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

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

Projet de Fin de Cycle

Présenté en vue de l'obtention du diplôme de Master en Automatique

Thème

Proposition d'un nouvel algorithme d'optimisation basé sur la théorie de la gravitation

---

Préparé et présenté par :

Farid LACHOURI

Mourad TIBOUCHE

Dirigé par :

Dr. Lyes TIGHZERT

Prof. Boubekeur MENDIL

Devant le jury :

M. S.Mezzah

Mr. S.Hadji

Année Universitaire : **2017/2018**

Remerciements

Ce travail a été effectué sous la direction de Monsieur Dr. Lyes TIGHZERT et du prof. Boubekeur MENDIL, on les remercie très sincèrement pour la confiance qu'ils nous ont accordée, pour la patience inouïe qu'ils nous ont témoignée tout au long de l'élaboration de ce travail. Nos discussions stimulantes ont beaucoup contribué à la clarification des notions ainsi qu'à la bonne présentation de ce mémoire. Qu'ils reçoivent l'expression de notre très grande reconnaissance.

Enfin, dans le souci de n'oublier personne, que tous ceux qui nous ont aidés, de près ou de loin, que ce soit par leur amitié, leurs conseils ou leur soutien moral et matériel, trouvent dans ces quelques aimables et sincères mots l'expression de notre profonde gratitude.

Sommaires

LISTE DES TABLEAUX

LISTE DES FIGURES

INTRODUCTION GENERALE

Contexte

Pour résoudre les problèmes d'optimisation avec un espace de recherche de grande dimension, les algorithmes d'optimisation classiques ne fournissent pas une solution appropriée en raison de l'espace de recherche qui augmente avec la taille du problème. Au cours des dernières décennies, il y a eu un intérêt croissant pour les algorithmes inspirés par les comportements des phénomènes naturels. De nombreux chercheurs ont montré que ces algorithmes sont bien adaptés pour résoudre des problèmes complexes dans des domaines différents. Diverses approches heuristiques ont été adoptées par les recherches, on peut citer : les algorithmes génétiques, le recuit simulé, l'algorithme de recherche par des colonies de fourmis et l'optimisation par des essaims de particulaire (PSO). Ces algorithmes ont trouvé des applications diverses dans de nombreux domaines. Cependant, il n'y a pas d'algorithme spécifique pour obtenir la meilleure solution pour tous les problèmes. Certains algorithmes donnent une meilleure solution pour certains problèmes particuliers que d'autres. Par conséquent, la recherche de nouveaux algorithmes d'optimisation heuristique reste un problème ouvert. C'est dans ce contexte que s'inscrit l'étude menée dans ce mémoire.

Problématique

L'une des méthodes de recherche heuristique récemment développée est connue sous le nom de la recherche gravitationnelle, ou Gravitational Search Algorithm (GSA). Basée sur les lois de la théorie newtonienne de la gravitation, cette méthode s'est révélée très performante pour résoudre des problèmes complexes. Comme tant d'autres algorithmes basés sur des populations, le GSA nécessite des capacités de calculs et de mémoire très élevées. Ceci rend son implémentation hardware sur des microcontrôleurs en vue d'une réalisation pratique très délicate. Nous traitons dans ce mémoire les problématiques liées à l'espace mémoire et les capacités de calculs requis par le GSA. L'objectif est de réduire significativement les capacités de calculs et de mémoire que le GSA nécessite.

La solution proposée dans ce mémoire est basée sur la représentation compacte. Nous proposons de compacter toute l'information reliée aux masses, c'est-à-dire aux particules, que le GSA utilise pour trouver la solution optimale. L'l'algorithme que nous introduisons ici est appelé : compact gravitational search algorithm (cGSA). Ce dernier utilise une distribution pour représenter la population. L'algorithme obtenu ne nécessite qu'un strict minimum d'espace mémoire et de capacité de calculs.

La méthodologie suivie dans notre étude, en vue de la validation de notre approche, est inspirée des travaux récents en relation avec les métaheuristiques. Nous proposons de tester notre algorithme sur des fonctions benchmarks qui ont été utilisées par les auteurs qui ont introduit le GSA. Les résultats obtenus sont comparés au GSA classique et au PSO. Le cGSA s'est révélé très compétitif et son application réelle est recommandée. C'est ce qui nous a motivés à l'utiliser en robotique mobile. Notre approche a été implémentée pour résoudre un problème de planification de trajectoire d'un robot mobile dans un environnement qui contient des obstacles.

L'organisation de ce mémoire est répartie en quatre chapitres. Dans le premier, nous proposons un état de l'art qui recouvre une recherche bibliographique sur les métaheuristiques. Dans le deuxième chapitre, nous présentons la recherche gravitationnelle et son principe de fonctionnement. Le troisième chapitre est consacré à la représentation compacte et les algorithmes qui utilisent ce concept. Le dernier chapitre de ce mémoire introduit notre contribution. Nous présentons le cGSA, son principe de fonctionnement, ses résultats expérimentaux et une étude comparative en vue de la validation de notre approche. Les résultats obtenus par le cGSA sont très compétitifs. Son application à un problème de planification de trajectoire d'un robot en présence d'obstacles montre que le cGSA présente de bonnes performances.

CHAPITRE 1

Introduction aux méta-heuristiques

I.1. Introduction :

Une métaheuristique est un algorithme d'optimisation visant à résoudre des problèmes d'optimisation difficile (souvent issus des domaines de la recherche opérationnelle, de l'ingénierie ou de l'intelligence artificielle) pour lesquels on ne connaît pas de méthode classique plus efficace.

Les métaheuristiques sont des algorithmes stochastiques itératifs, qui progressent vers un optimum global, c'est-à-dire l'extremum global d'une fonction, par échantillonnage d'une fonction objectif. Elles se comportent comme des algorithmes de recherche, tentant d'apprendre les caractéristiques d'un problème afin d'en trouver une approximation de la meilleure solution (d'une manière proche des algorithmes d'approximation).

Il existe un grand nombre de métaheuristiques différentes, allant de la simple recherche locale à des algorithmes complexes de recherche globale. Ces méthodes utilisent cependant un haut niveau d'abstraction, leur permettant d'être adaptées à une large gamme de problèmes différents. Dans ce chapitre, on va présenter certains algorithmes, ainsi que la classification de ces algorithmes. [01]

I.2. Généralités

I.2.1. Terminologies

Le mot «heuristique» est grec et signifie «savoir», «trouver», «découvrir» ou «guider une enquête» [52]. Plus précisément, «les heuristiques sont des techniques qui recherchent de bonnes solutions (quasi-optimales) à un coût de calcul raisonnable sans pouvoir garantir la faisabilité ou l'optimalité, ou même dans de nombreux cas indiquer à quel point une solution réalisable est proche de l'optimalité. Russell et Norvig, 1995 [59].

Une terminologie légèrement différente considère que les métaheuristiques sont une forme d'algorithmes d'optimisation stochastique, hybridés avec une recherche locale. Le terme méta est donc pris au sens où les algorithmes peuvent regrouper plusieurs heuristiques. On rencontre cette définition essentiellement dans la littérature concernant les algorithmes évolutionnaires, où elle est utilisée pour désigner une spécialisation. Dans le cadre de la première terminologie, un algorithme évolutionnaire hybridé avec une recherche locale sera plutôt désigné sous le terme d'algorithme mimétique.

Les métaheuristiques sont souvent inspirées des systèmes naturels, qu'ils soient pris en physique (cas du recuit simulé), en biologie de l'évolution (cas des algorithmes génétiques) ou encore en *éthologie* (cas des algorithmes de colonies de fourmis ou de l'optimisation par des essaims particulaires).

I.2.2.Nomenclature :

Le but d'une métaheuristique est de résoudre un problème d'optimisation donné : elle cherche (une permutation, un vecteur, etc.) minimisant (ou maximisant) une fonction objectif, qui décrit la qualité d'une solution au problème.

L'ensemble des solutions possibles forme l'espace de recherche. L'espace de recherche est au minimum borné, mais peut être également limité par un ensemble de contraintes.

Les métaheuristiques manipulent une ou plusieurs solutions, à la recherche de l'optimum, la meilleure solution au problème.

Les itérations successives doivent permettre de passer d'une solution de mauvaise qualité à la solution optimale. L'algorithme s'arrête après avoir atteint un critère d'arrêt, consistant généralement en l'atteinte du temps d'exécution imparti, en nombre d'itérations octroyées, ou en une précision demandée.

Une solution ou un ensemble de solutions est parfois appelé un état, que la métaheuristique fait évoluer via des transitions ou des mouvements. Si une nouvelle solution est construite à partir d'une solution existante, elle est sa voisine, le choix du voisinage et de la structure de donnée le représentant peut être crucial.

Lorsqu'une solution est associée à une seule valeur, on parle de problème mono-objectif, lorsqu'elle est associée à plusieurs valeurs, de problème multiobjectif (ou multicritères). Dans ce dernier cas, on recherche un ensemble de solutions non dominées (le « front de Pareto »), solutions parmi lesquelles on ne peut décider si une solution est meilleure qu'une autre, aucune n'étant systématiquement inférieure aux autres sur tous les objectifs.

Dans certains cas, le but recherché est explicitement de trouver un ensemble d'optimums « satisfaisants ». L'algorithme doit alors trouver l'ensemble des solutions de bonne qualité, sans nécessairement se limiter au seul optimum : on parle de méthodes multimodales.

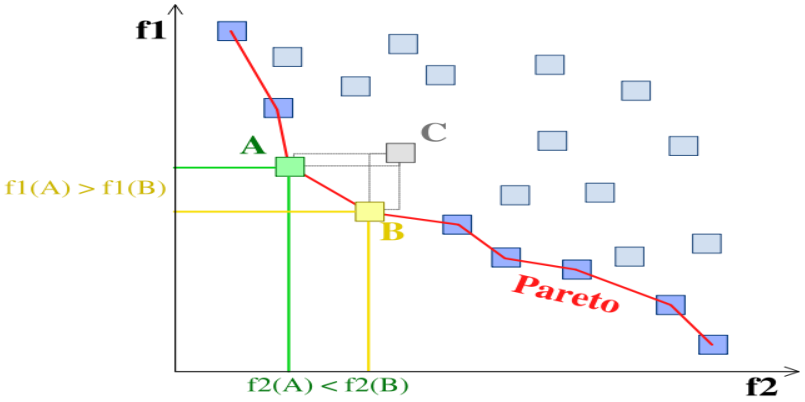

Fig.I.1. Exemple de front de Pareto dans un problème nécessitant la minimisation de deux objectifs (f1 et f2). Les points A et B sont « non dominés » alors que le point C n'est optimum pour aucun des objectifs [113].

I.2.3.Concepts généraux :

Les métaheuristiques ne nécessitent pas de connaissances particulières sur le problème optimisé pour fonctionner, le fait de pouvoir associer une (ou plusieurs) valeur à une solution est la seule information nécessaire.

En pratique, elles ne devraient être utilisées que sur des problèmes ne pouvant être optimisés par des méthodes mathématiques. Utilisées en lieu et place d'heuristiques spécialisées, elles montrent généralement de moins bonnes performances.

Les métaheuristiques sont souvent employées en optimisation combinatoire, mais on en rencontre également pour des problèmes continus ou mixtes (problèmes à variables discrètes et continues).

Certaines métaheuristiques sont théoriquement « convergentes » sous certaines conditions. Il est alors garanti que l'optimum global sera trouvé en un temps fini, la probabilité de ce faire augmentant asymptotiquement avec le temps. Cette garantie revient à considérer que l'algorithme se comporte au pire comme une recherche aléatoire pure (la probabilité de tenter toutes les solutions tendant vers 1). Cependant, les conditions nécessaires sont rarement vérifiées dans le cadre d'applications réelles. En pratique, la principale condition de convergence est de considérer que l'algorithme est ergodique (qu'il peut atteindre n'importe quelle solution à chaque mouvement), mais on se satisfait souvent d'une quasi-ergodicité (si la métaheuristique peut atteindre n'importe quelle solution en un nombre fini de mouvements).

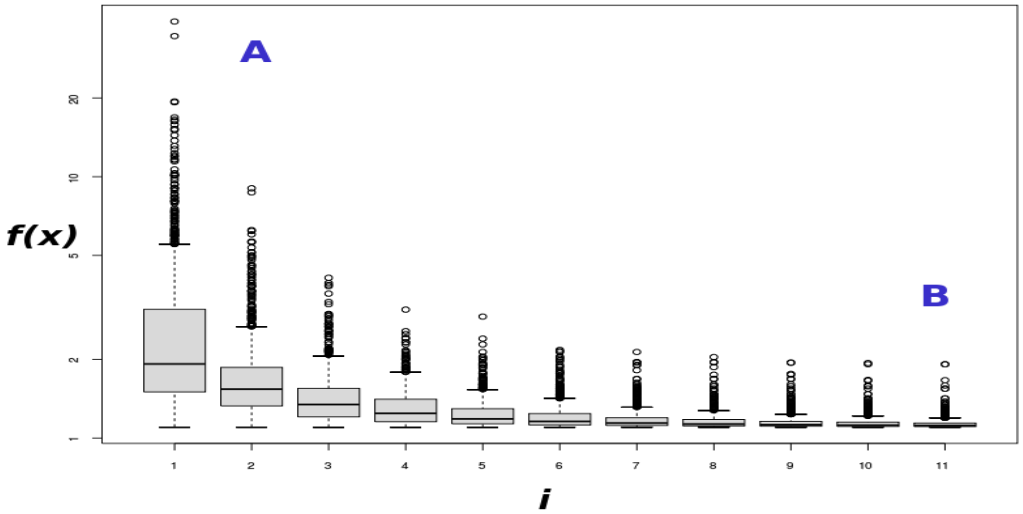

Fig.I.2 Comportement d'une métaheuristique. Le graphique représente les distributions des valeurs des optimums trouvés (sur un grand nombre d'exécutions) : l'algorithme passe d'une population de solution très dispersée (A) à une population plus centrée sur l'optimum trouvé (B) [114].

I.2.4.Organisation générale

D'une manière générale, les métaheuristiques s'articulent autour de plusieurs notions [109] :

- Voisinage ;
- Diversification/exploration ;
- Intensification/exploitation ;

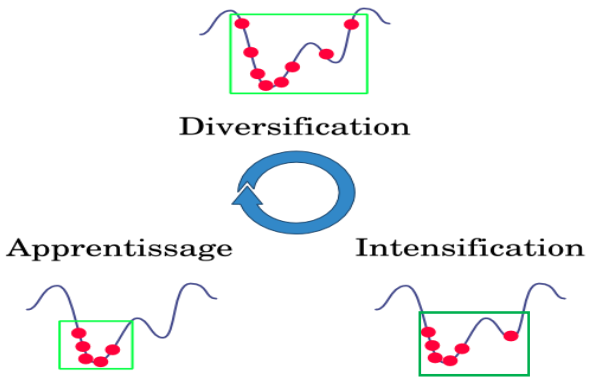

Fig.I.3. Les trois phases d'une métaheuristique itérative. Les points rouges représentent l'échantillonnage de la fonction objectif (ici à une dimension) [116].

I.2.4.1.Voisinage :

Le voisinage d'une solution est un sous-ensemble de solutions qu'il est possible d'atteindre par une série de transformations données. Par extension on désigne parfois par le terme « voisinage » l'ensemble des transformations considérées.

Un voisinage simple pour le problème du voyageur de commerce sera, par exemple, l'ensemble des solutions qu'il est possible de construire en permutant deux villes dans une solution donnée.

La notion de voisinage est sans doute le principe général le plus utilisé pour la conception d'heuristiques. Pour les problèmes combinatoires, le voisinage a un impact important sur le comportement des métaheuristiques, alors que pour des problèmes continus, la notion même de voisinage est plus difficile à cerner.

Bien qu'il n'existe que très peu de résultats théoriques sur l'adéquation entre un voisinage et un problème discret donné, il peut être possible d'en calculer des indicateurs empiriques, comme la rugosité [2]. Les techniques les plus classiques concernant la définition d'un voisinage tournent autour des notions de permutations, de chaînes d'éjections et d'optimisations partielles.

I.2.4.2.Intensification, diversification, apprentissage

La diversification (ou *exploration*: synonyme utilisé presque indifféremment dans la littérature spécialisée) désigne les processus visant à récolter de l'information sur le problème à optimiser. L'intensification (ou *exploitation*) vise à utiliser l'information déjà récoltée pour définir et parcourir les zones intéressantes de l'espace de recherche. La mémoire est le support de l'apprentissage, qui permet à l'algorithme de ne tenir compte que des zones où l'optimum global est susceptible de se trouver, évitant ainsi les optimums locaux.

Les métaheuristiques progressent de façon itérative, en alternant des phases d'intensification, de diversification et d'apprentissage, ou en mêlant ces notions de façon plus étroite. L'état de départ est souvent choisi aléatoirement, l'algorithme s'exécute ensuite jusqu'à ce qu'un critère d'arrêt soit atteint.

Les notions d'intensification et de diversification sont prépondérantes dans la conception des métaheuristiques, qui doivent atteindre un équilibre délicat entre ces deux dynamiques de recherche. Les deux notions ne sont donc pas contradictoires, mais complémentaires, et il existe de nombreuses stratégies mêlant à la fois l'un et l'autre de ces aspects.

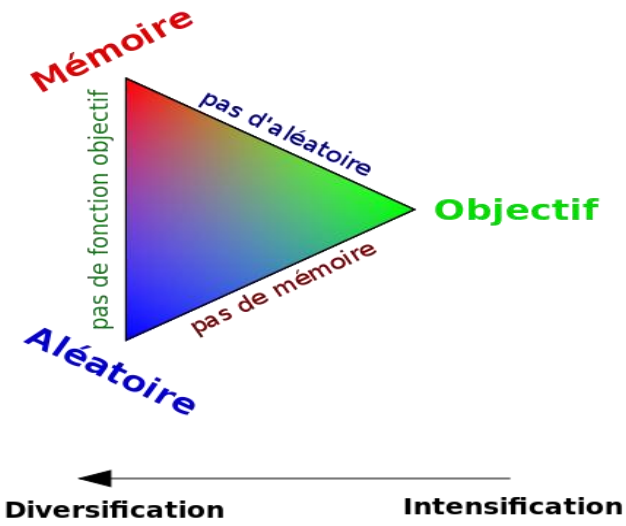

Fig.I.4. Les notions d'intensification et de diversification sont liées à l'utilisation de la fonction objective et aux processus aléatoires. Combinées avec la notion de mémoire, elles permettent de positionner les différents aspects des métaheuristiques entre eux [117].

I.3. Classification

Selon Clerc et Siarry [01] les métaheuristiques ont en commun les caractéristiques suivantes :

- La plupart des métaheuristiques utilisent des processus aléatoires comme moyens de récolter de l'information et de faire face à des problèmes comme l'explosion combinatoire.
- En plus de cette base stochastique, les métaheuristiques sont généralement itératives, c'est à dire qu'un même schéma de recherche est appliqué plusieurs fois au cours de l'optimisation, et directes, c'est à dire qu'elles n'utilisent pas l'information du gradient de la fonction objective. Elles tirent en particulier leur intérêt de leur capacité à éviter les optimums locaux, soit en acceptant une dégradation de la fonction objective au cours de leur progression, soit en utilisant une population de points comme méthode de recherche.
- Les métaheuristiques du fait de leur capacité à être utilisées sur un grand nombre de problèmes différents, se prêtent facilement à des extensions.
- Elles sont inspirées par analogie avec la réalité: avec la physique (le recuit simulé), avec la biologie (les algorithmes génétiques) ou avec l'éthologie (les colonies de fourmis)...
- Les concepts de base des métaheuristiques peuvent être décrits de manière abstraite, sans faire appel à un problème spécifique.
- Les métaheuristiques peuvent contenir des mécanismes qui permettent d'éviter d'être bloqué des régions de l'espace de recherche.
- Elles partagent aussi les mêmes inconvénients : les difficultés de réglage des paramètres, et le temps de calcul élevé.

I.3.1.Parcours et population

Les métaheuristiques les plus classiques sont celles fondées sur la notion de parcours. Dans cette optique, l'algorithme fait évoluer une seule solution sur l'espace de recherche à chaque itération. La notion de voisinage est alors primordiale.

Les plus connues dans cette classe sont le recuit simulé [14], la recherche avec tabous [16], la recherche à voisinage variable [107], la méthode GRASP [23] ou encore les méthodes de bruitage [108].

Dans cette classification, l'autre approche utilise la notion de population. La métaheuristique manipule un ensemble de solutions en parallèle, à chaque itération.

On peut citer les algorithmes génétiques [12], l'optimisation par essaims particulaires [24,25], les algorithmes de colonies de fourmis [22].

La frontière est parfois floue entre ces deux classes. On peut ainsi considérer qu'un recuit simulé où la température baisse par paliers, comme une structure à population. En effet, dans ce cas on manipule un ensemble de points à chaque palier, il s'agit simplement d'une méthode d'échantillonnage particulière.

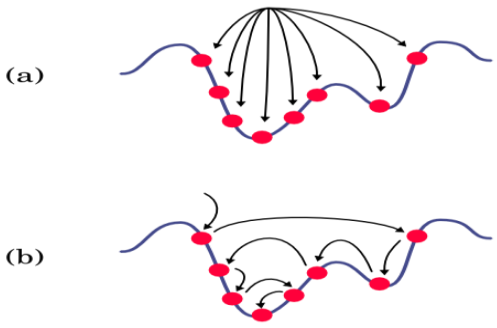

Fig.I.5. Principe général des métaheuristiques (a) à population, et (b) à parcours [118].

I.3.2.Emploi de mémoire :

Les métaheuristiques utilisent l'historique de leur recherche pour guider l'optimisation aux itérations suivantes.

Dans le cas le plus simple, elles se limitent à considérer l'état de la recherche à une itération donnée pour déterminer la prochaine itération : il s'agit alors d'un processus de décision markovien, et on parlera de méthode sans mémoire. C'est le cas de la plupart des méthodes de recherche locale.

Beaucoup de métaheuristiques utilisent une mémoire plus évoluée, que ce soit sur le court terme (solutions visitées récemment, par exemple) ou sur le long terme (mémorisation d'un ensemble de paramètres synthétiques décrivant la recherche).

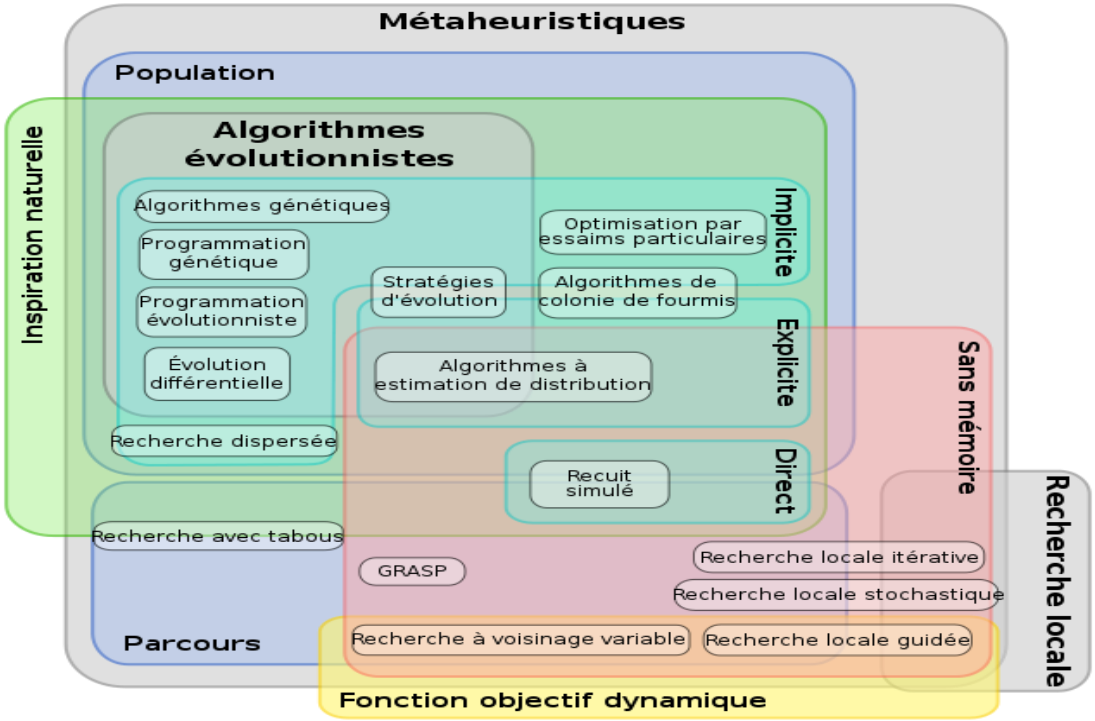

Fig.I.6. Les métaheuristiques peuvent être classées de nombreuses façons. Ce diagramme tente de présenter où se placent quelques-unes des méthodes les plus connues. Un élément présenté à cheval sur différentes catégories indique que l'algorithme peut être placé dans l'une ou l'autre classe, selon le point de vue adopté [115].

I.3.3.Fonction objectif statique ou dynamique :

La plupart des métaheuristiques utilisent la fonction objective en l'état, et font évoluer leur comportement de recherche de l'optimum. Cependant, certains algorithmes, comme la recherche locale guidée, modifient la représentation du problème, en incorporant l'information collectée durant la recherche, directement au sein de la fonction objective.

Il est donc possible de classer les métaheuristiques selon qu'elles utilisent une fonction objective statique (qui demeure inchangée tout au long de l'optimisation) ou dynamique (quand la fonction objective est modifiée au cours de la recherche), c'est-à-dire : problème dynamique.

I.3.4.Nombre de structures de voisinage

La plupart des métaheuristiques utilisées dans le cadre des problèmes d'optimisation combinatoire utilisent une seule structure de voisinage. Cependant, des

méthodes comme la recherche à voisinage variable permettent de changer de structure en cours de recherche [107].

I.3.5.Implicite, explicite, directe :

En considérant les métaheuristiques comme des méthodes itératives utilisant un échantillonnage de la fonction objective comme base d'apprentissage (définition plus particulièrement adaptée aux métaheuristiques à populations) le choix de l'échantillonnage entre deux itérations ne suit pas une loi donnée.

Dans la très grande majorité des cas, cet échantillonnage se fait sur une base aléatoire, et peut donc être décrit *via* une distribution de probabilités. Il existe alors trois classes de métaheuristiques, selon l'approche utilisée pour manipuler cette distribution.

La première classe est celle des méthodes *implicites*, où la distribution de probabilité n'est pas connue a priori. C'est le cas par exemple des algorithmes génétiques [111], où le choix de l'échantillonnage entre deux itérations ne suit pas une loi donnée, mais est fonction de règles locales.

Par opposition, on peut donc classer les méthodes *explicites*, qui utilisent une distribution de probabilité choisie à chaque itération. C'est le cas des algorithmes à estimation de distribution [111].

Dans cette classification, le recuit simulé [14] occupe une place particulière, puisqu'on peut considérer qu'il échantillonne la fonction objective en utilisant directement celle-ci comme distribution de probabilité (les meilleures solutions ayant une probabilité plus grande d'être tirées). Il n'est donc ni explicite ni implicite, mais plutôt « direct ».

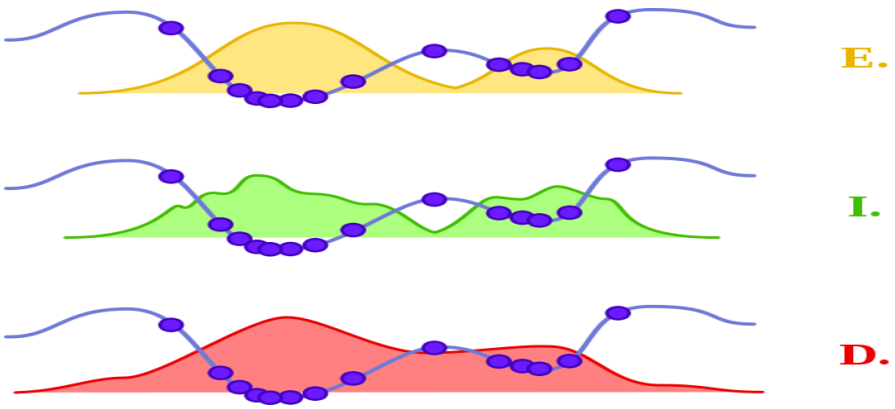

Fig.I.7. Les métaheuristiques peuvent être qualifiées d'explicites (E, ici sur une somme de deux gaussiennes), d'implicites (I) ou de directes (D), selon la façon dont elles gèrent la transition entre deux itérations [119].

I.3.6.Évolutionnaire ou non :

On trouve parfois une classification présentant les algorithmes d'optimisations stochastiques comme étant « évolutionnaire » ou non. L'algorithme sera considéré comme faisant partie de la classe des algorithmes évolutionnaires s'il manipule une population *via* des *opérateurs*, selon un algorithme général donné.

Cette façon de présenter les métaheuristiques dispose d'une nomenclature adaptée : on parlera d'opérateurs pour toute action modifiant l'état d'une ou plusieurs solutions. Un opérateur construisant une nouvelle solution sera dénommé *générateur*, alors qu'un opérateur modifiant une solution existante sera appelé mutateur.

Dans cette optique, la structure générale des algorithmes évolutionnaires enchaîne des étapes de sélection, de reproduction (ou croisement), de mutation et enfin de remplacement. Chaque étape utilise des opérateurs plus ou moins spécifiques.

I.3.7.Taxinomie de l'hybridation :

On parle d'hybridation quand la métaheuristique considérée est composée de plusieurs méthodes se répartissant les tâches de recherche. La taxinomie des métaheuristiques hybrides se sépare en deux parties : une classification hiérarchique et une classification plate. La classification est applicable aux méthodes déterministes aussi bien qu'aux métaheuristiques [4].

La classification hiérarchique se fonde sur le niveau (bas ou haut) de l'hybridation et sur son application (en relais ou concurrente). Dans une hybridation de bas niveau, une fonction donnée d'une métaheuristique (par exemple, la mutation dans un algorithme évolutionnaire) est remplacée par une autre métaheuristique (par exemple une recherche avec tabou [16]). Dans le cas du haut niveau, le fonctionnement interne « normal » des métaheuristiques n'est pas modifié. Dans une hybridation en relais, les métaheuristiques sont lancées les unes après les autres, chacune prenant en entrée la sortie produite par la précédente. Dans la concurrence (ou coévolution), chaque algorithme utilise une série d'agents coopérants ensemble.

Cette première classification dégage quatre classes générales:

- bas niveau et relais (abrégé LRH en anglais),
- bas niveau et coévolution (abrégé LCH),
- haut niveau et relais (HRH),
- haut niveau et coévolution (HCH).

La seconde partie dégage plusieurs critères, pouvant caractériser les hybridations :

- si l'hybridation se fait entre plusieurs instances d'une même métaheuristique, elle est homogène, sinon, elle est hétérogène ;
- si les méthodes recherchent dans tout l'espace de recherche, on parlera d'hybridation globale, si elles se limitent à des sous-parties de l'espace, d'hybridation partielle.
- si les algorithmes mis en jeu travaillent tous à résoudre le même problème, on parlera d'approche générale, s'ils sont lancés sur des problèmes différents, d'hybridation spécialisée.

Ces différentes catégories peuvent être combinées, la classification hiérarchique étant la plus générale.

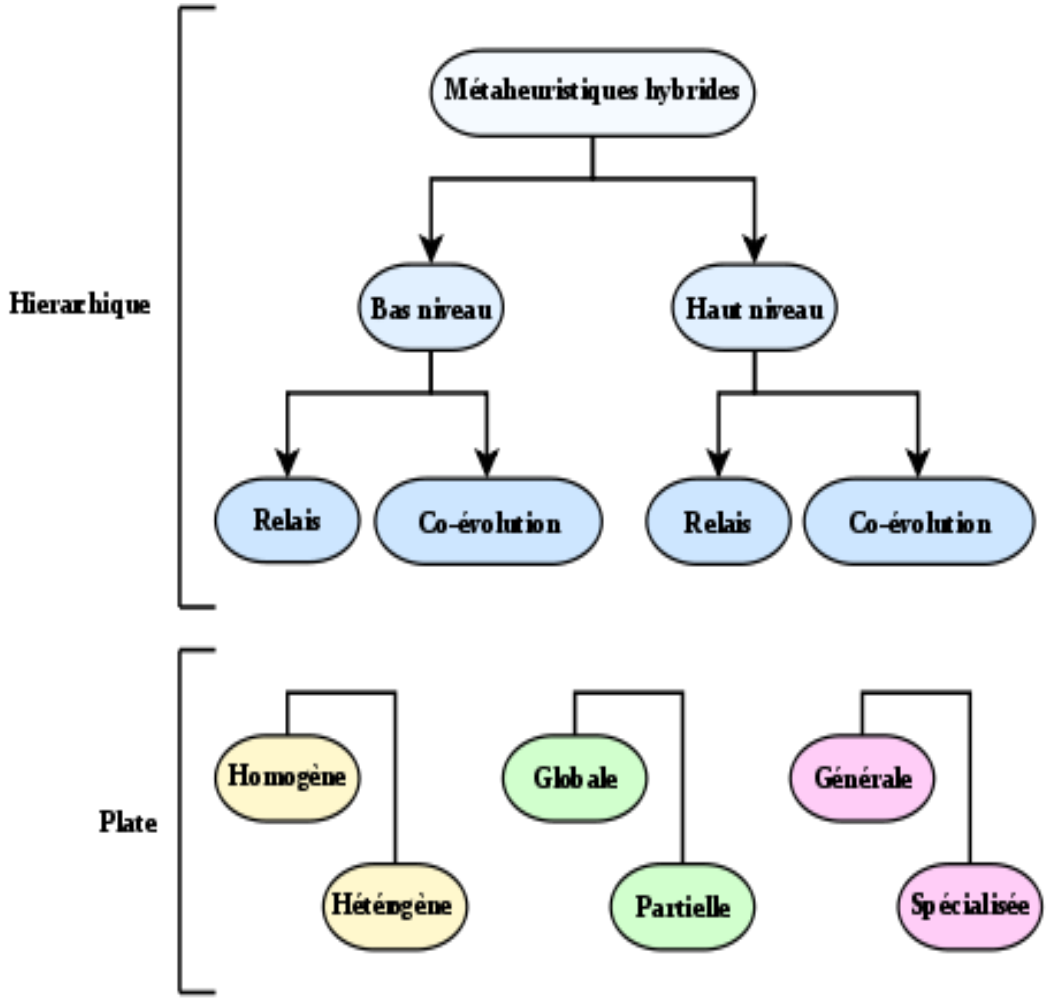

Fig.I.8. Taxinomie des métaheuristiques hybrides [120].

I.4.Applications

Les métaheuristiques sont souvent employées pour leur facilité de programmation et de manipulation. Elles sont en effet facilement adaptables à tout type de problème d'optimisation. Toutefois, elles sont le plus judicieusement employées sur des problèmes d'optimisation difficile, où des méthodes d'optimisation plus classiques (méthodes déterministes, notamment) montrent leurs limites.

De façon générale, on peut considérer que des problèmes présentant les caractéristiques suivantes sont assez propices à l'utilisation de métaheuristiques :

- NP-complétude ;
- nombreux optima locaux ;
- discontinuités ;
- contraintes fortes ;
- non-dérivabilité ;
- temps de calcul de la fonction objectif prohibitif ;
- solution approchée souhaitée ;
- etc.

I.4.1.Tests :

Pour tester une métaheuristique, une première étape consiste à utiliser des fonctions mathématiques spécialement conçues. Les algorithmes sont évalués sur la base d'un ensemble de fonctions, plus ou moins difficiles, puis comparer entre eux.

Les métaheuristiques étant généralement stochastiques, les tests doivent en principe être répétés un grand nombre de fois, puis exploités via des méthodes statistiques. Cependant, cette pratique reste relativement peu répandue dans la littérature spécialisée.

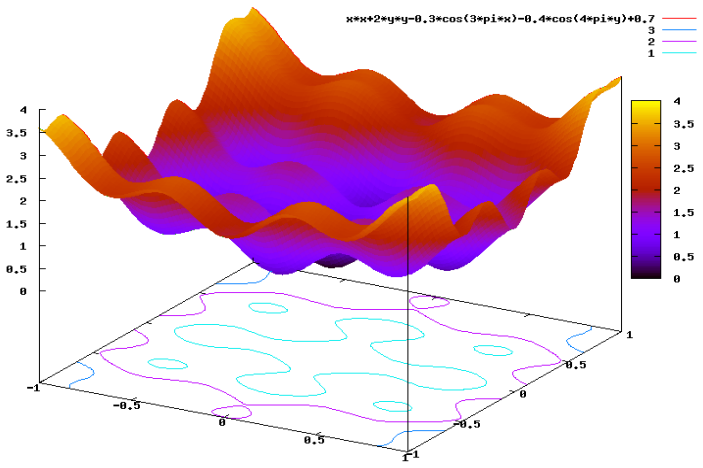

Fig.I.9. Exemple de problème test à minimiser (présenté ici pour deux variables) [121].

I.4.2.Problèmes réels :

Dans un numéro spécial de la revue scientifique European Journal of Operational Research, consacré aux applications des métaheuristiques [5], les éditeurs ont constaté que la majorité des 20 articles publiés le furent dans deux domaines : les problèmes d'ordonnancement et de logistique. Les méthodes les plus utilisées appartiennent à la famille des algorithmes évolutionnaires, souvent hybridés avec des méthodes de recherche locale.

Quelques exemples de problèmes concrets, optimisés via des métaheuristiques :

- problèmes de tournée de véhicules ;
- optimisation de réseaux mobiles UMTS ;
- gestion du trafic aérien [6]
- optimisation des plans de chargement des cœurs de réacteurs nucléaires;
- etc.

I.4.3.Avantages et inconvénients :

Les métaheuristiques étant très généralistes, elles peuvent être adaptées à tout type de problème d'optimisation pouvant se réduire à une « boîte noire ». Elles sont souvent moins puissantes que des méthodes exactes sur certains types de problèmes. Elles ne garantissent pas non plus la découverte de l'optimum global en un temps fini. Cependant, un grand nombre de problèmes réels ne peut être optimisé efficacement par des approches purement mathématiques, les métaheuristiques peuvent alors être utilisées avec profit.

La notion d'efficacité se rapporte généralement à deux objectifs contradictoires : la vitesse et la précision. La vitesse est souvent mesurée en nombre d'évaluations de la fonction objective, qui est la plupart du temps la partie la plus gourmande en temps de calcul. La précision se rapporte à la distance entre l'optimum trouvé par la métaheuristique et l'optimum réel, soit du point de vue de la solution, soit de celui de la valeur. Bien souvent, un algorithme rapide est peu précis, et inversement.

Généralement, un choix doit être fait quant au critère d'arrêt adéquat. Un nombre d'évaluations ou un temps imparti est souvent utilisé, mais on peut également choisir d'atteindre une valeur donnée de la fonction objective (le but étant alors de trouver une solution associée). Il est également possible de choisir des critères dépendants du comportement de l'algorithme, comme une dispersion minimale de la population de points ou un paramètre interne approprié. En tout état de cause, le choix du critère d'arrêt influencera la qualité de l'optimisation.

L'utilisation de métaheuristiques peut paraître relativement simple, en première approche, mais il est souvent nécessaire d'adapter l'algorithme au problème étudié. Tout d'abord, principalement dans le cadre de l'optimisation combinatoire, le choix de la représentation des solutions manipulées peut être crucial. Ensuite, la plupart des métaheuristiques disposent de paramètres dont le réglage n'est pas nécessairement trivial. Enfin, obtenir de bonnes performances passe généralement par une étape d'adaptation des diverses étapes de l'algorithme (initialisation, notamment). En pratique, seuls le savoir-faire et l'expérience de l'utilisateur permettent de gérer ces problèmes.

Il est admis que, d'un point de vue très général, aucune métaheuristique n'est réellement meilleure qu'une autre. En effet, une métaheuristique ne peut prétendre être plus efficace sur tous les problèmes, bien que certaines instances (c'est-à-dire l'algorithme lui-même, mais aussi un choix de paramètres et une implémentation donnée) puissent être plus adaptées que d'autres sur certaines classes de problèmes. Cette constatation est décrite par le théorème du no free lunch [98] (« pas de déjeuner gratuit »).

En dernière analyse, il est parfois possible que le choix de la représentation des solutions, ou plus généralement des méthodes associées à la métaheuristique ait plus d'influence sur les performances que le type d'algorithme lui-même. En pratique, cependant, les métaheuristiques se montrent plus puissantes que les méthodes de parcours exhaustif ou de recherche purement aléatoire.

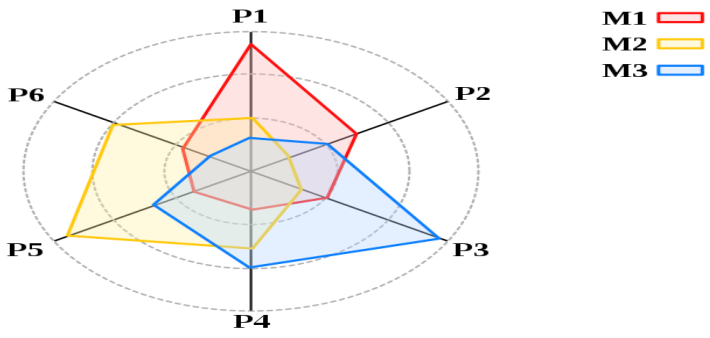

Fig.I.10. Le théorème du « no free lunch » explique qu'aucune instance de métaheuristique ne peut prétendre être la meilleure sur tous les problèmes. Une métaheuristique (M) n'est performante que pour une classe de problème (P) donnée [122].

I.5.Variantes :

I.5.1 Liste de métaheuristiques :

Les métaheuristiques les plus connues sont :

- Les algorithmes évolutionnistes, parmi lesquels :
  - les stratégies d'évolution, [8]
  - les algorithmes génétiques, [12]
  - les algorithmes à évolution différentielle, [27]
  - les algorithmes à estimation de distribution, [26]
  - les systèmes immunitaires artificiels, [17]
  - la recomposition de chemin (*Path relinking en anglais*) [100]
  - Shuffled Complex Evolution [101]
- Le recuit simulé, [14]
- Les algorithmes de colonies de fourmis, [22]
- Les algorithmes d'optimisation par essaims particulaires, [24,25]
- La recherche avec tabous, [16]
- La méthode GRASP. [103]

Il existe un très grand nombre d'autres métaheuristiques, plus ou moins connues :

- L'algorithme du kangourou, [104]
- La méthode de Fletcher et Powell, [105]
- La méthode du bruitage, [108]
- La tunnelisation stochastique, [106]
- l'escalade de collines à recommencements aléatoires, [ref]
- la méthode de l'entropie croisée, [28]
- l'algorithme de recherche d'harmonie [29]
- etc.

La recherche dans le domaine étant très active, il est impossible de produire une liste exhaustive des différentes métaheuristiques d'optimisation. La littérature spécialisée montre un grand nombre de variantes et d'hybridations entre méthodes, particulièrement dans le cas des algorithmes évolutionnaires.

I.5.2 Historique :

- 1952 : premiers travaux sur l'utilisation de méthodes stochastiques pour l'optimisation [7].
- 1954 : Barricelli effectue les premières simulations du processus d'évolution et les utilise sur des problèmes d'optimisation généraux [8].
- 1965 : Rechenberg conçoit le premier algorithme utilisant des stratégies d'évolution [9].
- 1966 : Fogel, Owens et Walsh proposent la programmation évolutionnaire [10].
- 1970 : Hastings propose l'algorithme de Metropolis-Hastings, permettant d'échantillonner n'importe quelle distribution de probabilité [11].
- 1970 : John Horton Conway conçoit le jeu de la vie, l'automate cellulaire le plus connu à ce jour.

- 1975 : travaillant sur les automates cellulaires, Holland propose les premiers algorithmes génétiques [12].
- 1980 : Smith utilise la programmation génétique [13].
- 1983 : s'appuyant sur les travaux d'Hastings, Kirkpatrick, Gelatt et Vecchi conçoivent le recuit simulé [14].
- 1985 : indépendamment de ceux-ci, Černý propose le même algorithme [15].
- 1986 : La première mention du terme méta-heuristique est faite par Fred Glover, lors de la conception de la recherche tabou [16] :

  « La recherche avec tabou peut être vue comme une "méta-heuristique", superposée à une autre heuristique. L'approche vise à éviter les optimums locaux par une stratégie d'interdiction (ou, plus généralement, de pénalisation) de certains mouvements. »

- 1986 : Farmer, Packard et Perelson travaillent sur les systèmes immunitaires artificiels [17].
- 1988 : la première conférence sur les algorithmes génétiques est organisée à l'université de l'Illinois à Urbana-Champaign.
- 1988 : des travaux sur le comportement collectif des fourmis trouvent une application en intelligence artificielle [18].
- 1988 : Koza dépose son premier brevet sur la programmation génétique [19].
- 1989 : Goldberg publie un des livres les plus connus sur les algorithmes génétiques [20].
- 1989 : *Evolver*, le premier logiciel d'optimisation par algorithmes génétiques est publié par la société *Axcelis*.
- 1989 : le terme *algorithme mémétique* apparait [21].
- 1991 : Les algorithmes de colonie de fourmis sont proposés par Marco Dorigo, dans sa thèse de doctorat [22].
- 1993 : le terme « Evolutionary Computation » (calcul évolutionnaire en français) se répand, avec la parution de la revue éponyme, publiée par le Massachusetts Institute of Technology.
- 1995 : Feo et Resende proposent la méthode GRASP (pour *Greedy randomized adaptive search procedure*s, « procédure de recherche gloutonne aléatoire adaptative » en français) [23].
- 1995 : Kennedy et Eberhart conçoivent l'optimisation par essaims particulaires [24,25].
- 1996 : Mühlenbein et Paaß proposent les algorithmes à estimation de distribution [26].
- 1997 : Storn et Price proposent un algorithme à évolution différentielle [27].
- 1997 : Rubinstein conçoit la méthode de l'entropie croisée [28].
- 1999 : Boettcher propose l'optimisation extrémale [29].
- 2000 : premiers algorithmes génétiques interactifs [30].
- 2013 : algorithme basé sur le comportement de chasse chez les manchots: Penguins Search Optimization Algorithm (PeSOA), Recent Trends in Applied Artificial Intelligence Lecture Notes in Computer Science Volume 7906, 2013, p. 222–231.

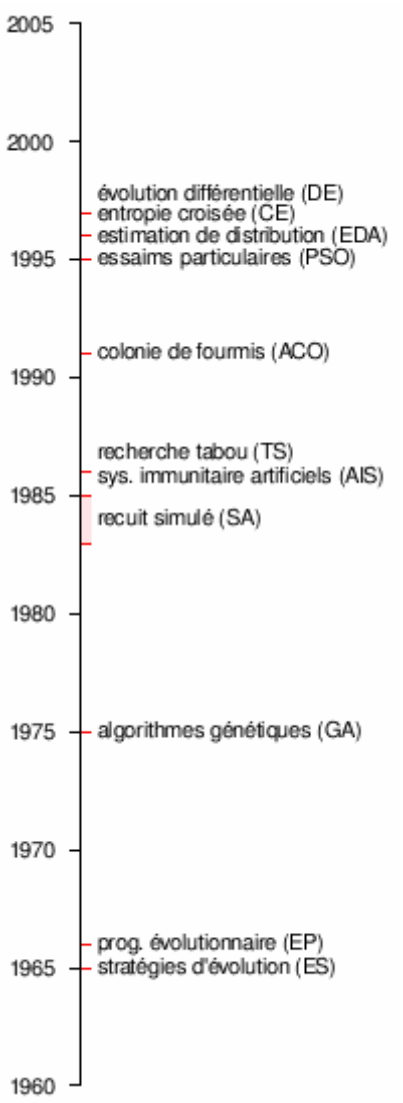

Fig.I.11. Chronologie des principales métaheuristiques, le nom est indiqué suivi de l'acronyme anglais entre parenthèses [123].

I.5.3.Extensions :

Les métaheuristiques, de par leur nature flexible, se prêtent particulièrement à des extensions. On peut ainsi trouver des adaptations pour des problèmes plus complexes que l'optimisation mono-objectif :

- problèmes multiobjectifs (plusieurs objectifs contradictoires doivent être optimisés de concert) [109],
- optimisation multimodale (plusieurs optimums doivent être trouvés) [109],
- optimisation en environnement incertain (un bruit est présent sur la valeur renvoyée par la fonction objectif, ou sur le passage des paramètres à celle-ci) [109],
- hybridations entre métaheuristiques [110],

- hybridations avec des méthodes exactes [110],
- problèmes dynamiques (la fonction objective change durant le temps de l'optimisation),

I.6.CONCOLUSION :

Ce chapitre a décrit des généralités sur les métaheuristiques et a montré leurs multiples facettes proposées dans la littérature depuis 50 ans de recherche pour la résolution approchée des problèmes d'optimisation difficile. Le succès de ces méthodes ne doit pas masquer la principale difficulté à laquelle est confronté l'utilisateur. En présence d'un problème d'optimisation concret, le choix d'une méthode "efficace" capable de produire une solution "optimale" ou de qualité acceptable au prix d'un temps de calcul "raisonnable" peut donner de mauvais résultats pour un autre.

Face à ce souci, la théorie n'est pas encore d'un grand secours, car les théorèmes de convergence sont souvent inexistants ou applicables seulement sous des hypothèses très restrictives. En outre, le réglage "optimal" des divers paramètres d'une métaheuristique, qui peut être préconisé par la théorie, est souvent inapplicable en pratique, car il induit un coût de calcul prohibitif. En conséquence, le choix d'une bonne méthode et le réglage des paramètres de celle-ci font généralement appel au savoir-faire et à l'expérience de l'utilisateur plutôt qu'à l'application fidèle de règles bien établies.

Dans le chapitre suivant, nous allons présenter l'algorithme de recherche gravitationnelle (GSA) qui est l'objet, ou la matière première, de ce mémoire.

CHIPITRE II
Recherche Gravitationnelle

II.1.Introduction

Dans la résolution de problèmes d'optimisation avec un espace de recherche à grande dimension, les algorithmes d'optimisation classiques ne fournissent pas une solution appropriée, car l'espace de recherche augmente de façon exponentielle avec la taille du problème, résolvant ainsi les problèmes d'utilisation de techniques exactes (telles que la recherche exhaustive) ne sont pas pratiques.

Au cours des dernières décennies, il y a eu un intérêt croissant pour les algorithmes inspirés par les comportements des phénomènes naturels [43, 35, 38, 47, 49, 51,62]. De nombreux chercheurs ont montré que ces algorithmes sont bien adaptés pour résoudre des problèmes complexes. Des problèmes tels que l'optimisation des fonctions objectives [36,66], la reconnaissance de formes [54,61], théorie du contrôle [32, 46,50], le traitement d'image [34,57], modélisation des filtres [45,53], etc. Diverses approches ont été adoptées par les chercheurs, par exemple l'algorithme génétique [62], recuit simulé [51], algorithme de recherche de colonies de fourmis [35], optimisation de l'essaim de particules [47], etc. Ces algorithmes sont amplement analysés par leur auteur et appliqués dans de nombreux domaines [31, 33, 37, 42, 55,64]. Ces algorithmes résolvent différents problèmes d'optimisation. Cependant, il n'y a pas d'algorithme spécifique pour obtenir la meilleure solution pour tous les problèmes. Certains algorithmes donnent une meilleure solution pour certains problèmes particuliers que d'autres. Par conséquent, la recherche de nouveaux algorithmes d'optimisation heuristique est un problème ouvert [65].

Dans ce chapitre, on va présenter une approche d'optimisation basée sur la loi de la gravité, à savoir *l'algorithme de recherche gravitationnelle* (GSA) est proposé [58]. Cet algorithme est basé sur la gravité newtonienne: «Chaque particule dans l'univers attire toutes les autres particules avec une force qui est directement proportionnelle au produit de leurs masses et inversement proportionnelle au carré de la distance entre elles».

II.2.Une brève revue des algorithmes heuristiques

Le mot «heuristique» est grec et signifie «savoir», «trouver», «découvrir» ou «guider une enquête» [52]. Plus précisément, «les heuristiques sont des techniques qui recherchent de bonnes solutions (quasi-optimales) à un coût de calcul raisonnable sans pouvoir garantir la faisabilité ou l'optimalité, ou même dans de nombreux cas indiquer

à quel point une solution réalisable est proche de l'optimalité. Russell et Norvig, 1995. [59]

Les algorithmes heuristiques imitent les processus physiques ou biologiques. Certains des plus célèbres de ces algorithmes sont l'algorithme génétique, Recuit simulé [51], système immunitaire artificiel [38], optimisation des colonies de fourmis, optimisation de l'essaim de particules [47] et l'algorithme de recherche bactérienne [41]. Genetic Algorithm, GA, sont inspirés de la théorie évolutionniste darwinienne [62], Simulated Recuit, SA, est conçu par l'utilisation des effets thermodynamiques [51], systèmes immunitaires artificiels, AIS, simuler biologique systèmes immunitaires [38], Ant Colony Optimization, ACO, imite le comportement des fourmis à la recherche de nourriture [35], Bacterial Foraging Algorithme, BFA, provient de la recherche et du butinage optimal des bactéries [41,49] et l'optimisation des essaims de particules, PSO, simule le comportement du troupeau d'oiseaux [33,47].

Tous les algorithmes heuristiques mentionnés ci-dessus ont un comportement stochastique. Cependant, Formato [39,40] a proposé l'algorithme de recherche heuristique déterministe basé sur la métaphore de la cinématique gravitationnelle qui est appelée Central Force Optimization, CFO.

Dans certains algorithmes stochastiques, comme SA, la recherche commence à partir d'un seul point et se poursuit de façon séquentielle, cependant, la plupart des algorithmes heuristiques recherchent parallèlement plusieurs points initiaux, par ex. algorithmes basés sur des essaims utilisent une collection d'agents similaires à un troupeau naturel d'oiseaux ou de poissons.

Dans un algorithme basé sur un essaim, chaque membre exécute une série d'opérations particulières et partage ses informations avec d'autres. Ces opérations sont presque très simples, mais leur effet collectif, connu sous le nom *d'intelligence d'essaim* [35,63], produit un résultat surprenant. Les interactions locales entre les agents fournissent un résultat global qui permet au système de résoudre le problème sans utiliser de contrôleur central. C'est ce que l'on appelle phénomène d'auto-organisation [35].

On peut reconnaître deux aspects communs dans les algorithmes heuristiques basés sur la population: l'exploration et l'exploitation. L'exploration est la capacité de parcourir n'importe quel point de l'espace de recherche, où l'exploitation est la capacité de trouver l'optimum autour d'une bonne solution. L'exploitation est ce qu'on appelle

aussi intensification. Dans les premières itérations, un algorithme de recherche heuristique explore l'espace de recherche pour trouver de nouvelles solutions. Pour éviter le piège dans un optimum local, l'algorithme doit utiliser une bonne stratégie d'exploration dans les premières itérations. Par conséquent, l'exploration est un problème important dans un algorithme heuristique basé sur la population. À la fin des itérations, l'exploration s'estompe et l'exploitation s'estompe, donc l'algorithme se règle en points semi-optimaux. Pour avoir une recherche de haute performance, une clé essentielle est un compromis entre exploration et exploitation. Cependant, tous les algorithmes heuristiques basés sur la population utilisent l'exploration et les aspects d'exploitation, mais ils utilisent des approches et des opérateurs différents.

D'un point de vue différent, les membres d'une population d'un algorithme de recherche basé sur la population passent trois étapes dans chaque itération pour réaliser les concepts d'exploration et d'exploitation: auto adaptation, coopération et compétition. Dans l'étape de l'auto adaptation, chaque membre (agent) améliore ses performances. Dans l'étape de coopération, les membres collaborent les uns avec les autres par transfert de l'information. Enfin, dans l'étape de la compétition, les membres rivalisent pour survivre. Ces étapes ont généralement des formes stochastiques, et pourraient être réalisées de différentes manières. Ces étapes, inspirées de la nature, sont les idées principales des algorithmes basés sur des populations. Ces concepts guident un algorithme pour trouver un optimum global.

Cependant, tous les algorithmes de recherche basés sur la population donnent des résultats assez satisfaisants, mais il n'existe pas d'algorithme heuristique pourront fournir une performance supérieure à d'autres dans la résolution de tous les problèmes d'optimisation. En d'autres termes, un algorithme peut résoudre un certain nombre de problèmes et fournir des résultats mauvais pour d'autres [65]. Par conséquent, la proposition de nouveaux algorithmes heuristiques hauts performance est toujours demandée. Nous présentons dans ce chapitre un algorithme récemment proposé et inspiré de la gravité. Dans la section suivante, un bref aperçu de la force gravitationnelle est donné pour fournir un contexte approprié et suivi par l'explication de GSA.

II.3.La loi de la gravité

La gravitation est la tendance des masses à accélérer les unes vers les autres. C'est l'une des quatre interactions fondamentales de la nature [60] (les autres sont: la force électromagnétique, la force nucléaire faible et la force nucléaire forte). Chaque particule dans

l'univers attire toutes les autres particules. La gravité est partout. L'inéluctabilité de la gravité la rend différente de toutes autres forces naturelles. La façon dont la force gravitationnelle de Newton se comporte est appelée «*action à distance*». Cela signifie que la gravité agit entre des particules séparées sans intermédiaire et sans délai. Dans la loi de Newton, chaque particule attire toutes les autres particules avec une «force gravitationnelle» [44,60]. La force gravitationnelle entre deux particules est directement proportionnelle au produit de leurs masses et inversement proportionnelle au carré de la distance entre eux [14]:

$$F = G\frac{M_1 M_2}{R^2} \tag{II.1}$$

Où F est l'amplitude de la force gravitationnelle, G est la constante gravitationnelle, M1 et M2 sont la masse, et R est la distance entre les deux particules.

La deuxième loi de Newton dit que lorsqu'une force, F, est appliquée à une particule, son accélération, *a*, ne dépend que de la force et de sa masse, M [44]:

$$a = \frac{F}{M} \tag{II.2}$$

Basé sur (1) et (2), il y a une force de gravité attirante parmi toutes les particules. Une augmentation de la distance entre deux particules signifie une diminution de la force de gravité entre les deux comme il est illustré sur la **Fig. II. 1**. Sur cette figure, F1j est la force qui agit sur M1 à partir de Mj. F1 est la force globale qui agit sur M1 et provoque le vecteur d'accélération a1. De plus, en raison de l'effet de diminution de la gravité, la valeur réelle de la «constante gravitationnelle» dépend de l'âge de l'univers. **Eq.II.3** donne la diminution de la constante gravitationnelle, G, avec l'âge [56]:

$$G(t) = G(t_0) \times (\frac{t_0}{t})^{\beta}, \beta < 1, \tag{II.3}$$

Où G (t) est la valeur de la constante gravitationnelle au temps t. G (t_0) est la valeur de la constante gravitationnelle au premier cosmique intervalle quantique du temps t_0 [56]. Trois types de masses sont définis en physique théorique:

II.3.1. La masse gravitationnelle active :

La masse gravitationnelle active (Ma), est une mesure de la force du champ gravitationnel due à un objet particulier. Le champ gravitationnel d'un objet avec une petite masse gravitationnelle active est plus faible que l'objet avec une masse gravitationnelle plus active.

II.3.2. La masse gravitationnelle passive :

La masse gravitationnelle passive (MP) est une mesure de la force de l'interaction d'un objet avec le champ gravitationnel. Dans le même champ gravitationnel, un objet avec une masse gravitationnelle passive plus petite subit une force plus petite qu'un objet avec une masse gravitationnelle passive plus grande.

II.3.3. La masse inertielle, Mi :

La masse inertielle est une mesure de la résistance d'un objet à la modification de son état de mouvement lorsqu'une force est appliquée. Un objet avec une grande masse inertielle change plus lentement son mouvement, et un objet à petite masse inertielle le change rapidement.

Maintenant, en considérant les aspects mentionnés ci-dessus, nous réécrivons les lois de Newton. La force gravitationnelle, Fij, qui agit sur la masse i par la masse j, est proportionnelle au produit de la pesanteur active de la masse j et de la pesanteur passive de la masse i, et inversement proportionnelle à la distance carrée entre elles. a_i est proportionnel à la force Fij et inversement proportionnel à la masse d'inertie. Plus précisément, on peut réécrire les Eqs. (II.1) et (II.2) comme suit:

$$F_{ij} = G\frac{M_{aj} \times M_{pi}}{R^2}, \tag{II.4}$$

$$a_i = \frac{F_{ij}}{M_{ii}} \quad , \tag{II.5}$$

Où Maj et Mpi représentent respectivement la masse gravitationnelle active de la particule j et la masse gravitationnelle passive de la particule i, et Mii représente la masse d'inertie de la particule i.

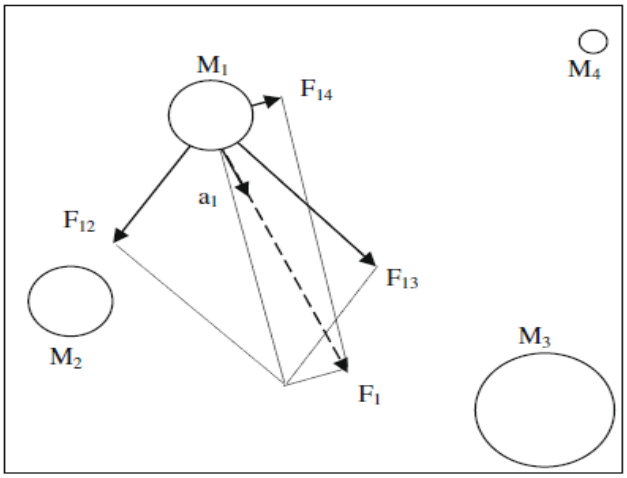

Fig. II. 1. Chaque masse accélère vers la force résultante qui l'agit des autres masses [124].

Bien que la masse inertielle, la masse gravitationnelle passive et la masse gravitationnelle active soient conceptuellement distinctes, aucune expérience n'a toujours démontré sans ambiguïté aucune différence entre elles. La théorie de la relativité générale repose sur l'hypothèse que la masse gravitationnelle inertielle et passive est équivalente. C'est ce qu'on appelle le principe d'équivalence faible [48]. La norme la relativité générale suppose aussi l'équivalence de la masse inertielle et de la masse gravitationnelle active; cette équivalence est parfois appelée le principe d'équivalent fort [48].

II.4.Algorithme de recherche gravitationnelle (GSA)

Dans cette section, nous présentons un algorithme d'optimisation basé sur la loi de la gravité [58]. Dans l'algorithme proposé, les agents sont considérés comme des objets et leur performance est mesurée par leur masse. Tous ces objets s'attirent par la force de la gravité, et cette force provoque un mouvement global de tous les objets vers les objets avec des masses plus lourdes. Par conséquent, les masses coopèrent en utilisant une forme directe de communication, par la force gravitationnelle. Les masses lourdes - qui correspondent aux bonnes solutions - se déplacent plus lentement que les plus légères, ceci garantit l'étape d'exploitation de l'algorithme.

En GSA, chaque masse (agent) a quatre spécifications: position, masse inertielle, masse gravitationnelle active, et gravitationnelle passive. La position de la masse, et ses masses gravitationnelles et inertielles sont déterminées en utilisant une fonction objective.

En d'autres termes, chaque masse présente une solution, et l'algorithme est navigué en ajustant correctement la gravité et la masse d'inertie. Au fil du temps, nous nous attendons à ce que les masses soient attirées par la masse la plus lourde.

La GSA pourrait être considérée comme un système isolé de masses. C'est comme un petit monde artificiel de masses obéissant aux lois newtoniennes de la gravitation et du mouvement. Plus précisément, les masses obéissent aux lois suivantes:

II.4.1. Loi de la gravité:

Chaque particule attire toutes les autres particules. La force gravitationnelle entre deux particules est directement proportionnelle au produit de leurs masses et inversement proportionnelle à la distance qui les sépare, R. Nous utilisons ici R au lieu

de R^2, parce que, d'après les résultats expérimentaux, R donne de meilleurs résultats que R^2 dans tous les cas expérimentaux.

II.4.2 Loi de mouvement:

La vitesse actuelle de toute masse est égale à la somme de la fraction de sa vitesse précédente et de la variation dans la vitesse. La variation de la vitesse ou l'accélération de n'importe quelle masse est égale à la force exercée par le système divisé par la masse inertielle.

Maintenant, considérons un système avec N agents (masses). Nous définissons la position de l'agent i par:

$$X_i = (x_i^1, ..., x_i^d, ..., x_i^n) \quad \text{Pour i=1,2,..., N,} \tag{II.6}$$

Où x_i^d représente la position de l'agent *i* dans la direction (dimension) *d*.

À un instant donné 't', nous définissons la force agissant sur la masse 'i' à partir de la masse 'j' comme suit:

$$F_{ij}^d(t) = G(t)\frac{M_{pi}(t) \times M_{aj}(t)}{R_{ji}(t) + \varepsilon}(x_j^d(t) - x_i^d(t)), \tag{II.7}$$

Où Maj est la masse gravitationnelle active liée à l'agent j, Mpi est la masse gravitationnelle passive liée à l'agent i, G (t) est constante gravitationnelle à l'instant t, ε est une petite constante, et Rij (t) est la distance euclidienne entre l'agent i et j :

$$R_{ij}(t) = \left\| X_i(t), X_j(t) \right\|_2. \tag{II.8}$$

Pour donner une caractéristique stochastique GSA, nous supposons que la force totale qui agit sur l'agent i dans une direction d soit une somme pondérée aléatoirement des composantes des forces exercées par d'autres agents:

$$F_i^d(t) = \sum_{j=1, j\neq i}^{N} rand_j F_{ij}^d(t), \tag{II.9}$$

Ou $rand_j$ est un nombre aléatoire dans l'intervalle [0,1]._

D'où, par la loi du mouvement, l'accélération de l'agent i à l'instant t, et dans la direction d, $a_i^d(t)$ est donnée comme suit:

$$a_i^d(t) = \frac{F_i^d(t)}{M_{ii}(t)}, \tag{II.10}$$

Où Mii est la masse inertielle de l'agent i.

De plus, la vitesse postérieure d'un agent est considérée comme la fraction de sa vitesse courante ajoutée à son accélération. Par conséquent, sa position et sa vitesse pourraient être calculées comme suit:

$$v_i^d(t+1) = rand_i \times v_i^d(t) + a_i^d(t), \tag{II.11}$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1), \tag{II.12}$$

Où $rand_i$ est une variable aléatoire uniforme dans l'intervalle [0, 1]. Ce nombre aléatoire est utilisé en GSA pour donner une caractéristique randomisée à la recherche.

La constante gravitationnelle, G, est initialisée au début et sera réduite avec le temps pour contrôler la précision de la recherche. En d'autres termes, G est une fonction de la valeur initiale (G_0) et du temps (t):

$$G(t) = G(G_0, t). \tag{II.13}$$

Les masses gravitationnelles et d'inertie sont simplement calculées par l'évaluation de la fonction fitness. Une masse plus lourde signifie un agent plus efficace. Cela signifie que les meilleurs agents ont des attractions plus élevées et se mouvaient plus lentement. En supposant l'égalité de la gravité et la masse d'inertie, les valeurs de masses sont calculées en utilisant la carte de la fitness. Nous mettons à jour la masse gravitationnelle et inertielle par les équations suivantes:

$$M_{ai} = M_{pi} = M_{ii} = Mi, \quad \text{Avec i=1,2,..., N,} \tag{II.14}$$

$$m_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)}, \tag{II.15}$$

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)}, \tag{II.16}$$

Où $fit_i(t)$ représente la valeur de fitness de l'agent i à l'instant t, et, worst (t) et best (t) sont définis comme suit (pour la minimisation d'un critère):

$$best(t) = \min_{j\in\{1,\ldots,N\}} fit_j(t), \tag{II.17}$$

$$worst(t) = \max_{j\in\{1,\ldots,N\}} fit_j(t), \tag{II.18}$$

Il est à noter que pour un problème de maximisation, Eqs. (17) et (18) sont remplacés par Eqs. (19) et (20), respectivement:

$$best(t) = \max_{j\in\{1,\ldots,N\}} fit_j(t), \tag{II.19}$$

$$worst(t) = \min_{j\in\{1,\ldots,N\}} fit_j(t), \tag{II.20}$$

Nous rappelons que pour éviter que l'algorithme soit piégé dans un optimum local, l'algorithme doit utiliser l'exploration au début. Par l'expiration progressive des itérations, l'exploration doit disparaître et l'exploitation prend plus place pour disparaitre elle aussi à la fin. Pour améliorer la performance de GSA en contrôlant exploration et exploitation seuls les agents Kbest attireront les autres. Kbest est une fonction du temps, avec la valeur initiale K_0 au début, décroissante avec le temps. De cette façon, au début, tous les agents appliquent la force, et comme le temps passe, Kbest est diminué linéairement et à la fin il y aura juste un agent appliquant la force aux autres. Par conséquent, Eq. (9) pourrait être modifié comme:

$$F_i^d(t) = \sum_{j \in Kbest,\, j \neq i}^{N} rand_j F_{ij}^d(t), \qquad \text{(II.21)}$$

Où Kbest est l'ensemble des premiers agents K avec la meilleure valeur de fitness et la plus grande masse. Les différentes étapes de l'algorithme proposé sont les suivants:

(a) Initialisation randomisée.
(b) Évaluation de la fitness des agents.
(c) Mettre à jour G (t), best (t), worst (t) et M*i* (t) pour i = 1,2,. . ., N.
(d) Calcul de la force totale dans différentes directions.
(e) Calcul de l'accélération et de la vitesse.
(f) Mise à jour de la position des agents.
(g) Répétez les étapes c à g jusqu'à ce que le critère d'arrêt soit atteint.
(h) Fin.

Le principe de GSA est montré dans la **Fig.II.2**.

Pour souligner l'efficacité du GSA, quelques remarques sont notées:

- Puisque chaque agent *peut observer* les performances des autres, la force gravitationnelle est un outil de transfert d'information.
- En raison de la force qui agit sur un agent par les agents de son voisinage, il peut voir l'espace autour de lui.
- Une masse lourde a un grand rayon d'attraction effectif et donc une grande intensité d'attraction. Par conséquent, les agents avec un des performances plus élevées ont une plus grande masse gravitationnelle. En conséquence, les autres agents ont tendance à se déplacer vers le meilleur agent.

E. Rashedi et al. / Information Sciences 179 (2009) 2232–2248

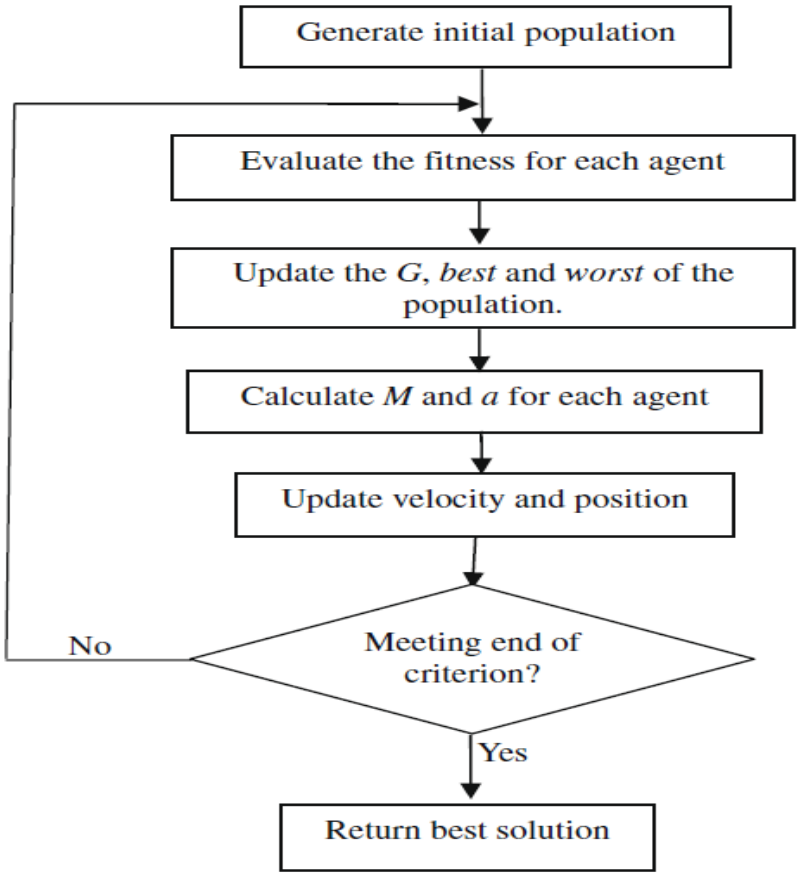

Fig.II.2 Principe de GSA

- La constante gravitationnelle ajuste la précision de la recherche, de sorte qu'elle diminue avec le temps (similaire à la température dans l'algorithme de recuit simulé).
- Ici, nous supposons que les masses gravitationnelles et d'inertie sont les mêmes. Cependant, pour certaines applications différentes valeurs pour eux peuvent être utilisées. Une plus grande masse d'inertie fournit un mouvement plus lent des agents dans l'espace de recherche et donc plus de précision. Inversement, une plus grande masse gravitationnelle provoque une plus grande attraction des agents. Cela permet une convergence plus rapide.

Le Pseudo Code de la version standard de l'algorithme de recherche gravitationnelle est le suivant :

Algorithm: The standard gravitational search algorithm

Parameters initialization
Set the initial values of gravitation constant G0, α and ξ
set the initial iteration t=0
Initial population
For i=1: i ≤N **do**
    Generate an initial population Xi (t) randomly, where $X_i = (x_i^1, ..., x_i^d, ..., x_i^n)$
end for
while termination criteria == 0
    Solutions evaluation
    Evaluate the fitness function f (xi (t)) for each agent in the population X (t).
    Assign the best, worst agent in the population X (t).
    Solution update
    Update the gravitational constant G as shown in equation 1.
    for i=1:i≤N **do**
        for j=i+1: j<N **do**
            Calculate the force action on agent i from agent j as shown in equation 4.
        end for
        Calculate the total force that act on agent i from agent j as shown in equation 7.
        Calculate the inertial mass Mi. as shown in equation 15, 16.
        Calculate the acceleration of the agent i as shown in equation 10.
        Update the velocity of agent i as shown in equation 11.
        Update de position of agent i as shown in equation 12.
    end
    Set t =t+1
end while
Produce the best solution
Return the best solution.

Fig.II.3. Pseudocode de GSA

II.6. CONCLUSION

Au cours des dernières années, diverses méthodes d'optimisation heuristique ont été développées. Certains de ces algorithmes sont inspirés par les comportements d'essaim dans la nature. Dans ce chapitre, nous avons exposé un algorithme d'optimisation appelé algorithme de recherche gravitationnelle (GSA). GSA est construit sur la base de la loi de Gravité et de la notion d'interaction entre les masses. L'algorithme GSA utilise la théorie de la physique newtonienne et ses agents de recherche sont un ensemble de masses. En GSA, nous avons un système isolé de masses. En utilisant la force gravitationnelle, chaque masse dans le système peut voir la situation

d'autres masses. La force gravitationnelle est donc un moyen de transférer des informations entre différentes masses. Dans ce qui suite on va présenter une nouvelle variante d'algorithme GSA basé sur la notion du compactage de l'information relative aux masses. Ce nouvel algorithme est appelé Compact Gravitational Search Algorithm (cGSA). Pour ce faire, nous allons présenter, tout d'abord, la représentation compacte.

CHAPITRE III

La représentation compacte

III.1.Introduction

La représentation compacte a été proposée pour réduire les capacités de calcul requises par les algorithmes basés sur des populations. Ces derniers nécessitent généralement la mémorisation de la position de chaque particule, ou individu. Dans ce chapitre, on va présenter plusieurs variantes d'algorithmes compacts. Nous traiterons le cas binaire ainsi que le cas réel. Entre autres, nous présenterons le cGA, le cFA et cPSO.

III.2.La représentation compacte :

Ce champ a commencé en 1999 quand Harik *et al* ont proposé le premier algorithme génétique compact (cGA) [67]. En tant qu'algorithme à estimation de distributions (EDA) [68, 69] qui est une métaheuristiques inspirée des algorithmes génétiques. Ils sont utilisés pour résoudre des problèmes d'optimisation, via la manipulation d'un échantillonnage de la fonction décrivant la qualité des solutions possibles. Comme toutes les métaheuristiques utilisant une population de points, ils sont itératifs. À l'inverse des algorithmes évolutionnaires « classiques », le cœur de la méthode consiste à estimer les relations entre les différentes variables d'un problème d'optimisation, grâce à l'estimation d'une distribution de probabilité. Ils n'emploient donc pas d'opérateurs de croisement ou de mutation, l'échantillon étant directement construit à partir des paramètres de distribution, estimés à l'itération précédente.

Le cGA génère des descendants selon un modèle probabiliste au lieu d'utiliser des opérateurs de recombinaison et de mutation. La population est représentée par le vecteur de probabilité (**PV**). 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 **PV**. 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. Si le bit j du gagnant est 1 et que celui du perdant est 0, la composante j de PV sera augmentée de 1 / N, où Np est la taille hypothétique de la population simulée. Inversement, il sera diminué avec 1 / N. Lorsque le $j^{ème}$ bit du gagnant est égal au $j^{ème}$ bit du perdant, la $j^{ème}$ composante du PV ne sera pas corrigée. Le pseudocode de cGA est présenté sur la **Fig.III.3**

III.2.1.Exemple de cas binaire

Premièrement, l'algorithme génère le vecteur de probabilité **PV,** on suppose que la dimension de ce vecteur est égale à 3 :

$$\mathbf{PV = [}0{,}5 \;\; 0{,}5 \;\; 0{,}5\mathbf{]} \tag{III.1}$$

Puis l'algorithme génère aléatoirement deux individus G1 et G2, avec la même dimension que **PV.** Donc qui est égal à 3 :

$$\mathbf{G1 = [1 \;\; 0 \;\; 1]} \tag{III.2}$$

$$\mathbf{G2 = [0 \;\; 1 \;\; 1]} \tag{III.3}$$

Nous supposons ici que G1 et le winner (gagnant) et G2 et looser (perdant). L'adaptation du vecteur de probabilité PV comme indiqué sur la Fig. III.1 :
On a G1 [1] = 1 et G2 [1]= 0. L'algorithme calcule la nouvelle valeur du PV [1]. Il rajoute à l'ancienne valeur du PV un petit chiffre (1/N), où N est un paramètre,

$$PV\,[1] = [0,\,5] + 1 / N \tag{III.4}$$

Fig. III. 1. Adaptation du PV - Le Cas binaire

Passent maintenant au deuxième chiffre des deux vecteurs G1 [2] = 0 et G2 [2]= 1, la nouvelle valeur du PV comme l'étape décrite précédemment, on diminue à

Adaptation du PV - Le Cas Binaire

```
1.      /* update the probability vector toward the best one */
2.    for i=1 : L  do
3.      if winner[i]≠loser[i] then
4.        if winner[i]==1 then
5.            p[i] = p[i] + 1/N
6.        else
7.            p[i] = p[i] - 1/N
8.        end if
9.      end if
10.   end for
```

l'ancienne valeur du PV un petit chiffre (1/ N), avec N la taille de la population, et dans notre exemple N=3 :

$$\mathbf{PV}\,[1] = [0{,}5] - 1 / N \tag{III.5}$$

III.2.2.Cas réel

L'idée de base consiste à représenter les individus par une densité de probabilité (PDF). On utiliser une distribution gaussienne ou pour plus de simplicité une distribution uniforme.

Chaque variable décisionnelle *i* du problème de dimension D à résoudre, est représentée par une fonction probabiliste. Une densité de probabilité normale, ou gaussienne, définie dans l'intervalle [-1, 1] est donnée par la formule suivante :

$$PDF(\mu\ [i],\sigma[i]) = \frac{e^{\frac{(x-\mu[i])^2}{2\sigma[i]^2}}\sqrt{\frac{2}{\pi}}}{\sigma[i](erf(\frac{\mu[i]+1}{\sqrt{2}\sigma[i]}) - erf(\frac{\mu[i]-1}{\sqrt{2}\sigma[i]}))} \quad \text{(III. 6)}$$

Où μ[i] et σ[i] sont, respectivement, la moyenne et l'écart-type de la variable *i*, avec i=1, 2 … D.

En utilisant cette représentation, la population est réduite à un vecteur de probabilité **PV** défini comme suivant :

$$PV^t = [\mu^t \ , \quad \sigma^t] \quad \text{(III. 7)}$$

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 individus, nous avons sa représentation compacte dans **PV**. Nous représentons sur la **Fig. III. 2** une densité de probabilité gaussienne (PDF) et sa fonction cumulative CDF.

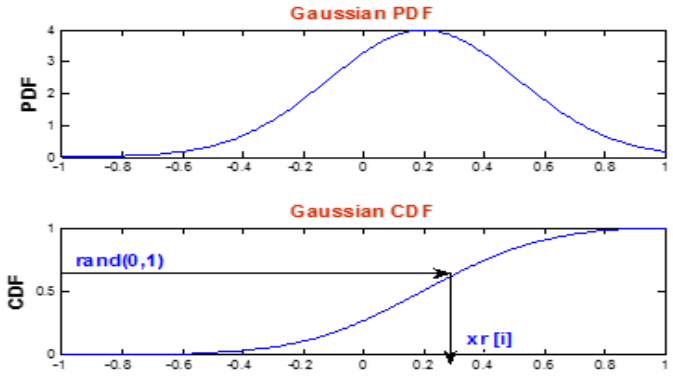

Fig. III.2. Représentation graphique d'une PDF normale et de sa CDF

Pour générer un individu de PV, nous générons de chaque fonction PV [i], Ième variable notée x[i]. Ceci est illustré sur la **Fig. III. 2**. Pour 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, correspondant à rand (0, 1), est ensuite calculée. Cette valeur, notée xr [i], doit être convertie à l'intervalle [a, b] par :

$$x[i] = (b-a)\frac{xr[i]+1}{2} + a \quad \text{(III. 8)}$$

Comme suggéré dans [70], 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 lucioles sont générées. Puis, un tournoi est effectué entre elles. Par conséquent, par rapport au critère d'optimisation, un gagnant et un perdant sont obtenus. Selon leurs allèles 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 [70]:

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

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

III.3.Variantes d'algorithmes compacts :

III.3.1.Algorithme génétique binaire compact :

Algorithme proposé par Harik en 1998. C'est la même procédure décrite en haut. Il utilise un vecteur de probabilité PV qui est adapté, après un tournoi désignant un gagnant et perdant, par la procédure décrite dans **Fig .III. 1.** Le pseudocode complet est donné dans la **Fig. III.3**

Algorithme 1: 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 individuals a and b */
9.      a=generate(p), b=generate(p)
10.     /* let them compete */
11.     [winner, loser]= compete(a,b)
12.       /* update the probability vector toward 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. P represents the final solution
```

Fig. III. 3. Pseudocode de bcGA binaire.

En 2008, Ernesto Mininno *et al* [70] ont proposé le premier cGA qui fonctionne avec des valeurs réelles. Leur variante est appelée algorithme génétique compact à valeurs réelles (rcGA). Il donne de meilleurs résultats que cGA binaire [70] tout en réduisant significativement le coût de calcul en évitant même les opérations supplémentaires liées aux conversions du binaire vers le réel. Le rcGA utilise une fonction de densité de probabilité (PDF : *Probability of Density Function*), par ex. une PDF gaussienne, pour représenter les solutions. Le **PV** des bcGA codées en binaire contient les probabilités pour que chaque allèle d'être égal à 0 ou à 1. De l'autre côté, le **PV** de rcGA contient la moyenne et l'écart-type de chaque allèle.

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 [71]. En 2012, la version compacte de PSO (cPSO) a également été proposée [72]. En 2013, Yang *et al* ont proposé un algorithme compact basé sur l'enseignement et l'apprentissage des élèves en classe appelé cTLBO [73]. En 2014, A.S. Soares *et al* ont proposé un algorithme génétique compact basé sur la mutation [75]. En 2016, B. V. Ha *et al* [76] ont proposé un nouvel algorithme génétique compact modifié qui fonctionne avec plus d'un vecteur de probabilité. Les auteurs ont proposé deux règles de mise à jour: une mise à jour locale et une mise à jour globale.

Notre grand intérêt pour l'optimisation compacte est du à trois faits. En préambule, 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 [77-78].

III.3.2.The compact harmony search algorithm:

La recherche harmonique est une méthode inspirée de la musique. Cette algorithme es noté *HSA : harmony search algorithm*. Dans la littérature, plusieurs variantes sont proposées de HSA. On peut citer: l'hybridation avec d'autres algorithmes évolutionnaire, comme l'optimisation par essaims de particules PSO [79], l'algorithme génétique [80], Simplex-Harmony Search (SHS) [81], etc., Mahdavi, Fesanghary et Damangir [82] ont proposé le Improved Harmony Search (IHS). Omran et Mahdavi [83] proposés le « global HS » (SGH). Le HSA est connu pour être performant et par conséquent il est appliqué dans de nombreux domaines très différents. On peut citer: la transmission de données [84], l'exploration de données [85], réseaux sans fil [86] pour ne citer que quelques-uns.

Une autre version compacte a été proposée à l'université de béjaia par Farid Lachouri et al [88] dans le cadre de leur mémoire de licence. L'algorithme proposé est appelé ***compact harmony search algorithm*** (cHSA). Cette variante est un algorithme d'estimation de distribution basé sur la recherche harmonique. Contrairement à d'autres algorithmes de recherche et d'optimisation stochastiques basés sur le concept de population. L'algorithme cHSA utilise une fonction de distribution pour représenter les harmonies connues (la population). Cet algorithme peut être considéré comme une combinaison entre l'algorithme évolutionnaire compact et la recherche harmonique.

Pour résumer le cHSA, nous le présentons à travers 4 étapes :

1. Improvisation : cette étape permet à l'algorithme d'improviser de nouvelles solutions. Comme la population (les harmonies stockées dans la mémoire harmonique) est représentée par une fonction de probabilité (uniforme), l'improvisation suit la formule mathématique suivante :

$$x(i) = \mu(i)+\sigma(i)*rand(-1,1) \qquad (III.11).$$

Avec x(i) est i ^{ième} paramètre, μ(i) est sa moyenne et σ(i) sa déviation standard.

L'algorithme génère deux individus (mélodies) durant chaque cycle. Cette procédure s'effectue sur trois étapes :

a. L'algorithme génère un nombre aléatoire. Si ce dernier est inférieur à HMCR (harmony consideration rate), l'algorithme improvise de VP (virtual population qui n'est que la représentation de la mémoire harmonique par une fonction de probabilité) suivant l'équation **III.11**.

b. L'algorithme génère un autre nombre aléatoire, si ce dernier est inférieur à PAR (pitch ajustement rate), le paramètre généré de VP est ajusté suivant l'équation suivante :

$$x(i)= x(i)+BW*rand(-1,1) \qquad (III.12)$$

avec BW est la bande d'ajustement. Nous notons que ce paramètre est adaptatif. Sa loi d'adaptation est donnée par (III.113).

$$BW=BW*BWDump \qquad (III.13)$$

Nous conseillons des valeurs entre 0.95 et 0.9999 pour Le paramètre BWDump

c. Dans le cas le premier nombre généré en (a) n'est pas inférieur à HMCR l'algorithme s'inspire aléatoirement. C'est-à-dire : il génère un nombre aléatoire dans l'espace de recherche.

Algorithm 2: Compact Harmony Search Algorithm

```
1.  N : Virtual population size
2.  λ : Initial standard deviation
3.  μ : mean of the probabilistic function
4.  m : problem dimension
5.  HMCR :Initialize harmony consideration rate
6.  BW_Dump Initialize the bandwidth dumping value (amortissement du pas d'ajustement)
7.  PAR Initialize the Pitch Adjustment Rate
8.  R : Initialize mutation rayon
9.  R_Dump : Initialize Mutation Rayon dumping value
10. t=1  /* iteration index */
11. for j=1:m do /* Initialization of PV */
12.       μ_j^t = 0;   σ_j^t = λ; /* means and standard deviation */
13. end
14.       Elite=generate( PV) /* Assume Elite */
15. while Termination condition /* Main loop */
16.     for i=1 : m do /* Improvisation from PV */
17.         /* improvising of G1and G2 */
18.       if rand < HMCR
19.           G1(i)=generate G1(i) from VP
20.          G2(i)=generate G1(i) from VP
21.           if rand < PAR
22.              G1(i)=G1(i)+rand(-1,1)*BW
23.              G2(i)=G2(i)+rand(-1,1)*BW
24.           endif
25.       else /* random generation in the search space */
26.           G1(i)= rand
27.           G2(i)= rand
28.     endif
29. endfor
30.  [ winner, loser ] = compete(G1, G2)
31. for i=1 : m do /* updating PV */
32.      μ_i^{t+1}= μ_i^t +1/N × (w_i^t –l_i^t );
33.     /* l_i and w_i are the genes of the loser and winner*/
34.   [σ_i^{t+1}]^2 = [σ_i^t]^2 +[μ_i^t]^2 -[μ_i^{t+1}]^2+1/N× ( [w_i^t]^2–[ l_i^t]^2)
35. endfor
36.    /* Update elite : compare elite to the winner between G1 and G2 */
37.  [ winner, loser ] = compete( winner, elite);
38.   elite= winner
39.    BW=BW*BW_Dump      /* BW_Dump <1 */
40. while j < =10      /* (1+1)-EA Procedure */
41.      NewElite=elite+R*randn
42.      [ winner, loser ] = compete(NewElite, elite)
43.      elite = winner ; /* Update elite */
44.      for i=1 : m do /* updating PV When NewElite is better than Elite*/
45.          μ_i^{t+1}= μ_i^t +1/N × (w_i^t –l_i^t );
46.          /* l_i and w_i are the genes of the loser and winner*/
47.          [σ_i^{t+1}]^2 = [σ_i^t]^2 +[μ_i^t]^2 -[μ_i^{t+1}]^2+1/N× ( [w_i^t]^2–[ l_i^t]^2);
48.      endfor
49.      R = R · R_Dump   /* R_Dump <1 */
50. j=j+1
51. end while
52. until termination condition
53. return elite and its cost
```

Figure. III. 4. Compact Harmony Search Algorithm.

2. Compétition : l'algorithme utilise une sélection par tournoi. C'est-à-dire : il compare deux individus générés comme décrit dans l'étape d'improvisations. Le meilleur d'entre ces deux est noté winner (gagnant) et le mauvais est noté loser (perdant).

3. Ajustement de VP : l'ajustement de VP s'effectue en ajustant les paramètres qui décrivent la fonction de probabilité utilisée pour estimer les harmonies connues. La formule d'ajustement est la même proposée par Minimo et al. [87] pour leur algorithme nommé cDE (compact differential evolution). Voir **Eq. III.**9 Et **Eq. III. 10.**
4. Procédure (1+1)-EA [90] : durant cette étape un raffinement est appliqué à l'élite (la meilleure solution déjà trouvée). Nous conseillons, pour une meilleure convergence de l'algorithme d'effectuer 10 cycles de raffinement par (1+1)-EA. La formule mathématique pour la création de nouvelles solutions pour cet algorithme est donnée en (III.14)

$$Newelite(i)= elite(i)+R*rand(-1,1) \quad \text{(III.14)}$$

Avec R le rayon de mutation. Ce paramètre est aussi adaptatif, sa loi d'adaptation est la suivante :

$$R=R*R_{Dump} \quad \text{(III.15)}$$

Nous conseillons des valeurs entre 0.95 et 0.999 pour R_{Dump}

5. À chaque fois qu'une solution meilleure est trouvée, l'algorithme ajuste la VP comme il est décrit dans l'étape 3.

Comme tout autre algorithme évolutionnaire, le cHSA actualise ses paramètres adaptatifs. Ceci inclut BW et $R_{Dump.}$ Comme est connu dans la littérature, l'utilisation des représentations compactes permet de réduire les capacités de calcul requises pour trouver les meilleures solutions. Le lecteur peut consulter les travaux Harik et al. [89] Minimo et al. [87] à ce sujet.

À titre d'exemple, avec le standard HSA, si nous avons un problème à 50 Dimensions avec 50 harmonies, il nous faut 2500 variables à enregistrer dans la mémoire de la CPU. Par contre pour le cHSA, il ne mémorise que cent paramètres qui sont les moyennes et les déviations standards de la fonction de probabilité utilisée.

III.3.3.Algorithmes compacts inspirés des lucioles (cFAs)

La dénomination FA vient de l'anglais Firefly Algorithm. Cet algorithme s'inspire des comportements des lucioles. Il a été introduit par Xin-She Yang à l'Université de Cambridge en se basant sur la modélisation des caractéristiques de luminosité et d'attractivité des lucioles [91-92]. Comme d'autres algorithmes basés sur la population, il utilise les positions des lucioles pour représenter les solutions trouvées. Chaque luciole est capable de rayonner des signaux pour attirer les autres vers sa propre position. Les

lucioles, représentées à la **Fig. III. 6**, sont supposées bisexuelles, et l'attractivité n'est pas pour la sexualité. Si la luciole x_i est meilleure que x_j, x_j sera déplacée vers x_i. Par conséquent, les meilleures lucioles attireront les autres vers elles. Pour résumer les étapes de l'algorithme, avec une simplicité suffisante, nous pouvons le diviser en trois étapes:

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

$$I\,(r\,) \;=\; I_0 e^{-\gamma\, r^2} \qquad \text{(III. 16)}$$

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

- **Attractivité**: exprime l'attraction entre deux lucioles. Elle peut être exprimée par:

$$ß(r\,) \;=\; ß_0 e^{-\gamma\, r^2} \qquad \text{(III. 17)}$$

Où $ß_0$ est l'attractivité des lucioles 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:

$$x_j^{t+1} = x_j^t + ß_0 e^{-\gamma\, r^2}\left(x_i^t - x_j^t\right) + \alpha\; randn(-0.5, 0.5) \qquad \text{(III. 18)}$$

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 pseudocode de FA est représenté sur la **Fig. III. 7**.

Fig. III. 5 un essaim de lucioles durant la nuit.

Algorithm 4: Firefly Algorithm

1. Problem Definition : Objective function *f(x)*
2. Generate an initial population of *n* fireflies
3. Define light absorption coefficient γ, I_0 and β_0
4. **while** termination condition == 0 **do**
5. **for** i = 1 : *N* **do** **/*** all fireflies */
6. **for** j = 1 : N (all n fireflies) (inner loop) **do**
7. **if** ($\boldsymbol{fitness}(\boldsymbol{x_i})$> $\boldsymbol{fitness}(\boldsymbol{x_j})$) **then**
   /* ">" means better */
8. Move firefly x_j towards firefly x_i according to **Eq. III. 18**
9. **end if**
10. Vary attractiveness with distance r using **Eq. III. 17.**
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

Fig. III. 6 Pseudocode de FA.

III.3.3.1.permanent elitism-based compact Firefly Algorithm

La stratégie d'élitisme permanent a été proposée dans [74] pour bcGA. Cette stratégie a ensuite été utilisée dans rcGA [70] et dans cDE [71]. L'élitisme permanent crée une pression de sélection suffisante pour améliorer la convergence [74]. Les bons résultats obtenus par cette stratégie [70-71] ont motivé Lyes TIGHZERT et al. pour améliorer cFA. Nous décrivons ici l'algorithme de luciole compacte basé sur l'élitisme permanent. Il a été noté pe-cFA : *permanent elitism-based compact Firefly Algorithm*. [112**]**

Algorithm 5 : permanent elitism-based compact Firefly Algorithm

1. N : Virtual population size
2. λ : Initial standard deviation
3. m : problem dimension
4. t=1 /* iteration index */
5. **for** j=1:m **do** /* Initialization of PV */
6. $\mu_j^t = 0$; $\sigma_j^t = \lambda$;
7. **end**
8. elite=generate(PV) */* Assume an elite */*
9. **while** *Termination condition* **do**
10. */* Main loop, the termination condition can be fixed by the user */*
11. G_1=generate ($PV^{(t)}$)
12. [winner, loser] = compete(G_1, elite)
13. G1=loser; elite=winner
14. Move G1 toward the elite using **Eq. III. 17**
15. [winner, loser] = compete(G1, elite)
16. elite = winner ; /* Update elite */
17. **for** i=1 : m **do** /* updating PV */
18. $\mu_i^{t+1} = \mu_i^t + 1/N \times (w_i^t - l_i^t)$;
19. $[\sigma_i^{t+1}]^2 = [\sigma_i^t]^2 + [u_i^t]^2 - [u_i^{t+1}]^2 + \frac{1}{N} \times ([w_i^t]^2 - [l_i^t]^2)$;
20. /* where l_i and w_i are the genes of the loser and winner*/
21. **end for**
22. **end while**
23. **return** the best individual

Fig. III. 7 Pseudocode de pe-cFA.

À chaque génération, une luciole G1 est générée à partir de PV. Nous comparons (compétition) G1 à l'élite (la meilleure position connue). Si G1 est meilleur que le candidat *elite*, nous mettons à jour ce dernier. Le perdant de la compétition est déplacé selon les **Eq. III. 18** vers le gagnant, c'est-à-dire le plus apte. Nous évaluons la nouvelle position, et nous la comparons à *elite*. Ce dernier est mis à jour une fois que la luciole déplacée le surpasse. Selon les allèles du vainqueur et du perdant de la dernière compétition, les moyennes et l'écart-type de NPDF sont mis à jour avec le même mécanisme que celui proposé dans [70-72], voir équations **Eq. III.9** et **Eq. III.10**. La **Fig. III.7** donne le pseudocode de pe-cFA.

III.3.4.ne-cFA

Le pe-cFA présenté ci-dessus conserve la meilleure solution actuelle jusqu'à ce qu'une meilleure solution soit trouvée. Dans cette section, nous exposons une autre variante de cFA qui utilise une stratégie d'élitisme non permanente. L'algorithme est appelé algorithme de luciole compacte à stratégie d'élitisme non permanente. (ne-cFA : *non-permanent elitism-based Firefly Algorithm*). L'idée dans l'élitisme non permanent

[70-71, 74] consiste à relâcher la pression de sélection, c'est-à-dire l'élitisme, en limitant la longueur de l'héritage du chromosome de l'élite. Cela signifie que le candidat *elite* est mis à jour lorsque les autres lucioles ne le dépassent pas après une longueur d'héritage donnée notée η. Ce schéma diminue la possibilité d'une convergence prématurée. Le pseudocode de ne-cFA peut être simplement dérivé de pe-cFA en ajoutant, ou en restreignant, la longueur de l'héritage des chromosomes des candidats *elite*. Il a été démontré que la longueur de l'hérédité η est limitée par la taille de la population simulée [70-71, 74]. Le pseudocode complet est représenté sur la **Fig. III. 8**.

Algorithm 6 : Non-permanent elitism-based compact Firefly Algorithm

1. N : Virtual population size
2. λ : Initial standard deviation
3. η= the allowed length of inheritance
4. m : problem dimension
5. t=1 /* iteration index */
6. **for** j=1:m **do**/* Initialization of PV */
7. $\mu_j^t = 0$; $\sigma_j^t = \lambda$;
8. **end**
9. elite=generate(PV) */* Assume an elite */*
10. θ=0; /* represent the actual length of inheritance*/
11. **while** Termination condition **do**
/* *Main loop, the termination condition can be fixed by the user* */
12. G_1=generate ($PV^{(t)}$)
13. [winner, loser] = compete(G_1, elite)
14. G1=loser; elite=winner
15. Move G1 toward the elite using **Eq. III.3**
16. [winner, loser] = compete(G1, elite)
17. **if** winner==G1 **OR** θ==η **then**
18. elite = G1;
19. θ=0;
20. **end if**
21. **for** i=1 : m **do** /* updating PV */
22. $\mu_i^{t+1} = \mu_i^t + 1/N \times (w_i^t - l_i^t)$;
23. $[\sigma_i^{t+1}]^2 = [\sigma_i^t]^2 + [u_i^t]^2 - [u_i^{t+1}]^2 + \frac{1}{N} \times ([w_i^t]^2 - [l_i^t]^2)$;
24. /* where l_i and w_i are the genes of the loser and winner*/
25. **end for**
26. θ=θ+1;
27. **end while**
28. **return** the best individual

Fig.III.8. Non-permanent elitism-based compact Firefly Algorithm

III.4.Conclusion

Dans ce chapitre nous avons présenté une méthode très connue dans la littérature spécialisée pour réduire les capacités de calculs des métaheuristiques basées sur la population. Le principe de la méthode est basé sur la représentation distributive, compacte, de la population. Nous avons présenté plusieurs algorithmes compacts

comme le bcGA, cHSA et cFA. Dans le chapitre suivant, nous allons proposer un nouvel algorithme compact basé sur la gravitation.

CHAPITRE IV
Compact Gravitational Search Algorithm

IV.1.ntroduction

Dans ce chapitre, on va présenter la version compacte de l'algorithme de recherche gravitationnelle. Cet algorithme nouveau est appelé cGSA pour : compact Gravitational Serach Algorithm (cGSA). Cet algorithme peut être vu comme la version compacte de GSA. On peut aussi le considéré comme une hybridation entre GSA et cGA (*compact genetic algorithm*). On va présenter dans ce chapitre notre algorithme et une validation sur des problèmes mathématiques. Nous donnons aussi une application en robotique mobile de notre algorithme.

IV.2.Problématique et objectif de notre proposition

Les algorithmes basés sur des populations demandent une grande mémoire et des capacités de calcul assez élevées pour résoudre un problème donné. Cela est le cas du GSA. C'est pour cela que nous adressons dans ce mémoire la problématique de la réduction des capacités de calcul requises par GSA. L'objectif est de réduire les capacités de calcul et de mémoire requises par GSA

IV.2.1.Solution Proposée et Motivation

Les algorithmes inspirés des essaims et des populations sont l'une des approches les plus efficaces pour élaborer des algorithmes de recherche. Les informations relatives à l'essaim ou à la population doivent être stockées et utilisées pour faire progresser l'algorithme. Généralement, afin de résoudre un problème de grande dimension avec des algorithmes stochastiques basés sur une population, un grand nombre d'individus est requis. Cependant, la capacité de calcul requise augmente en fonction de la taille de la population. L'optimisation compacte offre une alternative et assure une réduction des capacités de calcul requises. Les bons résultats obtenus par la méthode basée sur la représentation compacte nous ont motivés à adapter cette proposition pour réduire les capacités de calcul et de mémoire requises par GSA

IV.2.2.Proposition cGSA, The Compact Gravitational Search Algorithm

Un certain nombre de variantes ont été rapportées dans la littérature depuis la publication de la GSA originale en 2009. On peut citer : Rashedi et al. [92] ont proposé le Binary GSA (BGSA) par l'introduction d'une fonction de probabilité pour la valeur absolue de la vitesse, l'Adaptive GSA par Precup et al. Li et Zhou [94] ont proposé le Improved Gravitational Search Algorithm (IGSA) en introduisant une stratégie dans

l'espace de recherche, obéissant à la loi de la gravité et en utilisant la stratégie PSO comme hybridation. [95,96], on trouve aussi le Chaotic GSA [97].

Depuis sa création, GSA et ses variantes ont trouvé plusieurs applications dans le domaine de la production et de la distribution d'énergie telle que le problème de répartition de la puissance réactive [62,80] pour minimiser les pertes de puissance réelles et l'amélioration de la stabilité de la tension, un système de gestion de l'énergie pour optimiser les performances de production et du système de transmission.

Notre objectif et de proposer une nouvelle variante de l'algorithme de recherche gravitationnelle (Gravitational Search Algorithm). L'algorithme proposé est appelé ***compact gravitational search algorithm (cGSA)***. Cette nouvelle variante est un algorithme d'estimation de distribution basé sur la recherche gravitationnelle. Contrairement à d'autres algorithmes de recherche et d'optimisation stochastiques basés sur le concept de population. L'algorithme cGSA utilise une fonction de distribution pour représenter les agents connus (la population). Cet algorithme peut être considéré comme une combinaison entre l'algorithme évolutionnaire compact et la recherche gravitationnelle.

Pour résumer le cGSA, nous le présentons à travers 4 étapes :

1. Génération de la population virtuelle : cette étape permet à l'algorithme de générer une population représentée par une fonction de probabilité (uniforme ou normale),

2. L'algorithme génère deux individus (deux corps ou deux masses) durant chaque cycle, évalue la fitness des deux individus et mis à jour du Best.

$\sigma_i = 10$ et $\mu_i = 0$ pour i=1 : D où D est la dimension du problème à résoudre

3. Compétition : l'algorithme utilise une sélection par tournoi. C'est-à-dire : il compare deux individus générés. Le meilleur d'entre ces deux est noté winner (gagnant) et le mauvais est noté loser (perdant).

4. À chaque fois qu'une solution meilleure est trouvée, l'algorithme ajuste la VP comme décrit dans selon l'équation suivante :

$$\mu_i^{t+1} = \mu_i^{t} + 1/N \times (w_i^{t} - l_i^{t}) \quad \textbf{(IV.1)}$$

$$\sigma_i^{t+1} = [\sigma_i^{t}]^2 + [u_i^{t}]^2 - [u_i^{t+1}]^2 + 1/N \times ([w_i^{t}]^2 - [l_i^{t}]^2) \quad \textbf{(IV.2)}$$

Avec $\mu(i)$ est la moyenne et $\sigma(i)$ sa déviation standard, N : Nombre d'individus, w_i^t est le ^{ième} Winner et l_i^t est le i ^{ième} Loser.

1. Mouvement des masses : après la compétition l'algorithme déplace le loser (perdant) vers le winner et le winner vers le best, par la loi de la gravité, et puis évalue la fitness de ces deux derniers.
2. Adaptation de VP comme en étape 4 Adaptation du best si il est dépassé.
3. Comme tout autre algorithme évolutionnaire, le cGSA actualise ses paramètres adaptatifs. Ceci incluse le paramètre de gravité G, comme est connu dans la littérature, l'utilisation des représentations compactes permet de réduire les capacités de calcule requises pour trouver les meilleures solutions. Le lecteur peut consulter les travaux Harik et al. [89] Minimo et al. [87] à ce sujet.
4. Si le critère d'arrêt n'est pas satisfait, on revient à l'étape 2.

L'organigramme est présenté en Fig. IV. 1

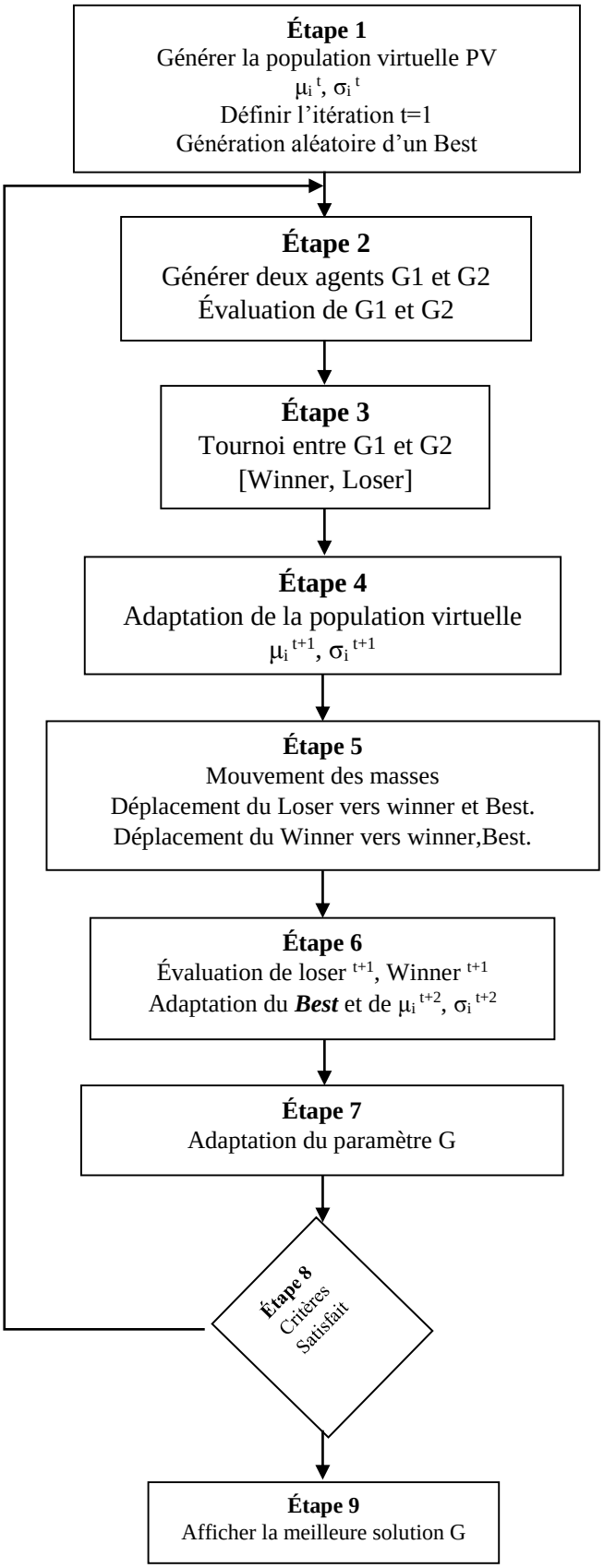

Fig.IV.1 Organigramme cGSA

Le pseudocode de cGSA est donné en Fig. IV.2

Algorithm : compact gravitational search Algorithm

1. N : Virtual population size
2. λ : Initial standard deviation
3. μ:mean of the probabilistic function
4. D: problem dimension
5. Get the initial values of gravitational constant G0
6. t=1 /* iteration index */
7. **for** j=1:D **do** /* Initialization of PV */
8. $\mu_j^t = 0$; $\sigma_j^t = \lambda$; /* means and standard deviation*/
9. **end for**
10. ***Best*** */*Assumption of a best */*
11. *Evaluate the fitness of the best*
12. **while** (Termination condition) /* Main loop */
13. [G_1 , G_2]=**generate** ($PV^{(t)}$)
14. Evaluate the fitness of the [G1,G2]
15. Update the best
16. [winner, loser] = *compete*(G_1, G2)
17. **Move** *the loser toward the winner*
18. **for** i=1 : m **do** /* updating PV */
19. $\mu_i^{t+1} = \mu_i^t + 1/N \times (w_i^t - l_i^t)$;
20. $[\sigma_i^{t+1}]^2 = [\sigma_i^t]^2 + [u_i^t]^2 - [u_i^{t+1}]^2 + 1/N \times ([w_i^t]^2 - [l_i^t]^2)$;
21. /* where l_i and w_i are the genes of the loser and winner*/
22. **end for**
23. Calculate the inertial mass Mi as shown in equation II.15, II.16.
24. Calculate the force acting on agent i from agent j as shown in equation 8
25. Evaluate the fitness of the force (winner)
26. [winner, loser] = *compete*(w, best)
27. *Move the winner toward the best*
28. **for** i=1 : m **do** /* updating PV */
29. $\mu_i^{t+2} = \mu_i^t + 1/N \times (w_i^t - l_i^t)$;
30. $[\sigma_i^{t+2}]^2 = [\sigma_i^t]^2 + [u_i^t]^2 - [u_i^{t+1}]^2 + 1/N \times ([w_i^t]^2 - [l_i^t]^2)$;
31. t=t+1
32. **End for**
33. **end while**
34. **Return** to the best individual.

Fig.IV.2: Principe de cGSA

IV.3.Tests de performance

IV.3.1. Fonctions de test et paramètres des algorithmes

Afin d'évaluer les performances de l'algorithme proposé, nous prenons de la littérature un ensemble de fonctions de référence célèbres. Cet ensemble comprend des fonctions unimodales et multimodales. Leurs formules mathématiques sont données dans le tableau IV.1, IV2 et IV3. Et les paramètres utilisés sont : **G= 100, D = 30 et N=30.** Avec G = Constant de la gravité, D = Dimension de problème et N = nombre d'individu.

IV.3.2.Fonction de test

TABLEAU IV.1 : Fonctions de test unimodales.

Test de Fonction	s
$F1=\sum_{i=1}^{n} x_i^2$	$[-100,100]^n$
$F2=\sum_{i=1}^{n}\lvert x_i\rvert+\prod_{i=1}^{n}\lvert x_j\rvert$	$[-10,10]^n$
$F3=\sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_j\right)^2$	$[-100,100]^n$
$F4=\max\{\lvert xi\rvert,1\le i\le n\}$	$[-100,100]^n$
$F5=\sum_{i=1}^{n-1}\left[100(x_{i+1}-1)^2+(x_i-1)^2\right]$	$[-30,30]^n$
$F6=\sum_{i=1}^{n}([x_i+0.5])^2$	$[-100,100]^n$
$F7=\sum_{i=1}^{n} ix_i^4+random[0,1]$	$[-1.28,1.28]^n$

TABLEAU IV.2 : Fonctions de test multimodales.

$F8=\sum_{i=1}^{n}-x_i\sin\sqrt{(x_i)}$	$[-500,500]^n$
$F9=\sum_{i=1}^{n}\left[x_i^2-10\cos(2\pi x_i)+10\right]$	$[-5.12,5.12]^n$
$F10=-20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right)-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right)+20+e$	$[-32,32]^n$
$F11=\frac{1}{400}\sum_{i=1}^{n}x_i^2-\prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	$[-600,600]^n$
$F12=\frac{\pi}{n}\{10\sin(\pi y_1)+\sum_{i=1}^{n-1}(y_i-1)^2[1+10\sin^2(\pi y_{i+1})]+(y_n-1)^2\}$ $+\sum_{i=1}^{n}u(x_i,10,100,4)$ $y_i=1+\frac{x_i+1}{4}$ $u(x_i,a,k,m)=\begin{cases}k(x_i-a)^m & x_i>a\\ 0 & -a<x_i<a\\ k(-x_i-a)^m & a_i<-a\end{cases}$	$[-50,50]^n$
$F13=0.1\{\sin^2(3\pi x_1)+\sum_{i=1}^{n}(x_i-1)^2[1+\sin^2(3\pi x_i+1)]+(x_n-1)^2[1+\sin^2(2\pi x_n)]\}$ $+\sum_{i=1}^{n}u(x_i,5,100,4)$	$[-50,50]^n$

TABLEAU IV.3 : Fonction de test multimodale avec dimension fixe

$F14 = \sum_{i=1}^{11}\left[a_i - \frac{x_1\left(b_i^2 + b_i x_2\right)}{b_i^2 + b_i x_3 + x_4}\right]^2$	$[-5,5]^4$
$F15 = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1 x_2 - 4x_2^2 + 4x_2^2$	$[-5,5]^2$
$F16 = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$	$[-5,10] \times [0,15]$
$F17 = \left[1 + (x_1 + x_2 + 1)^2\left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right]$ $\times\left[30 + (2x_1 - 3x_2)^2 \times \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right]$	$[-5,5]^2$

IV.4.Résultats et discussion

Les algorithmes sont testés pour les fonctions de référence ci-dessus. Chaque test a été répété 30 fois. Le nombre maximal d'itérations est fixé à 1000 comme indiqué dans [111]. Le tableau 4 donne les résultats finaux. Comme les algorithmes comparés sont stochastiques, on compare la moyenne trouvée dans 30 exécutions indépendantes. Ainsi, le tableau 4 donne les moyennes, l'écart-type et le temps d'excursions.

TABLEAU IV.2 les résultats de l'expérience, les moyens, et un écart-type du cGSA et GSA pour la fonction F1, F2, F3, F4 et F5 avec D = 30

Fonction		GSA	PSO	cGSA
F1	Moyenne	7.3×10^{-11}	1.8×10^{-3}	**$2{,}17 \times 10^{-49}$**
	L'écart-type	2.1×10^{-10}	5×10^{-2}	**$1{,}79 \times 10^{-49}$**
F2	Moyenne	4.03×10^{-5}	2.0	**$1{,}05 \times 10^{-26}$**
	L'écart-type	4.03×10^{-5}	2.0	**$8{,}85 \times 10^{-27}$**
F3	Moyenne	$0.16\times10^{+3}$	$4.1\times10^{+3}$	**$2{,}99 \times 10^{-14}$**
	L'écart-type	$0.15\times10^{+3}$	$2.9\times10^{+3}$	**$1{,}04 \times 10^{-13}$**
F4	Moyenne	3.7×10^{-6}	8.1	**$3{,}34 \times 10^{-12}$**
	L'écart-type	8.5×10^{-6}	23.6	**$6{,}41 \times 10^{-12}$**
F5	Moyenne	**25.16**	$3.6\times10^{+4}$	28,35
	L'écart-type	**25.18**	$3.7\times10^{+4}$	0,77
F6	Moyenne	**8.3×10^{-10}**	1.0×10^{-3}	3,01
	L'écart-type	**2.6×10^{-10}**	0.02	0,70
F7	Moyenne	0.018	0.04	**$7{,}14 \times 10^{-04}$**
	L'écart-type	0.533	1.04	**$4{,}98 \times 10^{-04}$**

		GSA	PSO	cGSA
F9	Moyenne	$-7.0195\times10^{+3}$	55.1	**51,62**
	L'écart-type	735.8297	72.8	**45,31**
F10	Moyenne	**36.0181**	9×10^{-3}	51,62
	L'écart-type	**17.5077**	0.02	$3{,}16\times10^{-12}$
F11	Moyenne	2.02×10^{-8}	0.01	$\mathbf{1{,}29\times10^{-12}}$
	L'écart-type	2.21×10^{-9}	0.055	**0,01**
F12	Moyenne	71.6393	0.29	**0,0064**
	L'écart-type	47.1895	$9.3\times10^{+3}$	**0,01**
F13	Moyenne	0.0346	$\mathbf{3.2\times10^{-32}}$	0,25
	L'écart-type	0.0916	$4.8\times10^{+5}$	0,20

		GSA	PSO	cGSA
F15	Moyenne	01770	2.8×10^{-3}	1,74
	L'écart-type	0.9651	215.60	0,23
F16	Moyenne	0.0073	-1.0316	**0,0011**
	L'écart-type	0.0095	-1.0316	$\mathbf{3{,}56\times10^{-04}}$
F17	Moyenne	-1.0316	0.3979	-1,03
	L'écart-type	5.16×10^{-16}	2.4112	$1{,}13\times10^{-06}$

IV.4.1.Temps d'exécution GSA et cGSA

TABLEAU IV.4 Evaluations des résultats de l'expérience, les moyens de transformation de temps du cGSA et GSA pour la fonction F1, F2, ... F17 avec D = 30

cGSA	**Fonction**	*F1*	*F2*	*F3*	*F4*	*F5*
	Temps	***0.9807***	***1.1232***	***4.4967***	***1.2400***	***1.5733***
	Fonction	F6	F7	F9	F10	F11
	Temps	**1.5128**	**2.0268**	**2.2687**	**1.4366**	**2.6696**
	Fonction	F12	F13	F15	F16	F17
	Temps	**2.7446**	**3.3969**	**3.3284**	**1.5705**	**1.2791**
GSA	**Fonction**	*F1*	*F2*	*F3*	*F4*	*F5*
	Temps	*3.7910*	*3.9246*	*5.8464*	*4.1283*	*4.1751*
	Fonction	F6	F7	F9	F10	F11
	Temps	4.0033	4.5517	4.2411	4.0545	4.1997
	Fonction	F12	F13	F15	F16	F17
	Temps	4.4603	5.6348	4.9918	4.2382	4.8005

1) *Qualité convergence et précision*: pour cette comparaison, nous utilisons la moyenne trouvée. Comme nous pouvons le voir, dans le Tableau IV.4, le cGSA dépasse le GSA dans plusieurs cas. Les résultats obtenus sont très compétitifs comparé au PSO aussi.

2) *Stabilité des algorithmes*: afin d'évaluer la stabilité de l'algorithme, nous calculons l'écart-type. Cette dernière donne la répartition des résultats trouvés autour des valeurs moyennes. Dans le Tableau IV.4, on peut voir que cGSA est très stable.

3) *Temps de calcul*: Le temps que prend un algorithme évolutif pour renvoyer la solution optimale est l'un des facteurs les plus importants. Dans le **Tableau IV.4**, nous donnons le temps en moyenne pour chaque fonction. On peut voir que cGSA est plus rapide que GSA.

IV.4.2. Comparaison graphique des performances de GSA et de cGSA :

Nous présentons ici les différences d'allure de la progression de notre algorithme comparé à la version originale.

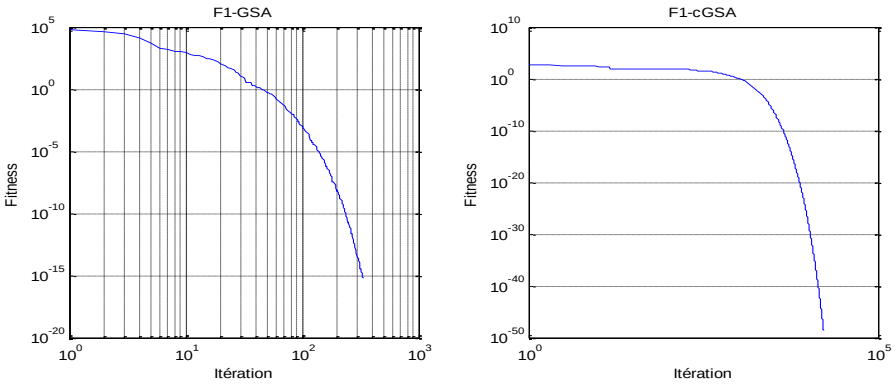

Fig.IV.3. Comparaison des performances de GSA et cGSA pour la minimisation de F1.

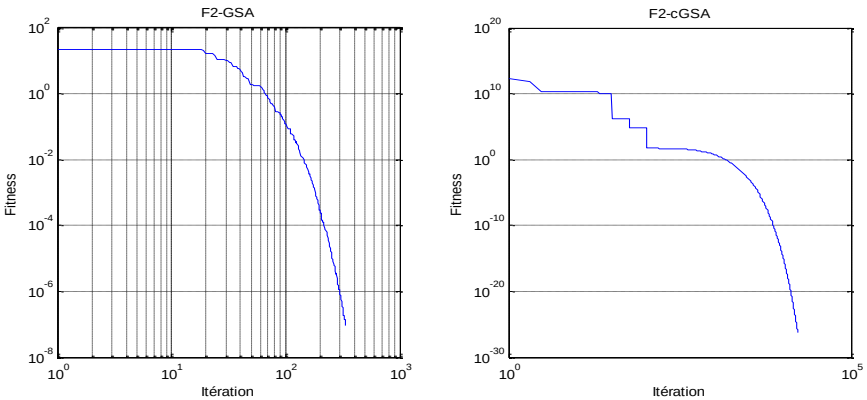

Fig.IV.4. Comparaison des performances de GSA et cGSA pour la minimisation de F2.

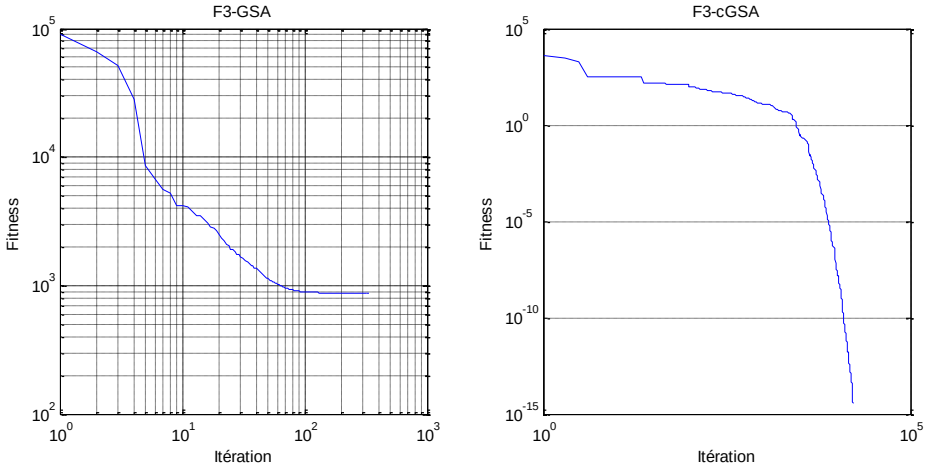

Fig.IV.5. Comparaison des performances de GSA et cGSA pour la minimisation de F3.

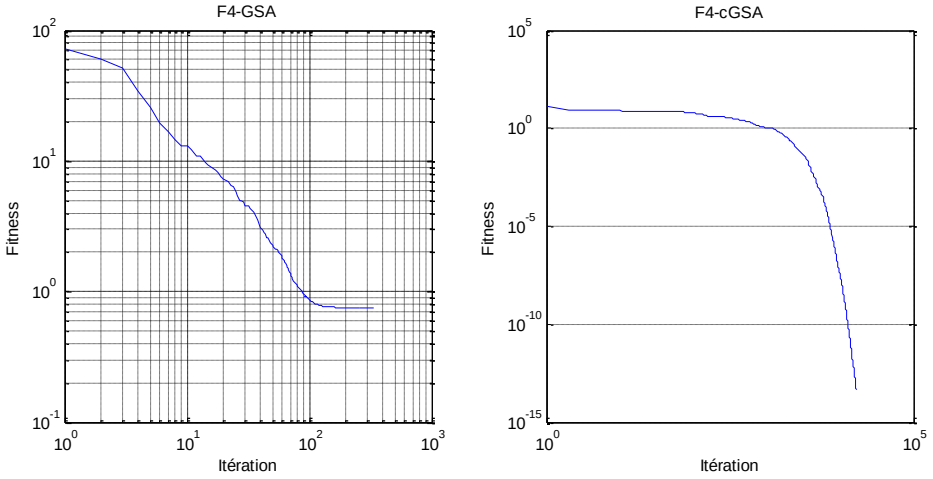

Fig.IV.6. Comparaison des performances de GSA et cGSA pour la minimisation de F4.

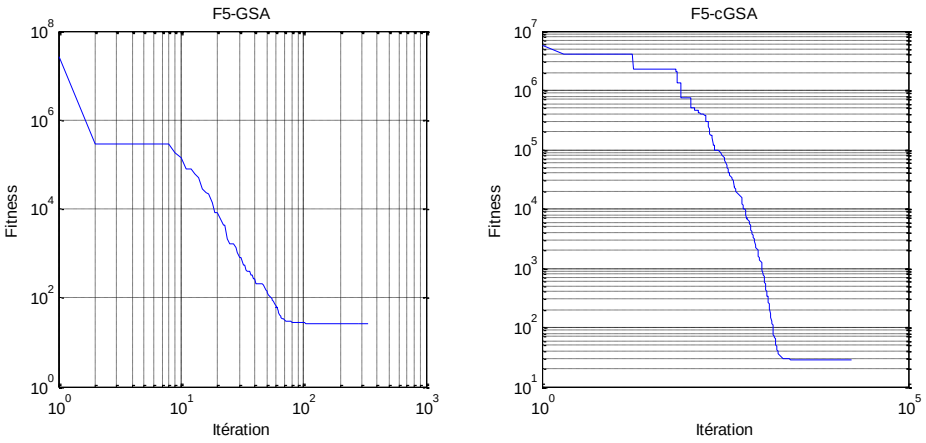

Fig.IV.7. Comparaison des performances de GSA et cGSA pour la minimisation de F5.

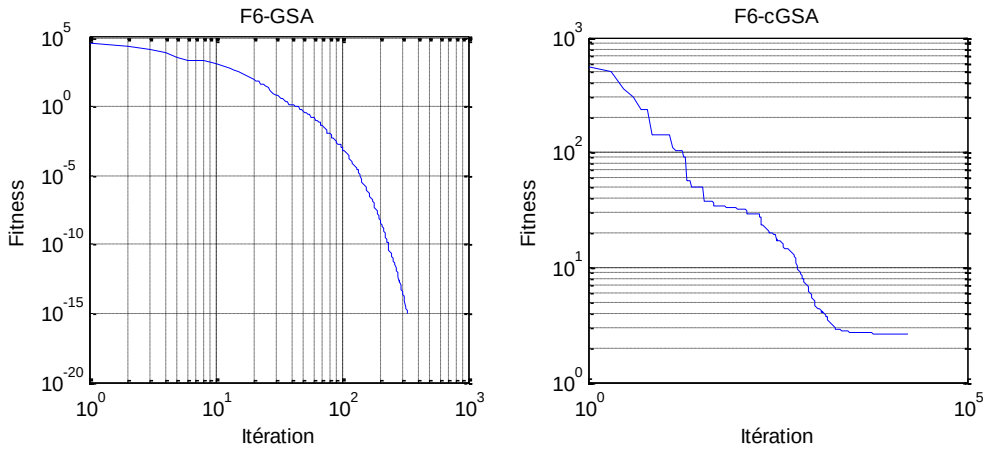

Fig.IV.8. Comparaison des performances de GSA et cGSA pour la minimisation de F6.

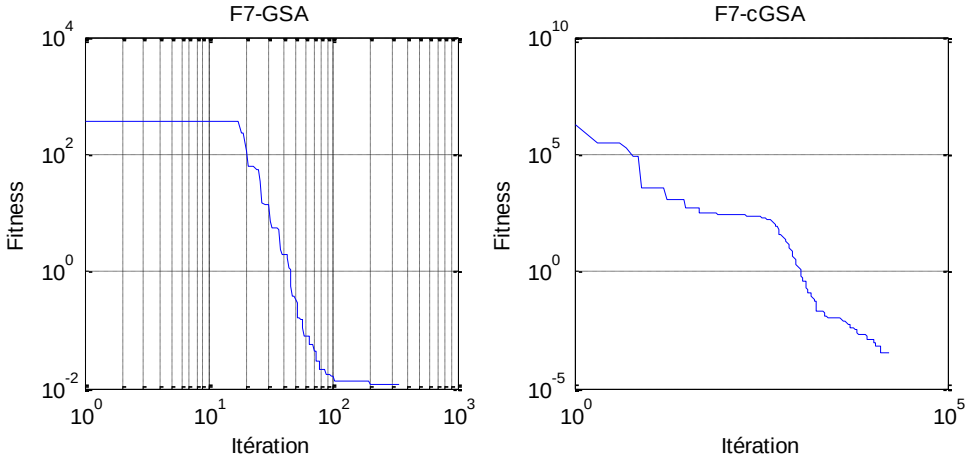

Fig.IV.9. Comparaison des performances de GSA et cGSA pour la minimisation de F7.

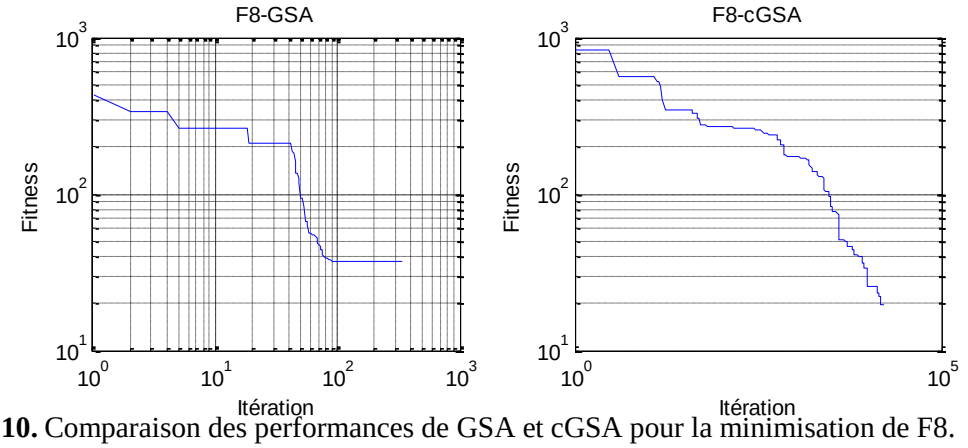

Fig.IV.10. Comparaison des performances de GSA et cGSA pour la minimisation de F8.

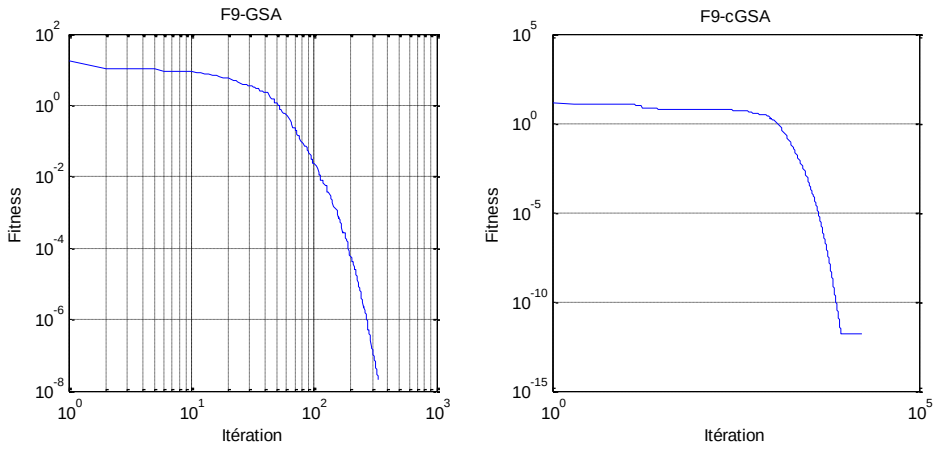

Fig.IV.11. Comparaison des performances de GSA et cGSA pour la minimisation de F9.

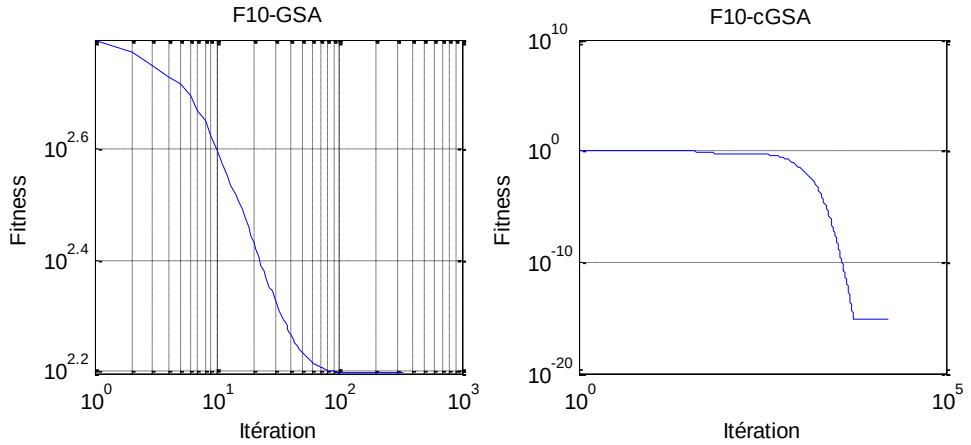

Fig.IV.12. Comparaison des performances de GSA et cGSA pour la minimisation de F10.

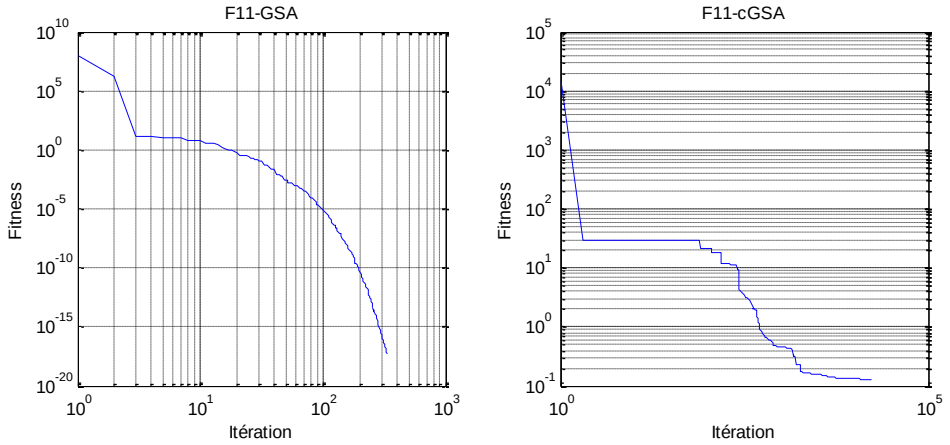

Fig.IV.13. Comparaison des performances de GSA et cGSA pour la minimisation de F11.

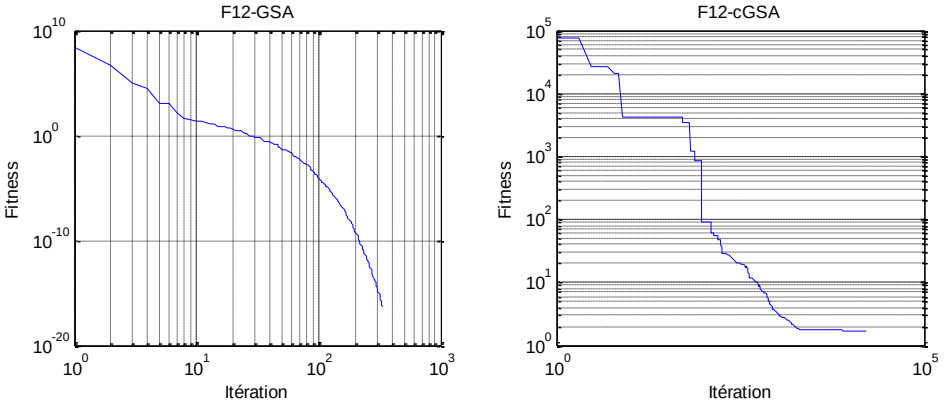

Fig.IV.14. Comparaison des performances de GSA et cGSA pour la minimisation de F12.

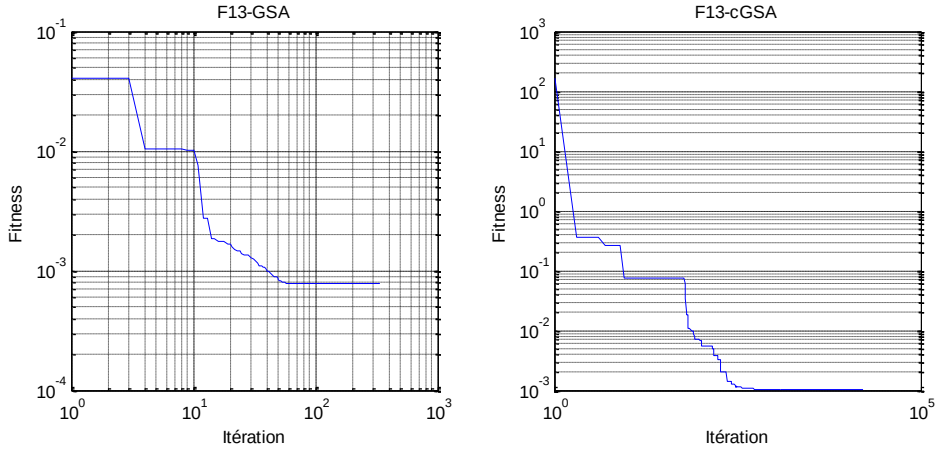

Fig.IV.15. Comparaison des performances de GSA et cGSA pour la minimisation de F13.

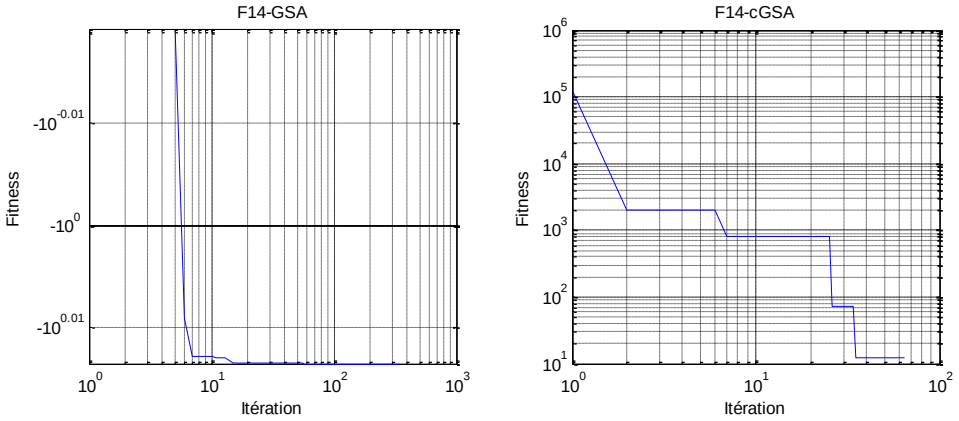

Fig.IV.16. Comparaison des performances de GSA et cGSA pour la minimisation de F14.

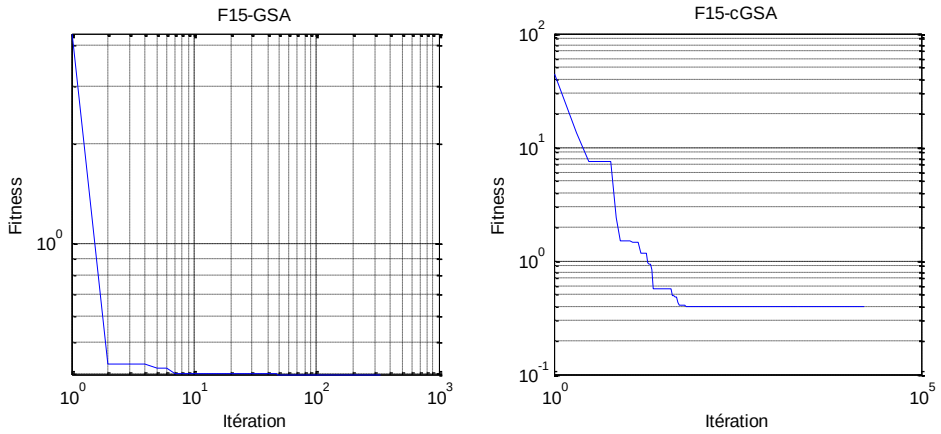

Fig.IV.17. Comparaison des performances de GSA et cGSA pour la minimisation de F15.

IV.5. Application à la planification de trajectoire d'un robot mobile

Dans cette section, nous traitons le problème de planification de trajectoire d'un robot mobile dans un environnement contenant des obstacles statiques. L'objectif est de retrouver le chemin le plus court à partir de la position initiale à la position finale tout en évitant les obstacles présents dans l'environnement.

Nous procédons à la modélisation sur Matlab d'un robot de dimension rectangulaire. On part du principe que le robot à la position exact de tous les obstacles. Nous avons développé une plateforme de simulation qui nous permettra de visualiser la progression de la recherche par cGSA du chemin optimal. Dans cette application on avait recours à un langage orienté objet. Le but principal de cette application et de pouvoir implémenter notre algorithme sur un problème d'évitement d'obstacle et de planification de trajectoire optimale. Nous allons attribuer au robot un point de départ ainsi qu'un point d'arrivée, le robot calcule le chemin le plus court possible en évitant tous les obstacles.

Pour la réalisation de cette application, nous allons procéder en deux grandes étapes :

IV.5.1.Modélisation d'un modèle du robot et de son environnement.

Pour pouvoir considérer le robot comme un point, nous avons augmenté la taille des obstacles de 40 CM. Nous avons supposé que le robot a une taille de 20 cm× 40 cm. 10. Les obstacles sont supposés circulaires et de tailles différentes. Ils sont donnés en couleur verte et l'augmentation hypothétique est donnée en couleur rouge. Le chemin est calculé par des splines cubiques avec 5 points intermédiaires.

IV.5.2.Implémentation de l'algorithme cGSA.

Nous implémentons le programme cGSA pour permettre au robot de retrouver le chemin optimal entre la position initiale et finale. L'algorithme doit retrouver les cordonnées des 5 points intermédiaires.

Le robot calcule le chemin le plus court possible entre le point de départ qui est (X= -10, Y= -15) et le point d'arriver qui est de (X= 10, Y= 12), en tenant compte des obstacles.

Le résultat obtenu est présenté en Fig. IV. 17. Nous concluons que le cGSA donne d'excellents résultats en utilisant un strict minimum de capacité de calcule. Ceci est la contribution de ce mémoire. L'objectif principal fixé est atteint avec succès.

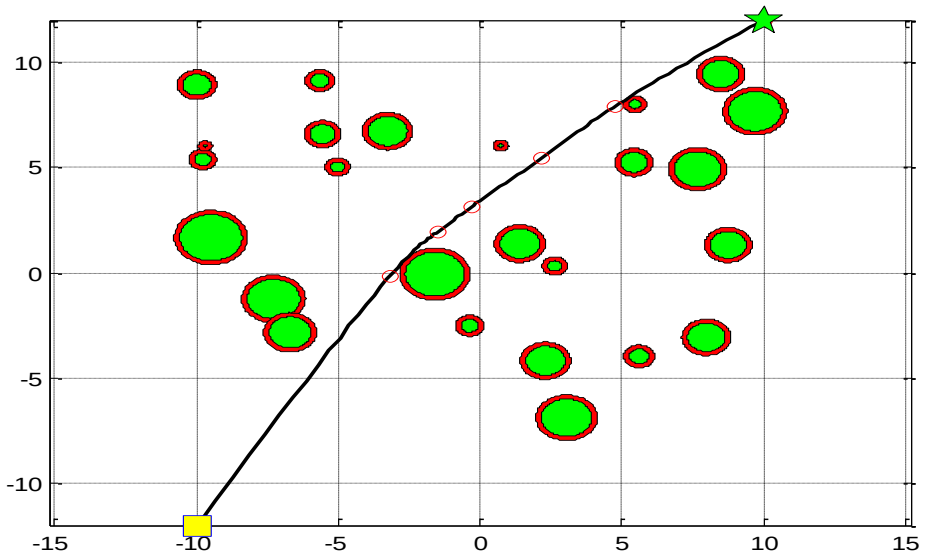

Fig.IV.18. Résultat obtenu pour la planification de trajectoire du robot mobile.

IV.6.Conclusion

Dans ce chapitre, un nouvel algorithme compact (cGSA) est introduit. Afin d'évaluer notre algorithme, nous l'avons testé sur un ensemble de fonctions de référence connues pour valider les performances des algorithmes. Les résultats obtenus par cGSA, dans la plupart des cas, fournissent des résultats meilleurs que PSO et la version standard de GSA. Nous avons réussi à réduire les capacités de calcul requises par GSA par l'utilisation de la représentation compacte sans influencer ses performances. Notre approche a permis d'améliorer les performances de GSA. On a constaté aussi que le

temps d'exécution du cGSA est largement inférieur à celui du GSA. De plus, nous avons présenté une application de l'algorithme cGSA sur un robot mobile en vue d'éviter des obstacles et de retrouver un chemin optimal. Ainsi, l'algorithme proposé est vivement recommandé pour des applications réelles.

Conclusions et Perspectives

Conclusions

Ce travail est un mémoire de Master en Automatique et système effectué à l'université de Bejaia. Il est consacré à la proposition d'un nouvel algorithme intelligent basé sur la théorie de la gravitation et la représentation compacte. Il nous a permis, entre autres, de nous confronter à plusieurs problèmes interdisciplinaires toujours en relation avec les métaheuristiques et l'automatique d'une manière générale.

Au cours des dernières années, diverses méthodes d'optimisation heuristique ont été développées. Certains de ces algorithmes sont inspirés par les comportements des essaims et d'autres sur la génétique**.** Dans ce mémoire, l'algorithme étudié est appelé algorithme de recherche gravitationnelle (GSA). GSA est construit sur la base de la loi de Gravité et de la notion d'interactions entre des masses ponctuelles. L'algorithme GSA utilise la théorie de la physique newtonienne et ses agents de recherche sont une collection de masses. La force gravitationnelle est un moyen de transférer des informations entre différentes masses.

Comme tout autre algorithme basé sur une population, le GSA présente l'inconvénient de ses capacités de calcul élevées. Dans ce mémoire, néanmoins, un nouvel algorithme dédié à l'optimisation et la recherche est proposé. Nous l'avons nommé ***compact Gravitational Search Algorithm (cGSA)***. L'idée principale consiste à compacter l'information relative à la population de masses et de la représenter statistiquement afin de réduire les capacités de calcul et de mémoire requise par GSA. Pour ce faire nous avons conduit notre travail en quatre étapes.

La première consiste à étudier le vaste domaine des métaheuristiques et de fournir un bref état de l'art des techniques utilisées. Nous avons présenté étude dans le chapitre 1 et cela nous permis d'approfondir le domaine des métaheuristiques. Dans la deuxième étape, nous avons étudié l'algorithme GSA qui est basé sur la théorie de la gravitation. Son principe de fonctionnement et les détails qui lui sont liés sont présentés dans le chapitre 2. Dans la troisième étape, nous avons étudié la réduction des capacités de calculs par la représentation compacte. Les algorithmes étudiés, comme le bCGA, pe-cFA et ne-cFA, sont présentés au chapitre 3. La quatrième étape consiste à appliquer la représentation compacte sur GSA. L'algorithme obtenu a été baptisé cGSA : *compact gravitational search algorithm*. L'algorithme et la validation par des fonctions de tests

sont présentés au chapitre 4. Nous avons fourni aussi une application de notre algorithme pour la planification de trajectoire d'un robot mobile dans un environnement contenant des obstacles.

Les grandes contributions de ce travail sont trois. Nous avons proposé un nouvel algorithme intelligent appelé cGSA. Nous avons réussi à réduire les capacités de calcul de ce dernier. L'algorithme obtenu, le cGSA, donne de meilleurs résultats comparés à la version standard. Nous avons donc réussi à l'améliorer.

Perspectives.

Les perspectives à tirer de notre travail sont diverses.

- Nous proposons dans le futur d'appliquer notre algorithme sur des problèmes d'optimisation réelle en industrie.
- L'utilisation des stratégies d'élitisme peut améliorer considérablement notre approche.
- L'hybridation de notre approche avec d'autres stratégies bio-inspiré permettra aussi de tester notre algorithme.

Nous souhaitons que notre travail soit publié et repris par d'autres étudiants et chercheurs.

Référence bibliographique

[1]. Clerc, M. et Siarry, P. (2004), Une nouvelle métaheuristique pour l'optimisation difficile: la méthode des essaims particulaires, J3eA - Vol. 3 - 7 (2004).

[2]. V. Angel, *La rugosité des paysages : une théorie pour la difficulté des problèmes d'optimisation combinatoire relativement aux métaheuristiques*, thèse de doctorat de l'université de Paris-Sud, Orsay, 1998.

[3]. J. Dréo, J.-P. Aumasson, W. Tfaili, P. Siarry, *Adaptive Learning Search, a new tool to help comprehending metaheuristics*, to appear in the International Journal on Artificial Intelligence Tools.

[4]. El-Ghazali Talbi, *A taxonomy of hybrid metaheuristics*, Journal of Heuristics, volume 8, n° 2, pages 541-564, septembre 2002.

[5]. W. Dullaert, M. Sevaux, K. Sörensen et J. Springael, *Applications of metaheuristics*, numéro spécial du *European Journal of Operational Research*, n° 179, 2007.

[6]. Nicolas Durand, David Gianazza, Jean-Baptiste Gotteland et Jean-Marc Alliot, *Metaheuristics for Air Traffic Management*, John Wiley & Sons, coll. « Computer Engineering Series / Métaheuristics set » (n° 2), 2016 (ISBN 9781848218109)

[7]. Robbins, H. and Monro, S., *A Stochastic Approximation Method*, Annals of Mathematical Statistics, vol. 22, pp. 400-407, 1951

[8]. Barricelli, Nils Aall, *Esempi numerici di processi di evoluzione*, Methodos, pp. 45-68, 1954

[9]. Rechenberg, I., *Cybernetic Solution Path of an Experimental Problem*, Royal Aircraft Establishment Library Translation, 1965

[10]. Fogel, L., Owens, A.J., Walsh, M.J., *Artificial Intelligence through Simulated Evolution*, Wiley, 1966

[11]. W.K. Hastings. *Monte Carlo Sampling Methods Using Markov Chains and Their Applications*, Biometrika, volume 57, n° 1, pages 97-109, 1970

[12]. Holland, John H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975

[13]. Smith, S.F., *A Learning System Based on Genetic Adaptive Algorithms*, PhD dissertation (University of Pittsburgh), 1980

[14]. S. Kirkpatrick, C. D. Gelatt et M. P. Vecchi, *Optimization by Simulated Annealing*, Science, volume 220, n° 4598, pages 671-680, 1983

[15]. V. Cerny, *A thermodynamical approach to the travelling salesman problem : an efficient simulation algorithm* Journal of Optimization Theory and Applications, volume45, pages 41-51, 1985

[16]. Fred GLover, Future Paths for Integer Programming and Links to Artificial Intelligence, Comput. & Ops. Res.Vol. 13, No.5, pp. 533-549, 1986

[17]. J.D. Farmer, N. Packard and A. Perelson, *The immune system, adaptation and machine learning*, Physica D, vol. 22, pp. 187--204, 1986

[18]. F. Moyson, B. Manderick, *The collective behaviour of Ants : an Example of Self-Organization in Massive Parallelism*, Actes de AAAI Spring Symposium on Parallel Models of Intelligence, Stanford, Californie, 1988

[19]. Koza, John R. *Non-Linear Genetic Algorithms for Solving Problems*. United States Patent 4,935,877. Filed May 20, 1988. Issued June 19, 1990

[20]. Goldberg, David E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Kluwer Academic Publishers, Boston, MA., 1989

[21]. P. Moscato, *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts : Towards Memetic Algorithms*, Caltech Concurrent Computation Program, C3P Report 826, 1989.

[22]. M. Dorigo, *Optimization, Learning and Natural Algorithms*, Thèse de doctorat, Politecnico di Milano, Italie, 1992.

[23]. Feo, T., Resende, M., *Greedy randomized adaptive search procedure*, Journal of Global Optimization, tome 42, page 32--37, 1992

[24]. Eberhart, R. C. et Kennedy, J., *A new optimizer using particle swarm theory*, Proceedings of the Sixth International Symposium on Micromachine and Human Science, Nagoya, Japan. pp. 39-43, 1995

[25]. Kennedy, J. et Eberhart, R. C., *Particle swarm optimization*, Proceedings of IEEE International Conference on Neural Networks, Piscataway, NJ. pp. 1942-1948, 1995

[26]. Mülhenbein, H., Paaß, G., *From recombination of genes to the estimation of distribution I. Binary parameters*, Lectures Notes in Computer Science 1411: Parallel Problem Solving from Nature, tome PPSN IV, pages 178--187, 1996

[27]. Rainer Storn, Kenneth Price, *Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces*, Journal of Global Optimization, volume 11, n^o 4, pages 341-359, 1997

[28]. Rubinstein, R.Y., *Optimization of Computer simulation Models with Rare Events*, European Journal of Operations Research, 99, 89-112, 1997

[29]. Stefan Boettcher, Allon G. Percus, "Extremal Optimization : Methods derived from Co-Evolution", Proceedings of the Genetic and Evolutionary Computation Conference (1999)

[30]. Takagi, H., *Active user intervention in an EC Search*, Proceesings of the JCIS 2000

[31]. A. Badr, A. Fahmy, A proof of convergence for ant algorithms, Information Sciences 160 (2004) 267–279.

[32]. Z. Baojiang, L. Shiyong, Ant colony optimization algorithm and its application to neuro-fuzzy controller design, Journal of Systems Engineering andElectronics 18 (2007) 603–610.

[33]. F.V.D. Bergh, A.P. Engelbrecht, A study of particle swarm optimization particle trajectories, Information Sciences 176 (2006) 937–971.

[34]. O. Cordon, S. Damas, J. Santamarı, A fast and accurate approach for 3D image registration using the scatter search evolutionary algorithm, Pattern Recognition Letters 27 (2006) 1191–1200.

[35]. M. Dorigo, V. Maniezzo, A. Colorni, The ant system: optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man, and Cybernetics – Part B 26 (1) (1996) 29–41.

[36]. W. Du, B. Li, Multi-strategy ensemble particle swarm optimization for dynamic optimization, Information Sciences 178 (2008) 3096–3109.

[37]. I. Ellabib, P. Calamai, O. Basir, Exchange strategies for multiple ant colony system, Information Sciences 177 (2007) 1248–1264.

[38]. J.D. Farmer, N.H. Packard, A.S. Perelson, The immune system, adaptation and machine learning, Physica D 2 (1986) 187–204.

[39]. R.A. Formato, Central force optimization: a new metaheuristic with applications in applied electromagnetics, Progress in Electromagnetics Research 77 (2007) 425–491.

[40]. R.A. Formato, Central force optimization: a new nature inspired computational framework for multidimensional search and optimization, Studies in Computational Intelligence 129 (2008) 221–238.
[41]. V. Gazi, K.M. Passino, Stability analysis of social foraging swarms, IEEE Transactions on Systems, Man, and Cybernetics – Part B 34 (1) (2004) 539–557.
[42]. C. Hamzacebi, Improving genetic algorithms' performance by local search for continuous function optimization, Applied Mathematics and Computation 196 (1) (2008) 309–317.
[43]. R.L. Haupt, E. Haupt, Practical Genetic Algorithms, second ed., John Wiley & Sons, 2004.
[44]. D. Holliday, R. Resnick, J. Walker, Fundamentals of physics, John Wiley and Sons, 1993.
[45]. A. Kalinlia, N. Karabogab, Artificial immune algorithm for IIR filter design, Engineering Applications of Artificial Intelligence 18 (2005) 919–929.
[46]. C. Karakuzu, Fuzzy controller training using particle swarm optimization for nonlinear system control, ISA Transactions 47 (2) (2008) 229–239.
[47]. J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, vol. 4, 1995, pp. 1942–1948.
[48]. I.R. Kenyon, General Relativity, Oxford University Press, 1990.
[49]. D.H. Kim, A. Abraham, J.H. Cho, A hybrid genetic algorithm and bacterial foraging approach for global optimization, Information Sciences 177 (2007) 3918–3937.
[50]. T.H. Kim, I. Maruta, T. Sugie, Robust PID controller tuning based on the constrained particle swarm optimization, Automatica 44 (2008) 1104–1110.
[51]. S. Kirkpatrick, C.D. Gelatto, M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680
[52]. A. Lazar, R.G. Reynolds, Heuristic knowledge discovery for archaeological data using genetic algorithms and rough sets, Artificial IntelligenceLaboratory, Department of Computer Science, Wayne State University, 2003.
[53]. Y.L. Lin, W.D. Chang, J.G. Hsieh, A particle swarm optimization approach to nonlinear rational filter modeling, Expert Systems with Applications 34(2008) 1194–1199.
[54]. Y. Liu, Z. Yi, H. Wu, M. Ye, K. Chen, A tabu search approach for the minimum sum-of-squares clustering problem, Information Sciences 178 (2008) 2680–2704.
[55]. M. Lozano, F. Herrera, J.R. Cano, Replacement strategies to preserve useful diversity in steady-state genetic algorithms, Information Sciences 178(2008) 4421–4433.
[56]. R. Mansouri, F. Nasseri, M. Khorrami, Effective time variation of G in a model universe with variable space dimension, Physics Letters 259 (1999) 194–200.
[57]. H. Nezamabadi-pour, S. Saryazdi, E. Rashedi, Edge detection using ant algorithms, Soft Computing 10 (2006) 623–628.
[58]. E. Rashedi, Gravitational Search Algorithm, M.Sc. Thesis, Shahid Bahonar University of Kerman, Kerman, Iran, 2007 (in Farsi).
[59]. S.J. Russell, P. Norvig, Artificial Intelligence a Modern Approach, Prentice Hall, Upper Saddle River, New Jersy, 1995.
[60]. B. Schutz, Gravity from the Ground Up, Cambridge University Press, 2003.
[61]. X. Tan, B. Bhanu, Fingerprint matching by genetic algorithms, Pattern Recognition 39 (2006) 465–477.
[62]. K.S. Tang, K.F. Man, S. Kwong, Q. He, Genetic algorithms and their applications, IEEE Signal Processing Magazine 13 (6) (1996) 22–37.

[63]. P. Tarasewich, P.R. McMullen, Swarm intelligence: power in numbers, Communication of ACM 45 (2002) 62–67.
[64]. P.K. Tripathi, S. Bandyopadhyay, S.K. Pal, Multi-objective particle swarm optimization with time variant inertia and acceleration coefficients,Information Sciences 177 (2007) 5033–5049.
[65]. D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation 1 (1997) 67–82.
[66]. X. Yao, Y. Liu, G. Lin, Evolutionary programming made faster, IEEE Transactions on Evolutionary Computation 3 (1999) 82–102.
[67]. Harik GR, Lobo FG, Goldberg DE. The compact genetic algorithm. IEEE transactions on evolutionary computation. 1999 Nov;3(4):287-97.
[68]. Martins JP, Delbem AC. Pairwise independence and its impact on Estimation of Distribution Algorithms. Swarm and Evolutionary Computation. 2016 Apr 30;27:80-96.
[69]. Ahmed A, Khan Q, Naeem M, Iqbal M, Anpalagan A, Awais M. An insight to the performance of estimation of distribution algorithm for multiple line outage identification. Swarm and Evolutionary Computation. 2017 Sep 14. DOI : 10.1016/j.swevo.2017.09.006
[70]. Mininno E, Cupertino F, Naso D. Real-valued compact genetic algorithms for embedded microcontroller optimization. IEEE Transactions on Evolutionary Computation. 2008 Apr;12(2):203-19.
[71]. Mininno E, Neri F, Cupertino F, Naso D. Compact differential evolution. IEEE Transactions on Evolutionary Computation. 2011 Feb;15(1):32-54.
[72]. Neri F, Mininno E, Iacca G. Compact particle swarm optimization. Information Sciences. 2013 Aug 1;239:96-121.
[73]. 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.
[74]. Ahn CW, Ramakrishna RS. Elitism-based compact genetic algorithms. IEEE Transactions on Evolutionary Computation. 2003 Aug;7(4):367-85.
[75]. 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.
[76]. 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.
[77]. Aporntewan C, Chongstitvatana P. A hardware implementation of the compact genetic algorithm. InEvolutionary Computation, 2001. Proceedings of the 2001 Congress on 2001 (Vol. 1, pp. 624-629). IEEE.
[78]. Gallagher JC, Vigraham S, Kramer G. A family of compact genetic algorithms for intrinsic evolvable hardware. IEEE Transactions on evolutionary computation. 2004 Apr;8(2):111-26.
[79]. Li, H.-Q., Li, L.: A novel hybrid particle swarm optimization algorithm combined with harmony search for high dimensional optimization problems. In: 2007 International Conference on Intelligent Pervasive Computing (IPC 2007), Jeju Island, Korea, October 11–13.(2007)

[80]. Li, L., Chi, S.-C., Lin, G.: Genetic algorithm incorporated with harmony procedure and its application to searching of non-circular critical slip surface in soil slopes. Shuili Xuebao / Journal of Hydraulic Engineering 36, 913–918 (2005) (in Chinese)

[81]. Jang, W.S., Kang, H.I., Lee, B.H.: Hybrid simplex-harmony search method for optimization problems. In: 2008 IEEE Congress on Evolutionary Computation (CEC 2008), Hong Kong, China, June 1–6 (2008)

[82]. Mahdavi, M., Fesanghary, M., Damangir, E.: An improved harmony search algorithm for solving optimization problems. Applied Mathematics and Computation 188, 1567–1579 (2007)

[83]. Omran, M.G.H., Mahdavi, M.: Global-best harmony search. Applied Mathematics and Computation 198, 643–656 (2008)

[84]. Forsati, R., Haghighat, A.T., Mahdavi, M.: Harmony search based algorithms for bandwidth- delay-constrained least-cost multicast routing. Computer Communications 31,
2505–2519 (2008)

[85]. Assad, Assif, and Kusum Deep. "Applications of Harmony Search Algorithm in Data Mining: A Survey." Proceedings of Fifth International Conference on Soft Computing for Problem Solving. Springer Singapore, 2016.

[86]. Alsaadi, Ali Shanon, Tat-Chee Wan, and Alhamza Munther. "Application of Harmony Search Optimization Algorithm to Improve Connectivity in Wireless Sensor Network with Non-uniform Density." JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 31.4 (2015): 1475-1489

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

[88]. LACHOURI, Farid, KHELIFI, Abdeldjallil, TIGHZERT, Lyes, *et al.*Self-stunding up of humanoid robot using a new intelligent algorithm. In : *Modelling, Identification and Control (ICMIC), 2016 8th International Conference on*. IEEE, 2016. p. 903-908.

[89]. Harik, G. R., Lobo, F. G., & Goldberg, D. E. (1999). The compact genetic algorithm. Evolutionary Computation, IEEE Transactions on, 3(4), 287-297.

[90]. Droste, S., Jansen, T., & Wegener, I. (2002). On the analysis of the (1+ 1) evolutionary algorithm. Theoretical Computer Science, 276(1), 51-81.

[91]. Yang, X.-S.: Firefly algorithm. In: Yang, X.-S. (ed.) Nature-Inspired meta heuristic Algorithms, pp. 79–90. Wiley Online, Library (2008)

[92]. Yang, X.-S.: Firefly algorithms for multimodal optimization. In: Stochastic Algorithms: Foundations and Applications, pp. 169–178. Springer, Berlin (2009)

[93]. E. Rashedi, H. Nezamabadi-pour and S. Saryazdi, BGSA: Binary gravitational search algorithm, Nat. Comput. 9 (2010) 727–745.

[94]. C. Li and J. Zhou, Parameters identi¯cation of hydraulic turbine governing system using improved gravitational search algorithm, Energy Convers. Manag. 52(1) (2011) 374–381.

[95]. . J. S. Chou and A. D. Pham, Smart colony-based support vector regression for enhanced forecasting in civil engineering, Comput.-Aided Civ. Infrastruct. Eng. 30(9) (2015) 715–732.

[96]. J. W. Wu, J. C. R. Tseng and W. N. Tsai, A hybrid linear text segmentation algorithm using hierarchical agglomerative clustering and discrete particle swarm optimization, Integr. Comput.-Aided Eng. 21(1) (2014) 35–46.

[97]. X.-H. Han, X.-M. Chang, L. Quan, X.-Y. Xiong, J.-X. Li, Z.-X. Zhang and Y. Liu, Feature subset selection by gravitational search algorithm optimization, Inf. Sci. 281 (2014) 128–146.

[98]. Wolpert, D.H., Macready, W.G. (1997), "No Free Lunch Theorems for Optimization", *IEEE Transactions on Evolutionary Computation* **1**, 67.

[99]. Geem, Zong Woo, Joong Hoon Kim, and G. V. Loganathan. "A new heuristic optimization algorithm: harmony search." Simulation 76.2 (2001): 60-68.

[100]. Glover, F., Laguna, M., & Martí, R. (2000). Fundamentals of scatter search and path relinking. *Control and cybernetics*, *29*(3), 653-684.

[101]. Duan, Q. Y., Gupta, V. K., & Sorooshian, S. (1993). Shuffled complex evolution approach for effective and efficient global minimization. *Journal of optimization theory and applications*, *76*(3), 501-521.

[102]. Dréo, J., Pétrowski, A., Siarry, P., & Taillard, E. (2003). *Métaheuristiques pour l'optimisation difficile* (p. 356). Eyrolles.

[103]. Hao, J. K., Galinier, P., & Habib, M. (1999). Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. *Revue d'intelligence artificielle*, *13*(2), 283-324.

[104]. Deroussi, L. (2002). *Heuristiques, métaheuristiques et systèmes de voisinage: application à des problèmes théoriques et industriels de type TSP et ordonnancement* (Doctoral dissertation, Clermont-Ferrand 2).

[105]. Polak, E., & Ribiere, G. (1969). Note sur la convergence de méthodes de directions conjuguées. *Revue française d'informatique et de recherche opérationnelle. Série rouge*, *3*(16), 35-43.

[106]. Boudjlida, K. (2012). *Méthodes d'optimisation numérique pour le calcul de stabilité thermodynamique des phases* (Doctoral dissertation, Pau).

[107]. Hao, J. K., Galinier, P., & Habib, M. (1999). Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. *Revue d'intelligence artificielle*, *13*(2), 283-324.

[108]. I. CHARON, O. HUDRY, The noising method: a new method for combinatorial optimization, Operations Research Letters 14 : 133-137, 1993.

[109]. **Lapetoule kévine**, « Les algorithmes métaheuristiques», édition Springer 01 juin 2006.

[110]. Rachid chelouah, «Chapitre I : Généralités sur les méthodes d'optimisation», université de Cergy, France , 15 Décembre 1999.

[111]. Rashedi, E., Nezamabadi-Pour, H., & Saryazdi, S. (2009). GSA: a gravitational search algorithm. *Information sciences*, *179*(13), 2232-2248.

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

[113]. fr.wikipedia.org/wiki/Optimum_de_Pareto.

[114]. fr.wikipedia.org/wiki/M%C3%A9taheuristique#/media/File:Metaheuristique_comportement.svg.

[115……124]fr.wikipedia.org/wiki/M%C3%A9taheuristique#Fonction_objectif_statique_ou_dynamique.