
F8	**Means**	2.9369e+02	1.7838e+02	2.9911e+02	1.7700e+02	**1.3153e+02**
	St.Dev.	1.8822e+01	1.8194e+01	1.6223e+01	4.4404e+01	**2.7652e+01**
F9	**Means**	3.0927e+02	2.5233e+02	2.8518e+02	1.8632e+02	**1.2926e+02**
	St.Dev.	1.8968e+01	1.7669e+01	1.4705e+01	6.3766e+01	**2.7803e+01**
F10	**Means**	6.2306e+03	8.0217e+03	6.0805e+03	4.0710e+03	**3.2120e+03**
	St.Dev.	3.5572e+02	4.2859e+02	3.0945e+02	7.9967e+02	**5.4785e+02**
F11	**Means**	7.3498e+03	9.1647e+03	6.6708e+03	4.2568e+03	**3.5123e+03**
	St.Dev.	5.6306e+02	4.4553e+02	5.8630e+02	6.0855e+02	**5.3875e+02**
F12	**Means**	1.3122e+00	3.3466e+00	2.0533e+00	**2.5507e-01**	3.8713e-01
	St.Dev.	3.9583e-01	4.7368e-01	3.8369e-01	**1.3011e-01**	1.8390e-01
F13	**Means**	9.6901e+00	8.0790e-01	5.4697e+00	**5.2607e-01**	5.3376e-01
	St.Dev.	1.3720e-01	1.3999e-01	2.1843e-01	**1.1221e-01**	8.2472e-02
F14	**Means**	3.1760e+02	5.9888e-01	1.4601e+02	**4.3423e-01**	5.5838e-01
	St.Dev.	5.4235e+00	2.3568e-01	1.2097e+01	**2.2309e-01**	3.4415e-01
F15	**Means**	1.9442e+05	2.2693e+01	2.0314e+04	**8.2511e+00**	8.7195e+00
	St.Dev.	1.9989e+04	2.4871e+00	7.0537e+03	**2.5739e+00**	2.4848e+00
F16	**Means**	1.3787e+01	1.4183e+01	1.2900e+01	1.2595e+01	**1.2280e+01**
	St.Dev.	1.5064e-01	1.8100e-01	2.3019e-01	4.4780e-01	**3.3422e-01**
F17	**Means**	5.0874e+08	6.9737e+07	2.3418e+07	**3.9729e+04**	5.9315e+05
	St.Dev.	3.3732e+07	2.4365e+07	1.1535e+07	**2.7306e+04**	3.9588e+05
F18	**Means**	1.1407e+10	1.3428e+07	1.8996e+08	**7.9773e+03**	1.1536e+04
	St.Dev.	3.8573e+08	9.4947e+06	4.4384e+07	**6.0208e+03**	1.6917e+04
F19	**Means**	5.6109e+02	2.4373e+01	1.5040e+02	**1.6570e+01**	1.7416e+01
	St.Dev.	1.6872e+01	1.3331e+01	2.3585e+01	**1.5779e+01**	2.6553e+00
F20	**Means**	4.6965e+08	1.2709e+06	4.5357e+04	**3.5295e+02**	6.9483e+03
	St.Dev.	9.9246e+07	9.1569e+05	2.1063e+04	**1.2250e+02**	2.7261e+03
F21	**Means**	1.0007e+09	1.9534e+07	4.1073e+06	**3.6011e+04**	3.1426e+05
	St.Dev.	8.1258e+07	7.8924e+06	3.6052e+06	**2.1425e+04**	2.3690e+05
F22	**Means**	1.5516e+06	1.2849e+03	1.1539e+03	5.4161e+02	**5.1788e+02**
	St.Dev.	1.7540e+05	2.1688e+02	1.8606e+02	1.8803e+02	**2.1621e+02**
F23	**Means**	4.1747e+02	3.2131e+02	**3.0187e+02**	3.1525e+02	3.2002e+02
	St.Dev.	3.3522e+01	2.4566e+01	**1.0660e+02**	8.1095e-04	4.5442e+00
F24	**Means**	2.1495e+02	2.3034e+02	**2.0000e+02**	2.5102e+02	2.3578e+02
	St.Dev.	1.9971e+00	7.7287e+00	**1.3319e-05**	4.2289e+00	9.3020e+00
F25	**Means**	2.0402e+02	3.5648e+02	**2.0000e+02**	2.1342e+02	2.1209e+02
	St.Dev.	5.0362e-01	4.3864e+01	**0**	5.5530e+00	2.9913e+00
F26	**Means**	2.0012e+02	**1.0079e+02**	1.8807e+02	1.6041e+02	1.8032e+02
	St.Dev.	5.3981e-02	**1.3530e-01**	2.8387e+01	7.3640e+01	3.9809e+01
F27	**Means**	3.0805e+03	1.4334e+03	1.4084e+03	8.6106e+02	**6.1278e+02**
	St.Dev.	7.5526e+02	4.7976e+01	3.3564e+02	2.3420e+02	**1.9256e+02**
F28	**Means**	3.3264e+03	5.3078e+02	4.8216e+03	1.7338e+03	**4.9055e+02**
	St.Dev.	8.2953e+02	1.6805e+02	3.8003e+03	6.3199e+02	**7.8491e+01**
F29	**Means**	2.5762e+08	4.4550e+02	2.5306e+08	1.4294e+06	**2.1965e+02**
	St.Dev.	3.1915e+07	2.0899e+02	2.0509e+08	3.9163e+06	**5.2905e+00**
F30	**Means**	1.9067e+07	4.9455e+03	1.7659e+06	5.4540e+03	**1.1420e+03**
	St.Dev.	1.7990e+06	2.6108e+03	5.7710e+05	2.7557e+03	**3.1487e+02**

6.1. Comparaison graphique des performances des algorithmes testés

Nous présentons ici les différentes allures de la progression de notre algorithme comparé aux algorithmes utilisés dans le test pour D=10

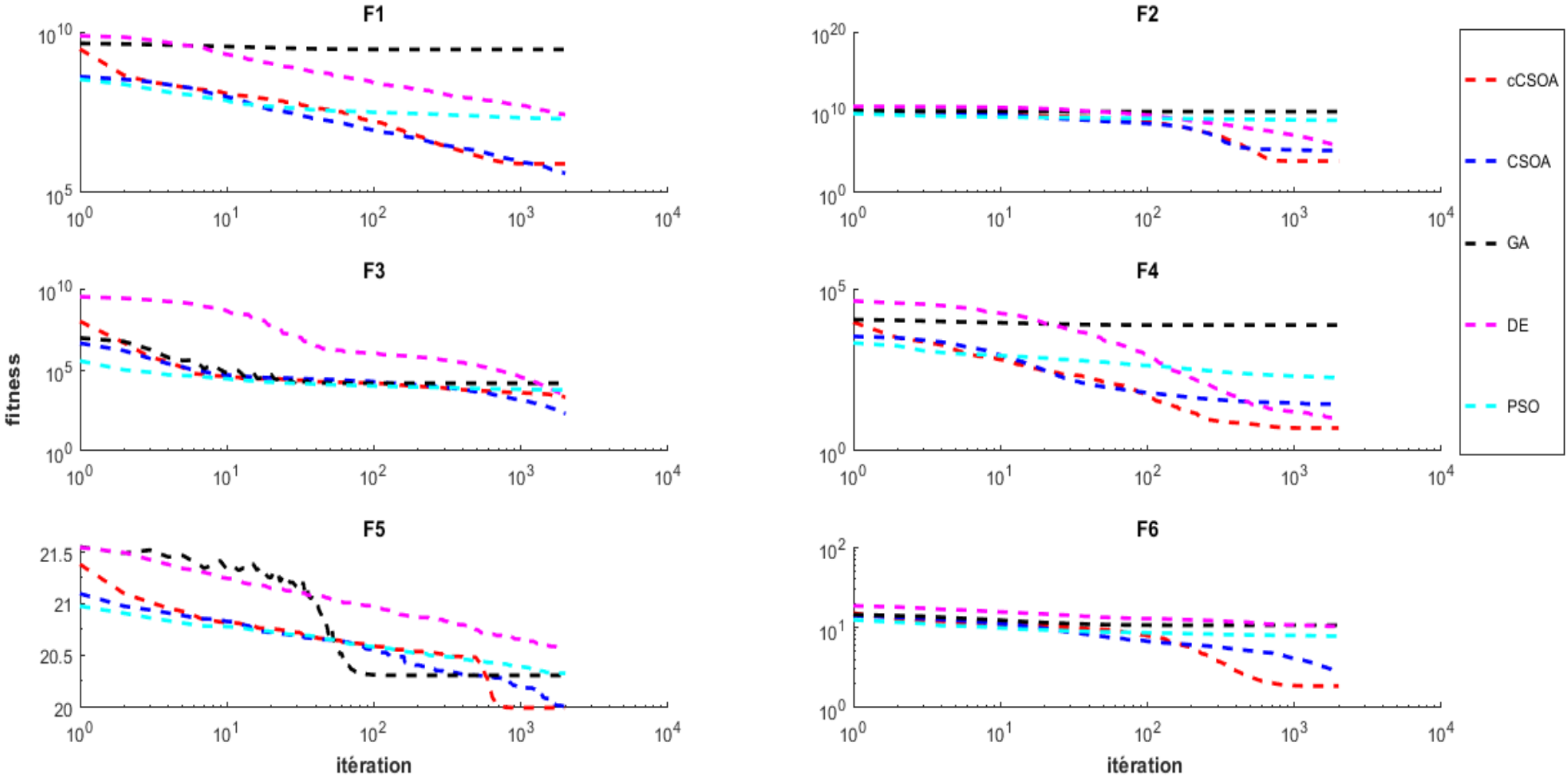


Figure III.6. Comparaison des performances des différents algorithmes de F1 jusqu'à F6

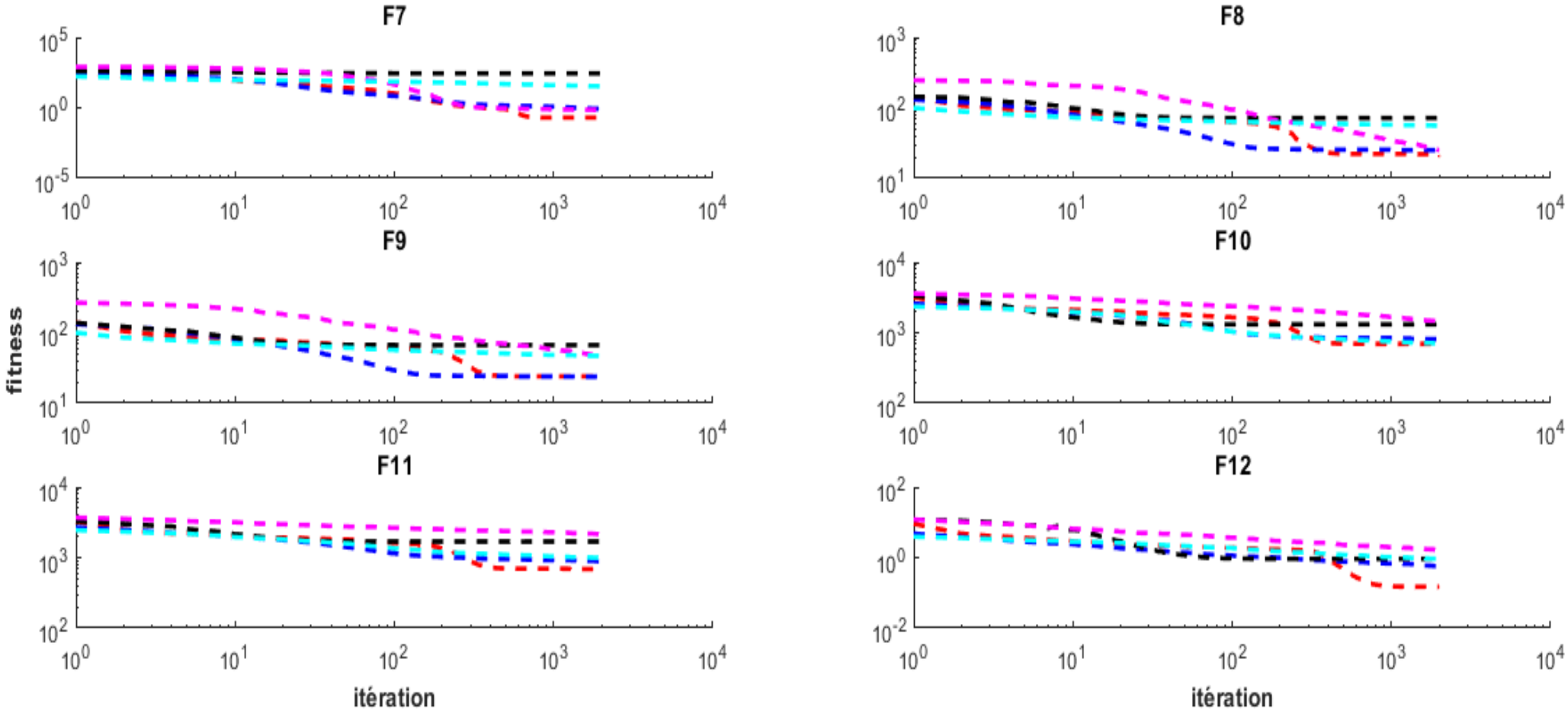


Figure III.7. Comparaison des performances des différents algorithmes de F7 jusqu'à F12

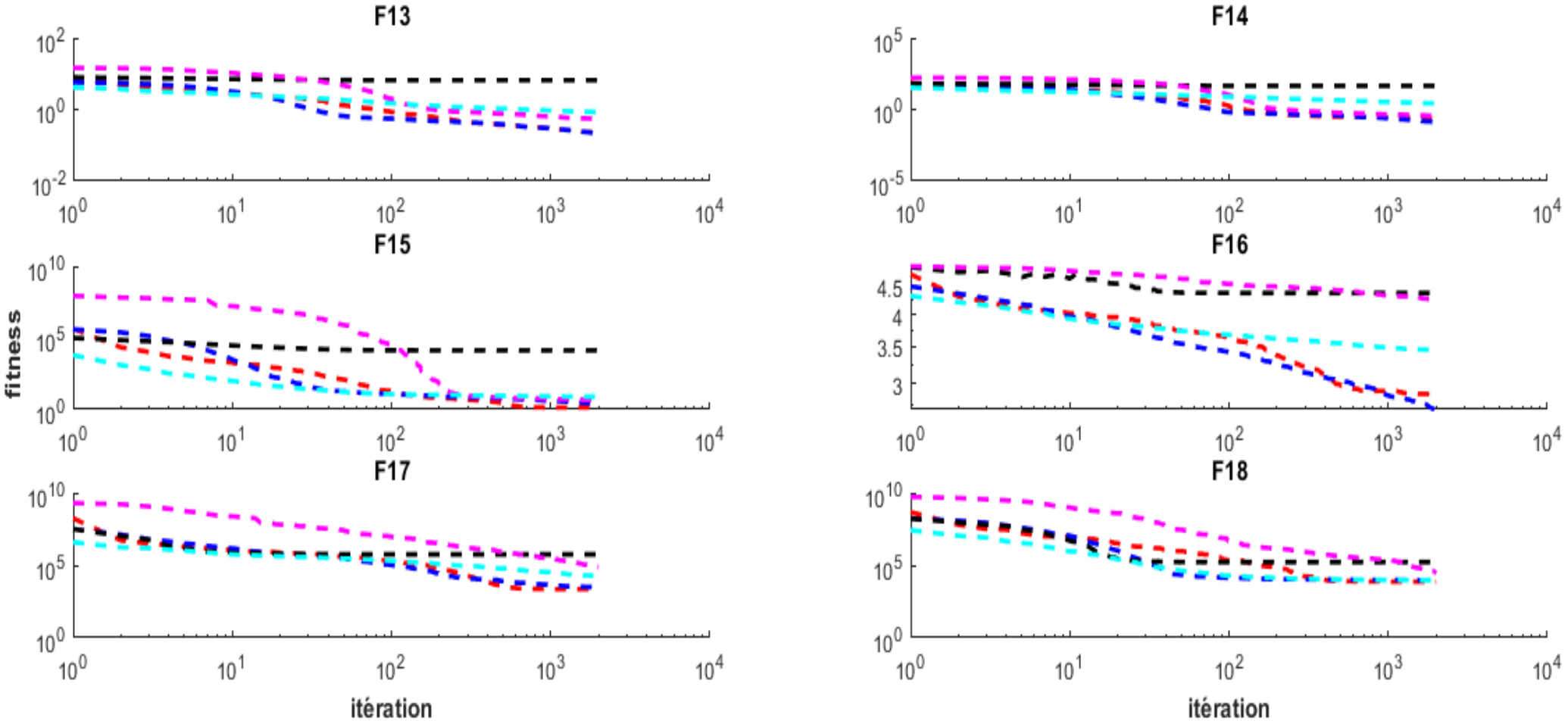


Figure III.8. Comparaison des performances des différents algorithmes de F13 jusqu'à F18

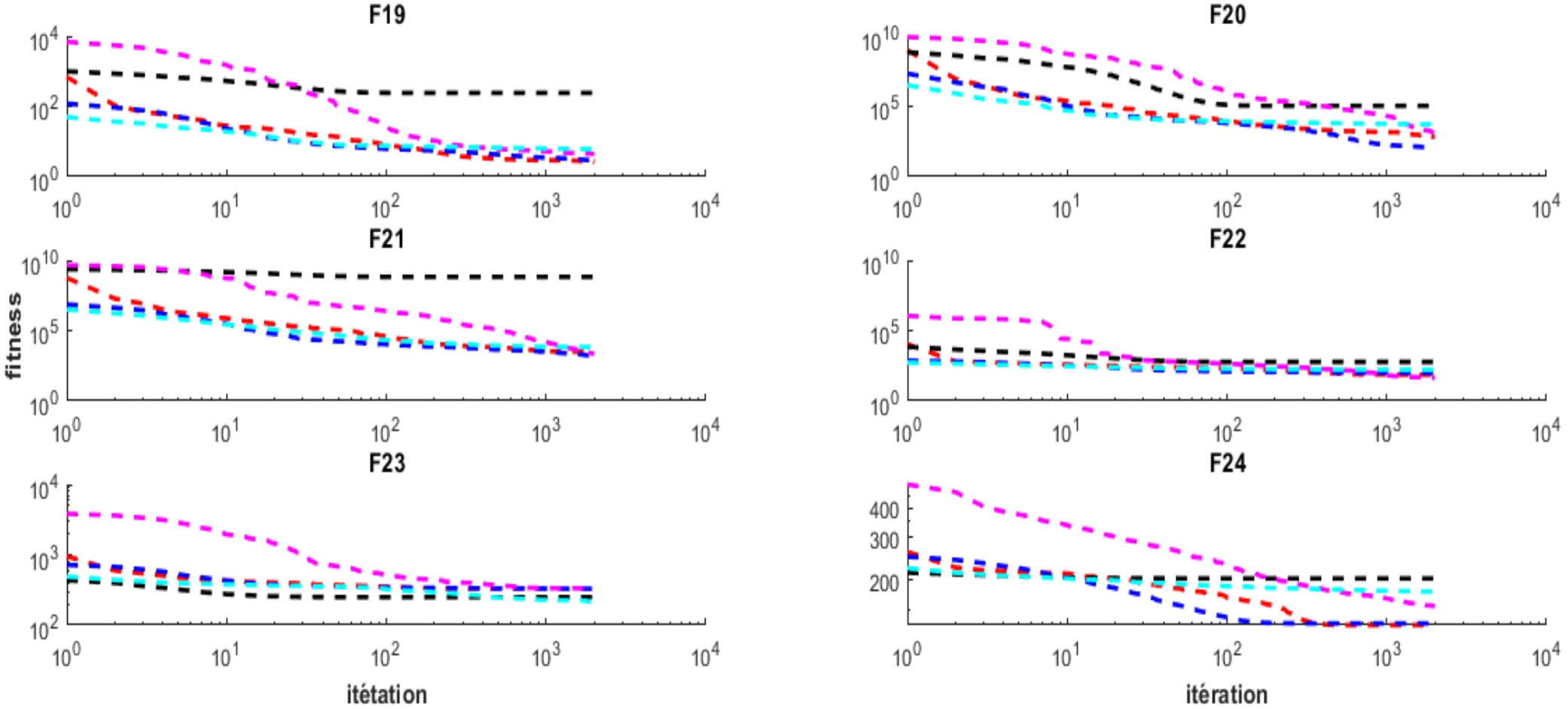


Figure III.9. Comparaison des performances des différents algorithmes de F19 jusqu'à F24

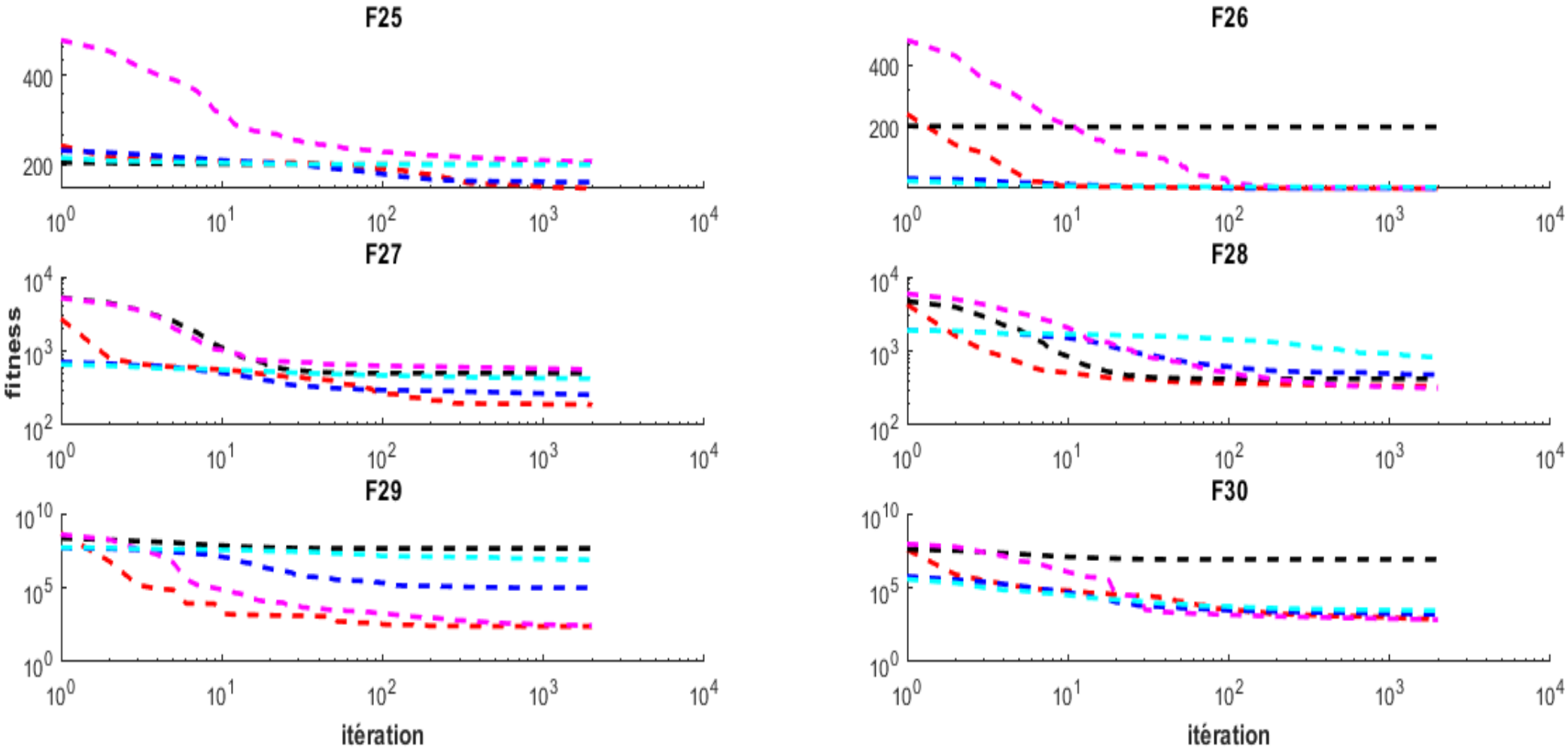


Figure III.10. Comparaison des performances des différents algorithmes de F25 jusqu'à F30

Tableau III.4 Evaluations des résultats du temps d'exécution du cCSOA et CSOA pour les 30 fonctions avec D = 30

	Temps d'exécution (secondes)		Taux de réduction du temps
Fonctions	CSOA	cCSOA	(%)
F1	18.6134	3.5773	80,78
F2	17.6808	2.8372	83,95
F3	17.6953	2.7945	84,20
F4	17.7401	2.7663	84,40
F5	18.2263	3.3226	81,77
F6	56.8108	43.2747	23,82
F7	18.2491	3.3887	81,43
F8	17.5963	2.6972	84,67
F9	18.1524	3.2590	82,04
F10	17.9815	3.0856	82,83
F11	18.5563	3.6683	80,23
F12	28.0290	13.5019	51,82
F13	17.7017	2.8059	84,14
F14	17.6271	2.7556	84,36
F15	18.1906	3.3554	81,55
F16	18.2372	3.3448	81,65
F17	18.4163	3.5764	80,58
F18	18.3912	3.0083	83,64
F9	29.6030	11.2447	62,01
F20	20.5983	3.1179	84,86
F21	21.6082	3.3060	84,70
F22	22.1795	4.4770	79,81
F23	25.4060	6.6904	73,66
F24	23.2304	5.0033	78,46
F25	24.6681	6.5128	73,59
F26	62.0825	48.2973	22,20
F27	61.6334	48.1542	21,86
F28	23.3230	8.5032	63,54
F29	28.3679	13.6892	51,74
F30	21.6420	6.7517	68,80
moyenne	24.6079	9.0922	63.05

6.2.Comparaison entre cCSOA et GA

Les résultats du cCSOA sont plus performants que ceux de GA et cela pour toutes les fonctions, soit pour D=10 ou D=30.

6.3..Comparaison entre cCSOA, DE et PSO

La comparaison entre les deux algorithmes montre que les résultats du cCSOA dépassent largement ceux du DE et du PSO, pour D=10 et D=30. Il y a quelques rares cas ou DE et PSO montrent des résultats plus performants. Exemple : dans le cas de DE, D=10, on a **F22**, **F28** et **F30.**

6.4.. Comparaison entre cCSOA vs CSOA

Les deux tableaux montrent bien que les résultats donnés par cCSOA surpassent dans plus de la moitié des cas les résultats affichés par l'algorithme original CSOA, pour les autres problèmes de test, cCSOA affiche des performances comparables et compétitives.

6.5.. Temps d'exécution pour cCSOA et CSOA

Le temps que prend un algorithme pour renvoyer la solution optimale est l'un des facteurs les plus importants. Dans le **Tableau III.4** nous donnons le temps en moyenne pour chaque fonction, et le taux de réduction moyen. On peut voir que le cCSOA est plus rapide que CSOA, et que la moyenne du taux de réduction du temps d'exécution atteint 63%, ce qui est considérable.

7. Conclusion

Dans ce chapitre, nous avons introduit un nouvel algorithme compact appelé *compact cockroach swarm optimization algorithm* (cCSOA). Le but est de réduire les capacités de la mémoire et de calcul requises par la version originale du CSOA. Afin d'évaluer les performances de notre approche, nous l'avons testé sur un ensemble de fonctions de référence connues, toutes tirées du Toolbox de IEEE issu de congrès international pour le calcul évolutionnaire de 2014 (CEC2014). Ce dernier est largement utilisé par la communauté scientifique. Les résultats obtenus montrent que le cCSOA est très compétitif comparé à la version originale du CSOA, à GA, à DE et au PSO. Il fournit de meilleurs résultats dans la plupart des fonctions de test. L'approche développée dans ce mémoire assure à la fois diversification et intensification de la recherche dans l'espace décisionnel. Nous avons

atteint notre objectif fixé et nous avons réussi à réduire les capacités de calcul requises par l'algorithme CSOA avec l'utilisation de la représentation compacte. Ainsi nous avons amélioré les performances de CSOA et on a aussi réussi à réduire le temps d'exécution avec un pourcentage très important. Ceci est dû à la réduction de la complexité de l'algorithme.

Conclusion Générale

Conclusions

Ce travail est un mémoire de Master en Automatique et systèmes effectué au sein de l'université A. MIRA-Bejaia. Il est consacré à la réduction des capacités de calculs requises par les algorithmes bio-inspirés par l'exploitation des concepts de la représentation compacte, avec application à l'algorithme CSOA. .

Au cours des dernières années, diverses méthodes d'optimisation métaheuristique ont été développées. Beaucoup de ces algorithmes sont inspirés de la nature et par le comportement de certaines espèces et surtout celles qui vivent en groupes. Ces méthodes sont souvent appliquées à la résolution des problèmes difficiles, mathématiques ou d'ingénierie. Notre mémoire est consacré à l'étude de l'algorithme d'optimisation des essaims de blattes (CSOA). Le CSOA imite le comportement des blattes dans leur quête de nourriture. Il se base sur le déplacement des blattes et leur façon de socialiser afin de trouver une solution optimale. Le processus de dispersion de chacun des agents de l'essaim permet au CSOA de garder la diversité et donc de ne pas aboutir à une convergence prématurée (converger vers un optimum local).

Comme tout autre algorithme basé sur une population, CSOA présente un inconvénient en ce qui concerne ses exigences élevées en capacité de calculs, surtout quand la dimension du problème est assez grande. Ceci se traduit en un temps d'exécution élevé et une difficulté d'implémentation quand les capacités de calcul ne sont pas disponibles. Dans ce mémoire nous avons proposé une variante de l'algorithme CSOA. Nous l'avons nommé ***compact cockroach swarm optimization algorithm (*cCSOA*).*** Notre objectif été de réussir à réduire les capacités des calculs et de mémoire requises par l'algorithme original. Pour ce faire, nous avons conduit notre travail en trois étapes.

La première consiste à présenter une synthèse de quelques systèmes bio-inspirés et de fournir un bref état de l'art des techniques utilisées, cela nous permit d'acquérir de profondes connaissances dans ce domaine. Dans la deuxième étape nous avons étudié en profondeur l'algorithme CSOA qui est basé sur le comportement d'un essaim de blattes et son principe de fonctions. Nous avons aussi présenté un bref état de l'art de cette méthode, incluant des variantes et des applications retrouvées dans la littérature spécialisée. Dans la troisième étape, nous avons étudié la représentation compacte et on l'a appliqué sur CSOA afin d'obtenir un

algorithme compact. L'algorithme obtenu a été baptisé ***compact cockroach swarm optimization algorithm.*** Afin d'étudier les performances de notre approche, nous avons utilisé le Toolbox de IEEE issu de congrès international pour le calcul évolutionnaire de 2014 (CEC2014). Ce Toolbox est largement recommandé par les chercheurs au niveau mondial afin d'évaluer et de comparer les algorithmes destinés à l'optimisation mathématique et industrielle. Ce qui est le cas du cCOSA introduit dans ce mémoire.

Les grandes contributions de ce travail sont trois. Nous avons présenté une nouvelle variante de CSOA appelée cCSOA. Nous avons réussi à réduire les capacités de calcul requises par l'algorithme original. Les résultats obtenus sont comparés aux algorithmes génétiques (GA), algorithme d'évolution différentielle (DE), le PSO et le CSOA. Notre étude révèle que le cCSOA est plus performant. Son temps d'exécution est beaucoup plus inférieur comparé à celui de l'algorithme original, on peut dire qu'on a atteint notre objectif.

Perspectives

Ce travail peut être amélioré dans l'avenir. Du moins, nous proposons dans les travaux à venir de :

- Implémenter dans le futur notre algorithme sur des problèmes réels.
- Introduction de la variable « faim » dans notre algorithme peut améliorer notre approche.
- Utiliser les distributions uniformes afin de réduire encore plus les capacités de calcul requises et sa complexité.

Référence bibliographique

[1] Bonabeau, E., Dorigo, M., & Theraulaz, G. L'Intelligence en essaim. In *JFIADSMA*(1999, November), (pp. 25-38).

[2] Labed. S, " Méthodes bio-inpirées hybrides pour la résolution de problèmes complexes",Thèse de doctorat en science, Université de Constantine, 2013

[3] T. Bäck, EvolutionaryAlgorithms in Theory and Practice, Oxford UniversityPress, 1996

[4] Amédée, Souquet, and RadetFrancois-Gérard. "'ALGORITHMES GENETIQUES'." TE de fin d'année (2004).)

[5] R. Dupas**,** "Amélioration de performance des systèmes de production: apport des algorithmes évolutionnistes aux problèmes d'ordonnancement cycliques et flexibles", Université d'Artois (2004).

[6] A. Nabonne. "Algorithmes évolutionnaires et problèmes inverses", chapitre 8, juin 2004.

[7] J.PhilippeRennard, "GeneticAlgorithmViewer: Démonstration d'un algorithme génétique",www.rennad.org/alif, Avril 2000.]

[8] D. Dasgupta&Z.Ji&F.Gonzalez « Artificial Immune System (AIS) Research in the Last Five Years », IEEE.2003]

[9] Artificial Immune Systems: A novel data analysis technique inspired by the immune network theory », PhDThesis, University of Wales, 2001**.**

[10] Stephanie Forrest, Alan S. Perelson, Lawrence Allen, and Rajesh Cherukuri. Self-nonself discrimination in a computer. In *SP '94 : Proceedings of the 1994 IEEE Symposium on Security and Privacy*, page 202, Washington, DC, USA, 1994. IEEE Computer Society.

[11] E. Bonabeau, M. Dorigo, and G. Theraulaz, Swarm Intel ligence: From Natural to ArtcialSystems. NY: Oxford Univ. Press, 1999.

[12] Ruiqing and T. Wansheng, Monkeyalgorithm for global numericaloptimization, Journal of UncertainSystems, vol. 2, no. 3, pp. 164–175, 2008

[13] X.-L. Li, Z.-J Shao and J.-X Qian, Optimizing method based on autonomous animats: Fish-swarm algorithm, Xitong Gongcheng Lilun yu Shijian/System Engineering Theory and Practice, 22(11):32, 2002.

[14] Broderick Crawford, Ricardo Soto, Natalia Berríos, Franklin Johnson, Fernando Paredes, Carlos Castro, and Enrique Norero" A Binary Cat SwarmOptimizationAlgorithm for the Non-Unicost Set CoveringProblem, HindawiPublishing Corporation MathematicalProblems in Engineering Volume 2015, Article ID 578541, PP. 1-8

[15] S.-A Chu, P-W Tsai, and J.-S Pan, Cat swarm optimization. Lecture Notes in Computer Science (includingsubseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 4099 LNAI:854– 858, 2006. Cited By (since 1996)

[16] Z. Zandi, E. AFJEI, and M.Sedighizadeh, Hybrid Big Bang–Bag Crunch Optimization Based optimal reactive power dispatch for voltage stability enhancement, Journal of Theoretical and Applied Information Technology 20th January 2013, Vol. 47 No.2, PP. 537-546.

[17] Lam, A. Y., & Li, V. O. (2012). Chemical reaction optimization: A tutorial. *MemeticComputing*, *4*(1), 3-17.

[18] Pinar Civicioglu, Backtracking search optimizational gorithm for numerical optimization problems, Applie dMathematics and Computation, 219(15):8121–8144, 2013.

[19] Kennedy, J.; Eberhart, R.C. Particle Swarm Optimization. IEEE Neural Netw. Proc. **1995**, 4, 1942–1948.

[20] Dorigo, M. Optimization, Learning and Natural Algorithms. Ph.D. Thesis, Politecnicodi Milano, Milano,Italy, 1992

[21] C. Blum and D. Merkle, "Swarm Intelligence Introduction and Applications", Natural Computing Series, Springer-verlag Berlin Heidelberg, (2008).

[22] D. Karaboga and A. Bahriye, "A survey: Algorithm Simulating Bee Swarm Intelligence", Springer Science and Business Media, (2009).

[23] T. C. Havens, C. J. Spain, N. G. Salmon, and J. M. Keller, "Roach infestation optimization," in Proceedings of the IEEE SwarmI ntelligence Symposium (SIS '08), September 2008.

[24] Chen, Z.H.; Tang, H.Y. Notice of Cockroach Swarm Optimization. In Proceedings of the Second International Conference on Computer Engineering and Technology (ICCET), Chengdu, China, 16–18 April 2010.

[25] M. Lihoreau, J. Costa and C. Rivault, "The Social Biology of Domiciliary Cockroaches: Colony Structure, Kin Recognition and Collective Decision" International Union for the Study of Social Insect, Published Online, (2012).

[26] Chen, Z. A Modified Cockroach Swarm Optimization. Adv. Eng. Forum **2011**, 11, 4–9.

[27] Cheng, L.; Wang, Z.; Song, Y.; Guo, A. Cockroach Swarm Optimization Algorithm for TSP. Adv. Eng. Forum **2011**, 1, 226–229.

[28] Chen, Z.; Tang, H.Y. Cockroach Swarm Optimization for Vehicle Routing Problems. Energy Proced. **2011**, 13, 30–35

[29] C. ZhaoHui and T. HaiYan, "Cockroach swarm optimization,"in *Proceedings of the 2nd International Conference on ComputerEngineering and Technology (ICCET '10)*, vol. 6,pp. 652–655,April 2010.

[30] T. Havens, C. Spain, N. Salmon and J. Keller, “Roach Infestation Optimization”, IEEE Swarm Intelligence Symposium, (2008).

[31] R. Jeanson, et al. “Self-organized Aggregation in Cockroaches”, Animal Behaviour,**69**, 169-180, (2005).

[32] J. B. Williams, M. Louis, R. Christine and A. Nalepal, “Cockroaches Ecology, Behaviour and Natural History”, Johns Hopkins University Press Baltimore, 2007.

[33] OBAGBUWA, Ibidun et ABIDOYE, Ademola. Binary cockroach swarm optimization for combinatorial optimization problem. Algorithms, 2016, vol. 9, no 3, p. 59.

[34] OBAGBUWA, I. C. et ADEWUMI, A. O. An improved cockroach swarm optimization. The Scientific World Journal, 2014, vol. 2014

[35] Cheng, Le, et al. "Cockroach swarm optimization algorithm for TSP." *Advanced Engineering Forum*. Vol. 1. Trans Tech Publications, 2011.

[36] Larra˜naga, P., Lozano, J. A., 2001. Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation. Kluwer.

[37] Neri, Ferrante, Giovanni Iacca, and Ernesto Mininno. "Compact Optimization."

[38] Harik GR, Lobo FG, GoldbergDE (1999) The compact genetic algorithm. IEEE Transactions on Evolutionary Computation 3(4):287–297

[39] J. C. Gallagher, S. Vigraham, and G. Kramer, “A family of compact genetic algorithms for intrinsic evolvable hardware,” *IEEE Trans. Evol. Comput.*, vol. 8, pp. 111–126, Apr. 2004.

[40] R. Baraglia, J. I. Hidalgo, and R. Perego, “A hybrid heuristic for the traveling salesman problem,” *IEEE Trans. Evol. Comput.*, vol. 5, no. 6, pp. 613–622, Dec. 2001.

[41] Mininno E, Cupertino F, Naso D (2008) Real-valued compact genetic algorithms for embedded microcontroller optimization. IEEE Transactions on Evolutionary Computation 12(2):203–219

[42] Gautschi W (1972) Error function and fresnel integrals. In: Abramowitz M, Stegun IA (eds) Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, chap 7, pp 297–309

[43] Mininno E, Neri F, Cupertino F, Naso D. Compact differential evolution. IEEE Transactions on Evolutionary Computation. 2011 Feb;15(1):32-54.

[44] Neri F, Mininno E, Iacca G. Compact particle swarm optimization. Information Sciences. 2013 Aug 1;239:96-121.

[45] Yang Z, Li K, Guo Y. A new compact teaching-learning-based optimization method. InInternational Conference on Intelligent Computing 2014 Aug 3 (pp. 717-726). Springer, Cham.

[46] Soares AS, de Lima TW, Soares FA, Coelho CJ, Federson FM, Delbem AC, Van Baalen J. Mutation-based compact genetic algorithm for spectroscopy variable selection in

determining protein concentration in wheat grain. Electronics Letters. 2014 Jun 10;50(13):932-4.

[47] Ha BV, Mussetta M, Pirinoli P, Zich RE. Modified compact genetic algorithm for thinned array synthesis. IEEE Antennas and Wireless Propagation Letters. 2016;15:1105-8.

[48] Yang X. S.,Firefly algorithm for multimodal optimization in proceedings of the stochastic Algorithms", (SAGA 109) vol.5792, Oct.2009.

[49] Saoucha A., Ghanem N, K., Benmammar B., On applying firefly algorithm for cognitive radio networks. Communications and Mirjana VUKOVIC, Milenko PIKULA, "Firefly Algorithm for Constrained Optimization Problems". ISBN Vehicular Technology in the Benelux (SCVT), 2014 IEEE 21st Symposium on. IEEE, 2014.

[50] Romana C. H., Adis A., Milan TUBA,: 978-960-474-330-8.

[51] Prabhneet k, Taranjot k., A Comparative Study of Various Metaheuristic Algorithms, Guru Teg Bahadur Institute of Technology, GGSIPU, New Delhi.

[52] X.S. Yang, *Nature-Inspired Metaheuristic Algorithms*, Luniver Press, Frome, 2008.

[53] Tighzert L, Mendil B. CFO: A new compact swarm intelligent algorithm for global optimization and optimal bipedal robots walking. In Modelling, Identification and Control (ICMIC), 2016 8th International Conference on 2016 Nov 15 (pp. 487-492). IEEE.

[54] Tighzert L, Fonlupt C, Mendil B. A set of new compact firefly algorithms. Swarm and Evolutionary Computation. 2017 Dec 16.

[55] J. J. Liang, B. Y. Qu, P. N. Suganthan, "Problem Definitions and Evaluation Criteria for the CEC 2014 Special Session and Competition on Single Objective Real-Parameter Numerical Optimization," Technical Report 201311, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China And Technical Report, Nanyang Technological University, Singapore, Technical Report, Dec. 2013.

[56] Tanabe, R., &Fukunaga, A. (2013, June). Success-history based parameter adaptation for differential evolution. In *2013 IEEE congress on evolutionary computation* (pp. 71-78). IEEE

[57] Kennedy, J.; Eberhart, R. (1995). "Particle Swarm Optimization". Proceedings of IEEE International Conference on Neural Networks. IV. pp. 1942–1948. doi:10.1109/ICNN.1995.488968.

[58] Abido, M. A. (2002). Optimal power flow using particle swarm optimization. *International Journal of Electrical Power &Energy Systems*, *24*(7), 563-571.

[59] Tighzert L, Fonlupt C, Mendil B. Towards compact swarm intelligence: a new compact firefly optimisation technique. International Journal of Computer Applications in Technology. 2019 Jun 7;60(2):108-23.

[60] Tighzert L, Fonlupt C, Mendil B, Bruneau O. New compact music-inspired algorithms. In2017 5th International Conference on Electrical Engineering-Boumerdes (ICEE-B) 2017 Oct 29 (pp. 1-6). IEEE.

Résumé

Ce travail est un projet de fin d'étude pour l'obtention du diplôme de Master en Automatique. Il est dédié à la proposition d'un nouvel algorithme bio-inspiré. L'algorithme proposé est appelé ***compact cockroach algorithms*** (cCA). Il est inspiré des comportements d'un essaim de blattes et se base sur la représentation compacte. Il fait alors partie des algorithmes de la **compact swarm intelligence**. Notre approche vise à développer un algorithme qui ne nécessite qu'un très faible cout de calcul. Le cCA est validé en se basant sur les critères proposés par IEEE au CEC2014. Les résultats obtenus montrent que notre algorithme est très compétitif comparé à l'état de l'art du domaine.

Abstract

This work is a project of end of study for the obtention of the diploma of Master in Automatic. It is dedicated to the proposal of a new bio-inspired algorithm. The proposed algorithm is called **compact cockroach algorithms** (cCA). It is inspired by the behaviors of a swarm of cockroaches and is based on the compact representation. It is considered as a compact swarm intelligence algorithm. Our approach aims to develop an algorithm that requires only a very low cost of calculation. The cCA is validated based on the criteria proposed by IEEE at CEC2014. The results obtained show that our algorithm is very competitive compared to the domain's state-of-art.