UNIVERSITÉ PARIS-EST

ÉCOLE DOCTORALE MATHEMATIQUES ET STIC

(MSTIC, E.D. 532)

THÈSE DE DOCTORAT
EN INFORMATIQUE

par

Abbas EL DOR

Sujet de la thèse

Perfectionnement des algorithmes d'Optimisation par Essaim Particulaire. Applications en segmentation d'images et en électronique

Thèse soutenue le 5 décembre 2012

Jury :

Damien TRENTESAUX	Professeur des Universités	Université de Valenciennes	Président
Frédéric SAUBION	Professeur des Universités	Université d'Angers	Rapporteur
Marc SEVAUX	Professeur des Universités	Université de Bretagne-Sud	Rapporteur
Gérard BERTHIAU	Professeur des Universités	Université de Nantes	Examinateur
Laurent DEROUSSI	Maître de Conférences	Université de Clermont-Ferrand	Examinateur
Maurice CLERC	Ingénieur consultant	Annecy	Co-directeur de thèse
Patrick SIARRY	Professeur des Universités	Université Paris-Est Créteil	Directeur de thèse

Remerciements

Je voudrais tout d'abord remercier Patrick Siarry, Directeur de l'équipe Traitement de l'Image et du Signal du Laboratoire Images, Signaux et Systèmes Intelligents, et Directeur de cette thèse, ainsi que Maurice Clerc, pour m'avoir donné la possibilité de faire cette thèse, et pour leur encadrement parfait. Ils ont toute ma gratitude pour m'avoir laissé une grande liberté dans mes recherches, aidé et encouragé dans les moments difficiles et m'avoir consacré leur temps malgré leurs occupations.

Je tiens à exprimer ma gratitude à Frédéric Saubion et à Marc Sevaux pour avoir accepté d'être les rapporteurs de cette thèse. Je voudrais également remercier Damien Trentesaux pour avoir accepté de présider mon jury. Je remercie aussi Gérard Berthiau et Laurent Deroussi pour avoir accepté d'examiner mes travaux. J'adresse un grand merci à tous les membres de mon jury, pour avoir ainsi marqué leur intérêt pour mon travail, et pour les remarques qu'ils ont apportées durant la relecture et la soutenance de ma thèse.

Je souhaite également exprimer ma reconnaissance envers tous les membres du LiSSi, pour m'avoir accueilli chaleureusement, et pour toutes les conversations scientifiques ou non que l'on a pu avoir. Un grand merci à Patricia Jamin, Sandrine David et Frédéric Dumont, pour m'avoir aidé à surmonter toutes sortes de problèmes. Je remercie également Brigitte David, de l'ancienne Ecole Doctorale SIMME, et Sylvie Cach, de la nouvelle ED MSTIC.

Je remercie tous mes collègues doctorants, pour la bonne ambiance et leur amitié. Merci en particulier à Kamel Aloui, Ilhem Boussaïd, Julien Lepagnot et Mustapha Dakkak.

J'ai eu la chance d'enseigner en tant que vacataire, pendant mes deux premières années de thèse, à l'Université Paris-Est Créteil. J'ai beaucoup aimé enseigner à l'université, et je souhaite remercier toute l'équipe enseignante avec qui j'ai eu l'honneur de travailler. Merci notamment à Eric Petit et Ahmed Raji.

Je suis très reconnaissant envers les membres de l'IUT de Créteil/Vitry qui m'ont chaleureusement accueilli parmi eux. Un grand merci pour leur bienveillance, et pour tous les bons conseils qu'ils m'ont prodigués.

Ce travail n'aurait pas pu être réalisé sans le soutien de ma famille, que je remercie tout particulièrement. Un grand merci à mes parents, à mes frères et mes sœurs, qui m'ont soutenu tout au long de mes études et dont je serai indéfiniment redevable.

Je tiens aussi à remercier tous ceux qui ont, de près ou de loin, aidé à rendre ce travail possible, que ce soit par des idées ou par des encouragements.

Merci à tous !

Résumé

La résolution satisfaisante d'un problème d'optimisation *difficile*, qui comporte un grand nombre de solutions sous-optimales, justifie souvent le recours à une *métaheuristique* puissante. La majorité des algorithmes utilisés pour résoudre ces problèmes d'optimisation sont les métaheuristiques à population. Parmi celles-ci, nous intéressons à l'Optimisation par Essaim Particulaire (OEP, ou PSO en anglais) qui est apparue en 1995. PSO s'inspire de la dynamique d'animaux se déplaçant en groupes compacts (essaims d'abeilles, vols groupés d'oiseaux, bancs de poissons). Les particules d'un même essaim communiquent entre elles tout au long de la recherche pour construire une solution au problème posé, et ce en s'appuyant sur leur expérience collective. L'algorithme PSO, qui est simple à comprendre, à programmer et à utiliser, se révèle particulièrement efficace pour les problèmes d'optimisation à variables continues. Cependant, comme toutes les métaheuristiques, PSO possède des inconvénients, qui rebutent encore certains utilisateurs. Le problème de convergence prématurée, qui peut conduire les algorithmes de ce type à stagner dans un optimum local, est un de ces inconvénients. L'objectif de cette thèse est de proposer des mécanismes, incorporables à PSO, qui permettent de remédier à cet inconvénient et d'améliorer les performances et l'efficacité de PSO.

Nous proposons dans cette thèse deux algorithmes, nommés *PSO-2S* et *DEPSO-2S*, pour remédier au problème de la convergence prématurée. Ces algorithmes utilisent des idées innovantes et se caractérisent par de nouvelles stratégies d'initialisation dans plusieurs zones, afin d'assurer une bonne couverture de l'espace de recherche par les particules. Toujours dans le cadre de l'amélioration de PSO, nous avons élaboré une nouvelle topologie de voisinage, nommée *Dcluster*, qui organise le réseau de communication entre les particules. Les résultats obtenus sur un jeu de fonctions de test montrent l'efficacité des stratégies mises en œuvre par les différents algorithmes proposés. Enfin, *PSO-2S* est appliqué à des problèmes pratiques, en segmentation d'images et en électronique.

Mots clés : *Optimisation, optimisation continue, métaheuristiques, optimisation par essaim particulaire, segmentation d'image.*

Abstract

The successful resolution of a *difficult* optimization problem, comprising a large number of sub optimal solutions, often justifies the use of powerful *metaheuristics*. A wide range of algorithms used to solve these combinatorial problems belong to the class of population metaheuristics. Among them, Particle Swarm Optimization (PSO), appeared in 1995, is inspired by the movement of individuals in a swarm, like a bee swarm, a bird flock or a fish school. The particles of the same swarm communicate with each other to build a solution to the given problem. This is done by relying on their collective experience. This algorithm, which is easy to understand and implement, is particularly effective for optimization problems with continuous variables. However, like several metaheuristics, PSO shows some drawbacks that make some users avoid it. The premature convergence problem, where the algorithm converges to some local optima and does not progress anymore in order to find better solutions, is one of them. This thesis aims at proposing alternative methods, that can be incorporated in PSO to overcome these problems, and to improve the performance and the efficiency of PSO.

We propose two algorithms, called *PSO-2S* and *DEPSO-2S*, to cope with the premature convergence problem. Both algorithms use innovative ideas and are characterized by new initialization strategies in several areas to ensure good coverage of the search space by particles. To improve the PSO algorithm, we have also developed a new neighborhood topology, called *Dcluster*, which can be seen as the communication network between the particles. The obtained experimental results for some benchmark cases show the effectiveness of the strategies implemented in the proposed algorithms. Finally, *PSO-2S* is applied to real world problems in both image segmentation and electronics fields.

Keywords : *Optimization, continuous optimization, metaheuristics, particle swarm optimization, image segmentation.*

Table des matières

Introduction générale

Les problèmes d'optimisation occupent actuellement une place importante dans la communauté des ingénieurs, des scientifiques et des décideurs. En effet, ce genre de problèmes intervient dans leurs domaines d'activité qui sont très divers, comme la conception de systèmes mécaniques, le traitement d'images, l'électronique ou la recherche opérationnelle.

Un problème d'optimisation est défini par un ensemble de variables, une fonction objectif et un ensemble de contraintes. L'espace de recherche est l'ensemble des solutions possibles du problème. Résoudre un problème d'optimisation consiste à trouver la ou les meilleures solutions (en minimisant et/ou maximisant la/les fonctions objectifs du problème posé), tout en satisfaisant un ensemble de contraintes définies par l'utilisateur. Certains problèmes d'optimisation sont qualifiés de *difficiles*, et leur résolution dans un temps *raisonnable* nécessite l'utilisation d'algorithmes sophistiqués, comme les méthodes approchées (Les heuristiques et les métaheuristiques). Parmi les métaheuristiques destinées à résoudre ces problèmes, plus précisément les problèmes à variables continues, l'optimisation par essaim particulaire (OEP, ou PSO en anglais) est apparue en 1995. C'est le sujet principal de ce travail de thèse.

PSO est une méthode d'optimisation stochastique qui est inspiré d'un comportement social des animaux évoluant en essaim. Ce comportement social est modélisé par une équation mathématique permettant de guider les « particules » durant le processus de déplacement. Le déplacement d'une particule est influencé par trois composantes : la composante d'inertie, la composante cognitive et la composante sociale. Chacune de ces composantes reflète une partie de l'équation. PSO présente l'avantage d'être efficace sur une grande variété de problèmes, sans pour autant que l'utilisateur ait à modifier la structure de base de l'algorithme. Cependant, comme toutes les méthodes stochastiques, PSO présente des inconvénients, qui rebutent encore certains utilisateurs. Parmi ceux-ci, nous citons le problème de convergence prématurée, qui peut conduire les algorithmes de ce type à stagner dans un optimum local. L'objectif de cette thèse est de proposer des méthodes

alternatives, incorporables à PSO, qui permettent de remédier à ce problème et d'améliorer l'efficacité de PSO.

Cette thèse a été préparée au sein de l'Université Paris-Est Créteil (UPEC), dans le *Laboratoire Images, Signaux et Systèmes Intelligents* (LiSSi, E.A. 3956). Elle a été financée par une bourse libanaise (2 ans), puis par un demi-poste d'attaché temporaire à l'enseignement et à la recherche (ATER), partagé entre le LiSSi et l'IUT de Créteil-Vitry. Cette thèse a été dirigée par le Professeur P. Siarry, Directeur de l'équipe *Traitement de l'Image et du Signal*, et co-encadrée par Monsieur M. Clerc, spécialiste de la méthode PSO. Ont participé à mon travail de doctorat Julien Lepagnot, Maître de Conférences à l'Université de Haute-Alsace, à Mulhouse, et Mourad Fakhfakh, Maître de Conférences à l'École Nationale d'Ingénieurs de Sfax, en Tunisie.

Durant cette thèse, nous avons tout d'abord élaboré deux algorithmes, nommés *PSO-2S* et *DEPSO-2S*. Ces deux algorithmes utilisent des idées innovantes et se caractérisent par de nouvelles stratégies d'initialisation dans plusieurs zones de l'espace de recherche. Les algorithmes proposés ont contribué à remédier notablement au problème de la convergence prématurée. Ensuite, toujours dans le cadre de l'amélioration de l'algorithme PSO, nous nous sommes intéressés à la composante sociale de l'équation modélisant le déplacement des particules, plus précisément au réseau de communication entre les particules, appelé aussi « topologie de voisinage ». En effet, nous avons proposé une nouvelle topologie dynamique, appelée *Dcluster*, qui est une combinaison de deux topologies (statique et dynamique). Enfin, les applications étudiées ont porté sur l'utilisation de *PSO-2S* pour résoudre des problèmes de dimensionnement de circuits électroniques et de traitement d'images.

Le mémoire se divise en quatre chapitres. Le premier est consacré à l'étude bibliographique des méthodes d'optimisation, dont la plus grande partie concerne l'optimisation par essaim particulaire. Le deuxième chapitre est dédié aux extensions que nous avons apportées à la méthode PSO (*PSO-2S* et sa variante *DEPSO-2S*, l'hybridation de PSO avec un algorithme d'évolution différentielle (ou *Differential Evolution* en anglais). Le troisième chapitre porte sur la notion de voisinage dans PSO et la nouvelle topologie proposée *Dcluster*. Enfin, le dernier chapitre correspond aux applications de *PSO-2S* à des problèmes « réels ».

Dans le premier chapitre, nous décrivons d'abord le cadre de l'optimisation difficile, puis nous présentons un état de l'art des métaheuristiques d'optimisation pour les problèmes mono-objectifs : l'algorithme du recuit simulé, la recherche tabou, les algorithmes

évolutionnaires, l'algorithme de colonies de fourmis et l'algorithme à évolution différentielle. Ensuite, nous détaillons l'algorithme d'Optimisation par Essaim Particulaire (OEP, ou PSO en anglais). Enfin, nous présentons des fonctions de test, qui sont utilisées dans le deuxième et le troisième chapitre, pour mesurer les performances des algorithmes.

Dans le deuxième chapitre, nous décrivons l'algorithme *PSO-2S*, que nous avons élaboré durant cette thèse. Il se caractérise par une nouvelle stratégie d'initialisation dans plusieurs zones de l'espace de recherche. Cette stratégie est basée sur l'utilisation de deux types d'essaims de particules (auxiliaire et principal), en considérant les particules comme des électrons. Nous commençons par décrire *PSO-2S* en détail, puis nous en présentons une analyse expérimentale. Une comparaison de ses performances avec celles des autres algorithmes d'optimisation proposés dans la littérature est également effectué. Ensuite, une variante améliorée de *PSO-2S*, nommée *DEPSO-2S*, est décrite et analysée.

Dans le troisième chapitre, nous nous concentrons d'abord sur le concept de voisinage, en étudiant les caractéristiques des échanges d'informations entre les particules. Ensuite, nous présentons la topologie *Dcluster*, que nous avons élaborée durant cette thèse. Enfin, nous effectuons une analyse expérimentale, ainsi qu'une comparaison de ses performances avec celles d'autres algorithmes utilisant différentes topologies connues dans la littérature de PSO.

Le quatrième chapitre est consacré à l'application de *PSO-2S* en segmentation d'images et en électronique. Dans la première partie de ce chapitre, nous présentons la première application de *PSO-2S* en segmentation d'images. Le but est de segmenter des images multi-classes dont le nombre de classes est connu a priori. Dans le cadre de nos travaux, nous nous sommes limités à l'approche par seuillage d'images pour déterminer les seuils des différentes classes. *PSO-2S* est utilisé comme méthode d'optimisation pour minimiser un critère qui permet de déterminer ces seuils. La deuxième partie de ce chapitre porte sur l'application de *PSO-2S* au dimensionnement de circuits électroniques. Le but est de trouver les dimensions des transistors qui permettent de faire fonctionner de manière optimale les circuits étudiés. *PSO-2S* est utilisé pour minimiser la résistance parasite et maximiser la fréquence de coupure d'un convoyeur de courant de seconde génération CCII+.

Enfin, dans la conclusion générale du manuscrit, nous récapitulons nos principales contributions, puis nous exposons quelques perspectives pour améliorer les performances des algorithmes déjà développés.

CHAPITRE UN

MÉTAHEURISTIQUES D'OPTIMISATION : ÉTAT DE L'ART

1.1 Introduction

L'un des principes les plus fondamentaux de notre monde, qui occupe aussi une place importante dans le monde informatique, industriel, etc., est la recherche d'un état optimal. En effet, de nombreux problèmes scientifiques, sociaux, économiques et techniques ont des paramètres qui peuvent être ajustés pour produire un résultat plus souhaitable. Ceux-ci peuvent être considérés comme des problèmes d'optimisation et leur résolution est un sujet central en recherche opérationnelle. Des techniques ont été conçues pour résoudre ces problèmes, notamment les problèmes « difficiles » (par exemple ceux qui présentent de nombreux extrema locaux pauvres), en déterminant des solutions qui ne sont pas rigoureusement optimales, mais qui s'en approchent. Ces méthodes, appelées *heuristiques et métaheuristiques*, se basent généralement sur des phénomènes physiques, biologiques, socio-psychologiques, et peuvent faire appel au hasard.

Ce chapitre est structuré comme suit. Dans la section 1.2, nous décrivons d'abord le cadre de l'*optimisation difficile*, puis nous présentons un état de l'art des métaheuristiques d'optimisation pour les problèmes mono-objectifs, telles que l'algorithme du recuit simulé, la recherche tabou, les algorithmes évolutionnaires, l'algorithme de colonies de fourmis et l'algorithme à évolution différentielle.

Dans la section 1.3, nous détaillons l'algorithme d'Optimisation par Essaim Particulaire (OEP, ou PSO en anglais), qui constitue le sujet principal de ce travail de thèse. L'Optimisation par Essaim Particulaire est une métaheuristiques inventée par Kennedy et Eberhart [Kenn 95] en 1995. Elle s'inspire du comportement social des animaux évoluant en essaim. Les différentes particules d'un essaim communiquent directement avec leurs voisines et construisent ainsi une solution à un problème, en s'appuyant sur leur expérience collective.

Enfin, dans la section 1.4, nous présentons des fonctions analytiques (principales fonctions de test de CEC'05 « *Congress on Evolutionary Computation* » et problèmes réels) dont les minima globaux et locaux sont connus. Ces fonctions sont utilisées dans le deuxième et le troisième chapitre pour mesurer les performances des algorithmes.

1.2 Métaheuristiques pour l'optimisation mono-objectif difficile

1.2.1 Problème d'optimisation

Un problème d'optimisation au sens général est défini par un ensemble de variables, une fonction objectif f et un ensemble de contraintes d'égalité (ou d'inégalité) que les variables doivent satisfaire. L'ensemble des solutions possibles du problème forme l'espace de recherche E, où chaque dimension correspond à une variable. L'espace de recherche E est fini puisque le décideur précise exactement le domaine de définition de chaque variable entre autres pour des raisons de temps de calcul. Suivant le problème posé, nous cherchons à minimiser ou maximiser la fonction objectif f. Un problème d'optimisation peut être statique ou dynamique (i.e. la fonction objectif change avec le temps), mono-objectif ou multi-objectif (i.e. plusieurs fonctions objectifs doivent être optimisées) et avec ou sans contraintes. Ainsi, un problème d'optimisation peut être, par exemple, à la fois continu et dynamique.

Il existe de nombreuses méthodes déterministes (ou exactes) permettant de résoudre certains types de problèmes d'optimisation et d'obtenir la solution optimale du problème, en un temps raisonnable. Ces méthodes nécessitent que la fonction objectif présente un certain nombre de caractéristiques telles que la convexité, la continuité ou la dérivabilité. Nous pouvons citer, parmi les méthodes les plus connues, les méthodes de programmation linéaire [Schr 98], quadratique [Noce 00] et/ou dynamique [Bert 95], la méthode de Newton [Noce 00], la méthode du simplex [Neld 65] ou encore la méthode du gradient [Avri 76].

1.2.2 Optimisation difficile

Les méthodes de résolution exacte ne sont pas adaptées à toutes les problématiques, et donc certains problèmes sont trop complexes à résoudre par ces méthodes. Parmi ces

problématiques, nous pouvons citer l'existence de discontinuités, l'absence de convexité stricte, la non-dérivabilité, la présence de bruit ou encore la fonction objectif peut ne pas être définie précisément (e.g. quand c'est un cout). En outre, les méthodes de résolution exacte peuvent avoir un temps de résolution trop long. Dans ce cas, le problème d'optimisation est dit *difficile*, car aucune méthode exacte n'est capable de le résoudre en un temps raisonnable. Il est alors nécessaire d'avoir recours à des heuristiques de résolution dites méthodes *approchées*, qui fournissent un résultat sans garantie de l'optimalité.

L'optimisation difficile peut se diviser en deux types de problèmes : les problèmes à variables discrètes et les problèmes à variables continues :

- Un problème d'optimisation à variables discrètes consiste à trouver, dans un ensemble discret, une solution réalisable. Le problème majeur réside ici dans le fait que le nombre de solutions réalisables est généralement très élevé, donc il est très difficile de trouver la meilleure solution dans un temps « raisonnable ». Les problèmes à variables discrètes rassemble les problèmes de type NP-complets, tels que le problème du voyageur de commerce. L'utilisation d'algorithmes d'optimisation stochastiques, tels que les métaheuristiques, permettant de trouver une solution approchée en un temps raisonnable est donc courante.
- Dans le deuxième type, les variables du problème d'optimisation sont continues. C'est le cas par exemple des problèmes d'identification, où l'on cherche à minimiser l'erreur entre le modèle d'un système et des observations expérimentales. Ce type de problèmes est moins formalisé que le précédent, mais un certain nombre de difficultés sont bien connues, comme l'existence de nombreuses variables présentant des corrélations non identifiées, la présence de bruit ou plus généralement une fonction objectif accessible par simulation uniquement. En effet, la grande majorité des métaheuristiques existantes ont été créées pour résoudre des problèmes à variables discrètes [Dro 03]. Cependant, la nécessité croissante de méthodes pouvant résoudre ce type de problèmes a poussé les chercheurs à adapter leurs méthodes au cas continu. Il est à noter qu'il existe des problèmes à variables mixtes (i.e. le problème présente à la fois des variables discrètes et continues).

1.2.3 Algorithmes d'optimisation approchée

1.2.3.1 Heuristiques

L'utilisation de méthodes exactes n'est pas toujours possible pour un problème donné à cause d'un certain nombre de contraintes, telles que le temps de calcul souvent important ou bien la difficulté, voire l'impossibilité dans certains cas, d'une définition séparée du problème. Pour faire face à ces contraintes, nous utilisons des méthodes approchées, appelées *heuristiques*. Le terme heuristique dérive du grec ancien *heuriskêin* et signifie « trouver ». Il qualifie tout ce qui sert à la découverte et à l'exploitation. Il est à souligner ici qu'une méthode heuristique peut être déterministe ou stochastique.

Une heuristique est un algorithme qui fournit rapidement (en un temps polynomial) une solution approchée et réalisable, pas nécessairement optimale, pour un problème d'optimisation difficile. Cette méthode approximative est le contraire d'un algorithme exact qui donne une solution optimale pour un problème donné.

Il y a une multitude d'heuristiques qui ont déjà été proposées dans la littérature. Nous pouvons citer des heuristiques très simples telles que les algorithmes gloutons [Corm 90, DeVo 96] ou les approches par amélioration itérative [Basi 75]. Le principe des méthodes gloutonnes est de faire une succession de choix optimaux localement, jusqu'à ce que l'on ne puisse plus améliorer la solution, et ce, sans retour en arrière possible. Le fonctionnement d'une heuristique gloutonne est similaire à celui d'un algorithme glouton exact. La différence réside dans le fait que nous n'imposons plus que la solution obtenue soit optimale, nous obtenons donc un algorithme d'approximation.

1.2.3.2 Métaheuristiques

Des heuristiques plus poussées, adaptables à un grand nombre de problèmes différents, sans changements majeurs dans l'algorithme, ont été mises au point et ont donné naissance à une nouvelle famille d'algorithmes d'optimisation stochastiques : les *méta-heuristiques*. Le terme méta-heuristique a été inventé par Fred Glover en 1986, lors de la conception de la recherche tabou [Glov 86].

Les métaheuristiques forment une famille d'algorithmes d'optimisation visant à résoudre des problèmes d'optimisation difficile, pour lesquels nous ne connaissons pas de méthodes classiques plus efficaces. Elles sont généralement utilisées comme des méthodes génériques pouvant optimiser une large gamme de problèmes différents, d'où le qualificatif *méta*. Leur capacité à optimiser un problème à partir d'un nombre minimal d'informa-

tions est contrebalancée par le fait qu'elles n'offrent aucune garantie quant à l'optimalité de la meilleure solution trouvée. Cependant, du point de vue de la recherche opérationnelle, ce constat n'est pas forcément un désavantage, puisque l'on préfère toujours une approximation de l'optimum global trouvée rapidement à une valeur exacte trouvée dans un temps rédhibitoire.

Il existe un grand nombre de métaheuristiques différentes, allant de la simple recherche locale à des algorithmes complexes de recherche globale. La plupart des méta-heuristiques utilisent des processus aléatoires comme moyens de récolter de l'information et de faire face à des problèmes comme l'explosion combinatoire. Les métaheuristiques peuvent être considérées comme des algorithmes stochastiques itératifs, où elles manipulent une ou plusieurs solutions à la recherche de l'optimum. 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 ou en une précision demandée. Ces méthodes tirent leur intérêt de leur capacité à éviter les optima locaux, soit en acceptant des dégradations de la fonction objectif au cours du traitement, soit en utilisant une population de points comme méthode de recherche.

Les métaheuristiques sont souvent inspirées de processus naturels qui relèvent de la physique (l'algorithme du recuit simulé), de la biologie de l'évolution (les algorithmes génétiques) ou encore de l'éthologie (les algorithmes de colonies de fourmis ou l'optimisation par essaim particulaire).

Les métaheuristiques se caractérisant par leur capacité à résoudre des problèmes très divers, elles se prêtent naturellement à des extensions. Pour illustrer celles-ci, nous pouvons citer :

- Les métaheuristiques pour l'*optimisation multiobjectif* [Coll 02] : où il faut optimiser plusieurs objectifs contradictoires. Le but ne consiste pas ici à trouver un optimum global, mais à trouver un ensemble d'optima, qui forment une surface de compromis pour les différents objectifs du problème ;
- Les métaheuristiques pour l'*optimisation multimodale* [Gold 87] : où l'on ne cherche plus l'optimum global, mais l'ensemble des meilleurs optima globaux et/ou locaux ;
- Les métaheuristiques pour l'*optimisation de problèmes bruités* : où il existe une incertitude sur le calcul de la fonction objectif, dont il faut tenir compte dans la recherche de l'optimum ;

- Les métaheuristiques pour l'*optimisation dynamique* [Bran 02] : où la fonction objectif varie dans le temps, ce qui nécessite d'approcher l'optimum à chaque pas de temps ;
- Les métaheuristiques *hybrides* [Talb 02] : qui consistent à combiner différentes métaheuristiques, afin de tirer profit des avantages respectifs ;
- Les métaheuristiques *parallèles* [Alba 05] : où l'on cherche à accélérer le calcul, en répartissant la charge de calcul sur des unités fonctionnant de concert. Le problème revient alors à adapter les métaheuristiques pour qu'elles soient distribuées.

En général, l'utilisateur demande des méthodes efficaces et rapides permettant d'atteindre un optimum avec une précision acceptable dans un temps raisonnable, mais il a besoin aussi des méthodes simples à utiliser. Un des enjeux des métaheuristiques est donc de faciliter le choix d'une méthode et de simplifier son réglage pour l'adapter au mieux à un problème posé. De nombreuses méthodes existent dans la littérature, certaines, parmi les plus courantes, seront présentées plus en détail dans la section 1.2.4.

1.2.4 Différentes approches en optimisation statique

1.2.4.1 Algorithme du recuit simulé

Le recuit simulé est une méthode empirique inspirée d'un processus utilisé en métallurgie (appelé le *recuit*) où, pour atteindre les états de basse énergie d'un solide, on chauffe celui-ci jusqu'à des températures élevées, avant de le laisser refroidir lentement.

L'algorithme du recuit simulé a été proposé par Kirkpatrick et al. [Kirk 83] (et indépendamment, Cerny [Cern 85]). La description classique du recuit simulé le présente comme un algorithme probabiliste, où un point évolue dans l'espace de recherche. Le recuit simulé s'appuie sur l'algorithme de Metropolis [Metr 53, Hast 70] décrit par l'Algorithme 1.1, qui permet de décrire l'évolution d'un système en thermodynamique. Cette procédure permet de sortir des minima locaux avec une probabilité élevée si la température T est élevée et, quand l'algorithme atteint de basses températures, de conserver les états les plus probables. Ici, la méthode de Metropolis (ou toute autre méthode d'échantillonnage [Creu 83, Okam 95]) tient lieu de diversification, associée à la décroissance de température, qui contrôle l'intensification. L'algorithme de recuit simulé est résumé en Algorithme 1.2.

Algorithme de Metropolis

Initialiser un point de départ x_0 et une température T
pour i=1 à n **faire**
  tant que x_i n'est pas accepté **faire**
    si $f(x_i) \leq f(x_{i-1})$ **alors**
      accepter x_i
    fin
    si $f(x_i) > f(x_{i-1})$ **alors**
      accepter x_i avec la probabilité $e^{\frac{f(x_i)-f(x_{i-1})}{T}}$
    fin
  fin
fin

ALGORITHME 1.1: Algorithme de Metropolis.

Les inconvénients du recuit simulé résident dans :

1. Les « réglages », car l'algorithme dispose d'un nombre élevé de paramètres (température initiale, règle de décroissance de la température, durées des paliers de température, etc.) qui rendent les réglages de l'algorithme assez empiriques ;
2. Les « temps de calcul », qui peuvent devenir très importants.

Cependant, il existe des études qui s'attachent à régler de manière optimale les paramètres de l'algorithme [Cour 94]. Par ailleurs, pour surmonter le problème de temps de calcul, plusieurs méthodes de parallélisation des calculs ont été introduites [Azen 92].

Par contre, la méthode du recuit simulé a l'avantage d'être souple vis-à-vis des évolutions du problème et facile à implémenter. Elle a donné d'excellents résultats pour un certain nombre de problèmes, le plus souvent de grande taille. Bien qu'initialement créée pour fonctionner avec des variables discrètes, la méthode du recuit simulé possède des versions continues [Cour 94].

1.2.4.2 Algorithme de recherche tabou

La recherche tabou est une métaheuristique itérative qualifiée de « recherche locale » au sens large. L'idée de la recherche tabou consiste, à partir d'une position donnée, à en explorer le voisinage et à choisir le voisin qui minimise la fonction objectif.

Algorithme du recuit simulé

```
1  Déterminer une configuration aléatoire S
2  Choix des mécanismes de perturbation d'une configuration
3  Initialiser la température T
4  tant que le critère d'arrêt n'est pas satisfait faire
5      tant que l'équilibre n'est pas atteint faire
6          Tirer une nouvelle configuration S'
7          Appliquer la règle de Metropolis
8          si f(S') < f(S) alors
9              S_min = S'
10             f_min = f(S')
11         fin
12     fin
13     Décroître la température
14 fin
```

ALGORITHME 1.2: Algorithme du recuit simulé.

L'algorithme de recherche tabou a été introduit par Fred Glover en 1986 [Glov 86]. Le principe de base de cet algorithme est de pouvoir poursuivre la recherche de solutions même lorsqu'un optimum local est rencontré et ce, en permettant des déplacements qui n'améliorent pas la solution, et en utilisant le principe de la mémoire pour éviter les retours en arrière (mouvements cycliques). Il est essentiel de noter que cette opération peut conduire à augmenter la valeur de la fonction (dans un problème de minimisation) ; c'est le cas lorsque tous les points du voisinage ont une valeur plus élevée. C'est à partir de ce mécanisme que l'on sort d'un minimum local. En effet, comme l'algorithme de recuit simulé, la méthode de recherche tabou fonctionne avec une seule configuration courante, qui est actualisée au cours des itérations successives. La nouveauté ici est que, pour éviter le risque de retour à une configuration déjà visitée, on tient à jour une liste de mouvements interdits (ou de solutions interdites), appelée « liste tabou ». Le rôle de cette dernière évolue au cours de la résolution pour passer de l'exploration (aussi appelée « diversification ») à l'exploitation (également appelée « intensification »). Cette liste contient m mouvements $(t \rightarrow s)$ qui sont les inverses des m derniers mouvements $(s \rightarrow t)$ effectués. L'algorithme modélise ainsi une forme primaire de mémoire à court terme. L'algorithme de recherche

tabou peut être résumé par l'Algorithme 1.3.

Dans sa forme de base, l'algorithme de recherche tabou présente l'avantage de comporter moins de paramètres que l'algorithme de recuit simulé. Cependant, l'algorithme n'étant pas toujours performant, il est souvent approprié de lui ajouter des processus d'intensification et/ou de diversification, qui introduisent de nouveaux paramètres de contrôle [Glov 97].

Algorithme de recherche tabou

Déterminer une configuration aléatoire s
Initialiser une liste tabou vide
tant que le critère d'arrêt n'est pas satisfait **faire**
    Perturbation de s suivant N mouvements non tabous
    Évaluation des N voisins
    Sélection du meilleur voisin t
    Actualisation de la meilleure position connue s^*
    Insertion du mouvement $t \rightarrow s$ dans la liste tabou
    $s = t$
fin

ALGORITHME 1.3: Algorithme de recherche tabou.

1.2.4.3 Les algorithmes évolutionnaires

Les algorithmes évolutionnaires (AEs), élaborés au cours des années 1950 [Fras 57], sont des techniques de recherche inspirées par l'évolution biologique des espèces. Ils s'inspirent de l'évolution des êtres vivants (la théorie Darwinienne de la sélection naturelle des espèces) pour résoudre des problèmes d'optimisation. L'idée ici est que, les individus qui ont hérité des caractères bien adaptés à leur milieu ont tendance à vivre assez longtemps pour se reproduire, alors que les plus faibles ont tendance à disparaître.

Au cours des années 1970, avec l'avènement des calculateurs de forte puissance, de nombreuses approches de modélisation de l'évolution ont été réalisées. Nous pouvons citer :

– Les *statégies d'évolution* [Rech 65, Beye 01] qui ont été conçues pour résoudre des problèmes à variables continues. Elles sont axées sur la modélisation des paramètres

stratégiques qui contrôlent la variation dans l'évolution, autrement dit « l'évolution de l'évolution » ;

- La *programmation évolutionnaire* [Foge 62, Foge 66], qui vise à faire évoluer les structures d'automates à états finis par des successions de croisements et de mutations ;
- Les *algorithmes génétiques* [Holl 75], qui ont été conçus pour résoudre des problèmes d'optimisation à variables discrètes, en modélisant l'évolution génétique ;
- La *programmation génétique* [Koza 89, Koza 90], basée sur les algorithmes génétiques, mais où les individus (ou chromosomes) sont des programmes informatiques, représentés en utilisant une structure d'arbre ;
- L'*évolution différentielle* [Stor 97, Pric 05], qui a été conçue pour résoudre des problèmes à variables continues. Sa stratégie consiste à biaiser un opérateur de mutation, appliqué à un individu, en fonction des différences calculées avec d'autres individus sélectionnés aléatoirement. Une description complète et détaillée de cet algorithme est donnée dans la section 1.2.4.5.

Les approches évolutionnaires s'appuient sur un modèle commun présenté en Algorithme 1.4. Les individus soumis à l'évolution sont des solutions possibles du problème posé. L'ensemble de ces individus constitue une *population*. Cette population évolue durant une succession d'itérations, appelées *générations*. Au cours de chaque génération, une série d'opérateurs est appliquée aux individus, pour créer la population de la génération suivante. Chaque opérateur utilise un ou plusieurs individus, appelés *parents*, pour engendrer de nouveaux individus, appelés *enfants*. A la fin de chaque génération, une sélection d'enfants créés durant la génération remplace un sous-ensemble d'individus de la population.

Un algorithme évolutionnaire dispose de trois opérateurs principaux :

1. Un opérateur de sélection, qui favorise la propagation des meilleures solutions dans la population, tout en maintenant une certaine diversité génétique au sein de celle-ci ;
2. Un opérateur de croisement, mis en œuvre lors de la phase de création des enfants. Son but est d'échanger les gènes des différents parents pour créer les enfants. Un exemple de croisement simple pour des individus codés en représentation binaire est présenté dans la figure 1.1 ;

Algorithme évolutionnaire générique

Initialisation de la population de μ individus
Evaluation des μ individus
tant que le critère d'arrêt n'est pas satisfait **faire**
    Sélection de ρ individus en vue de la phase de reproduction
    Croisement des ρ individus sélectionnés
    Mutation des λ enfants obtenus
    Evaluation des λ enfants obtenus
    Sélection pour le remplacement
fin

ALGORITHME 1.4: Algorithme évolutionnaire générique.

3. Un opérateur de mutation, consistant à tirer aléatoirement une composante de l'individu parent et à la remplacer par une valeur aléatoire. L'apport d'un caractère aléatoire à la création de la descendance permet ainsi de maintenir une certaine diversité dans la population. La figure 1.2 montre un exemple de mutation, pour un individu codé en représentation binaire.

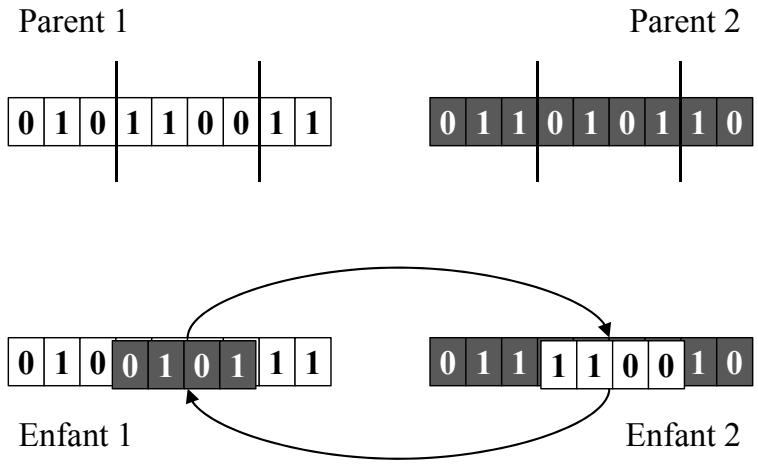


FIGURE 1.1: Exemple d'opérateur de croisement en représentation binaire.

Le principe d'un algorithme évolutionnaire se décrit simplement (voir figure 1.3).

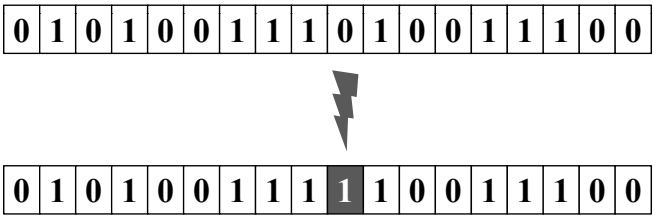


FIGURE 1.2: Exemple d'opérateur de mutation en représentation binaire.

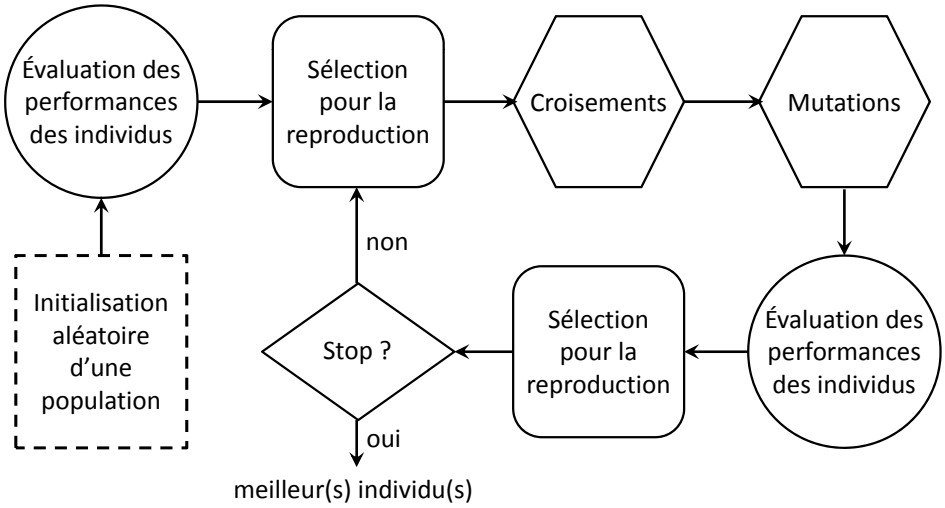


FIGURE 1.3: Principe d'un algorithme évolutionnaire standard.

1.2.4.4 Les algorithmes de colonies de fourmis

Les algorithmes de colonies de fourmis sont des algorithmes inspirés du comportement des fourmis et constituent une famille de métaheuristiques d'optimisation pour résoudre naturellement des problèmes complexes. Une telle aptitude s'avère possible en raison de la capacité des fourmis à communiquer entre elles indirectement, par le dépôt dans l'environnement de substances chimiques, appelées *phéromones*. Ce type de communication indirecte est appelé *stigmergie*. En anglais, le terme consacré à la principale classe d'algorithmes est *Ant Colony Optimization* (ACO).

La principale illustration de ce phénomène est donnée par la figure 1.4 : un obstacle est placé sur le trajet des fourmis qui, après une étape d'exploration, finiront par emprunter le plus court chemin entre le nid et la source de nourriture [Goss 89]. Les fourmis qui sont retournées le plus rapidement au nid en passant par la source de nourriture sont celles qui ont emprunté le chemin le plus court. Il en découle que la quantité de phéromones déposées par unité de temps sur ce trajet est plus importante que sur les autres. Par ailleurs,

une fourmi est d'autant plus attirée à un certain endroit que le taux de phéromones y est important. De ce fait, le plus court chemin a une probabilité plus importante d'être emprunté par les fourmis que les autres chemins et sera donc, à terme, emprunté par toutes les fourmis.

Le premier algorithme d'optimisation s'inspirant de cette analogie a été proposé par Colorni, Dorigo et Maniezzo [Colo 91, Dori 96], afin de résoudre le problème du voyageur de commerce. L'Algorithme 1.5 résume l'approche par colonies de fourmis proposée par les auteurs. Si l'on considère un problème de voyageur de commerce à N villes, chaque fourmi k parcourt le graphe et construit un trajet de longueur $n = N$. Pour chaque fourmi, le trajet d'une ville i à une ville j dépend de :

- La liste des déplacements possibles J_i^k, quand la fourmi k est sur la ville i ;
- L'inverse de la distance entre les villes $\eta_{ij} = \frac{1}{d_{ij}}$, appelée *visibilité*. Cette information est utilisée pour diriger les fourmis vers des villes proches et, ainsi, éviter de trop longs déplacements ;
- La quantité de phéromones déposée sur l'arête reliant deux villes $\tau_{ij}(t)$, appelée *intensité de la piste*. Cette quantité définit l'attractivité d'une piste et est modifiée après le passage d'une fourmi. C'est en quelque sorte la mémoire du système.

La règle de déplacement est la suivante :

$$p_{ij}^k(t) = \begin{cases} \frac{(\tau_{ij}(t))^\alpha (\eta_{ij})^\beta}{\sum_{l \in J_i^k} (\tau_{il}(t))^\alpha (\eta_{il})^\beta} & \text{si } j \in J_i^k \\ \\ 0 & \text{sinon} \end{cases} \tag{1.2.1}$$

où α et β sont deux paramètres contrôlant l'importance relative de l'intensité de la piste et de la visibilité.

Après un tour complet, chaque fourmi dépose une quantité de phéromones $\Delta\tau_{ij}^k(t)$ sur l'ensemble de son parcours. Cette quantité dépend de la *qualité* de la solution trouvée et est définie par :

$$\Delta\tau_{ij}^k(t) = \begin{cases} \frac{Q}{L^k(t)} & \text{si } (i,j) \in T^k(t) \\ \\ 0 & \text{sinon} \end{cases} \tag{1.2.2}$$

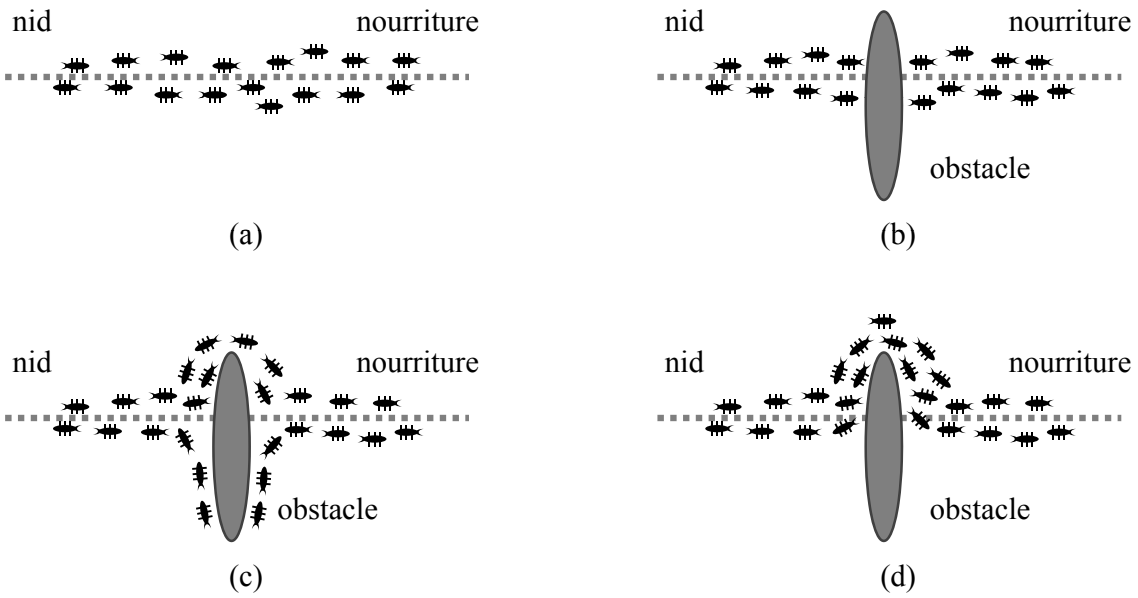


FIGURE 1.4: Illustration de la capacité des fourmis à chercher de la nourriture en minimisant leur parcours. (a) Recherche sans obstacle, (b) Apparition d'un obstacle, (c) Recherche du chemin optimal, (d) Chemin optimal trouvé.

ACO

```
t ← 1
tant que le critère d'arrêt n'est pas satisfait faire
    pour k = 1 à m faire
        Choisir une ville au hasard
        pour chaque ville non visitée i faire
            Choisir une ville j dans la liste J_i^k des villes restantes selon (1.2.1)
        fin
        Déposer une quantité de phéromones Δτ_ij^k(t) sur le trajet T^k(t) conformément à (1.2.2)
    fin
    Evaporer les phéromones selon (1.2.3)
    t ← t + 1
fin
```

ALGORITHME 1.5: Algorithme de colonies de fourmis pour le problème du voyageur de commerce.

où $T^k(t)$ est le trajet effectué par la fourmi k à l'itération t, $L^k(t)$ est la longueur de $T^k(t)$ et Q est un paramètre de réglage.

Enfin, il est nécessaire d'introduire un processus d'évaporation des phéromones. En effet, pour éviter de rester piégé dans des optima locaux, il est nécessaire qu'une fourmi « oublie » les mauvaises solutions. La règle de mise à jour est la suivante :

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t) \tag{1.2.3}$$

où $\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^k(t)$, m est le nombre de fourmis et ρ est un paramètre de réglage.

La démarche initiée par cette analogie a été étendue à la résolution d'autres problèmes d'optimisation, discrets et continus [Dori 05, Dro 03]. ACO présente plusieurs caractéristiques intéressantes, telles que le parallélisme intrinsèque élevé, la robustesse (une colonie peut maintenir une recherche efficace, même si certains de ses individus sont défaillants) ou encore la décentralisation (les fourmis n'obéissent pas à une autorité centralisée).

1.2.4.5 Algorithme à évolution différentielle

L'évolution différentielle (*Differential Evolution* DE) est une métaheuristique stochastique d'optimisation qui a été inspirée par les algorithmes génétiques et des stratégies évolutionnaires combinées avec une technique géométrique de recherche. Les algorithmes génétiques changent la structure des individus en utilisant la mutation et le croisement, alors que les stratégies évolutionnaires réalisent l'auto-adaptation par une manipulation géométrique des individus. Ces idées ont été mises en œuvre grâce à une opération simple, mais puissante, de mutation de vecteurs, proposée en 1995 par K. Price et R. Storn [Stor 97]. Même si, à l'origine, la méthode de l'évolution différentielle était conçue pour les problèmes d'optimisation continus et sans contraintes, ses extensions actuelles peuvent permettre de traiter les problèmes à variables mixtes et gèrent les contraintes non linéaires [Lamp].

Dans la méthode DE, la population initiale est générée par tirage aléatoire uniforme sur l'ensemble des valeurs possibles de chaque variable. Les bornes inférieures et supérieures des variables sont spécifiées par l'utilisateur selon la nature du problème. Après l'initialisation, l'algorithme effectue une série de transformations sur les individus, dans un processus appelé *évolution*.

La population contient N individus. Chaque individu $x_{i,G}$ est un vecteur de dimension D, où G désigne la génération :

$$x_{i,G} = (x_{1i,G}, x_{2i,G}, ..., x_{Di,G}) \text{ avec } i = 1, 2, ..., N \tag{1.2.4}$$

Le standard DE utilise trois techniques (mutation, croisement et sélection) comme les algorithmes génétiques. A chaque génération, l'algorithme applique successivement ces trois opérations sur chaque vecteur pour produire un vecteur d'essai (*trial vector*) :

$$u_{i,G+1} = (u_{1i,G+1}, u_{2i,G+1}, ..., u_{Di,G+1}) \text{ avec } i = 1, 2, ..., N \tag{1.2.5}$$

Une opération de sélection permet de choisir les individus à conserver pour la nouvelle génération $(G + 1)$.

a - Mutation

Pour chaque vecteur courant $x_{i,G}$, on génère un vecteur mutant $v_{i,G+1}$ qui peut être créé en utilisant une des stratégies de mutation suivantes :

- Rand/1 :

$$v_{i,G+1} = x_{r_1,G} + F.(x_{r_2,G} - x_{r_3,G}) \tag{1.2.6}$$

- Best/1 :

$$v_{i,G+1} = x_{best,G} + F.(x_{r_1,G} - x_{r_2,G}) \tag{1.2.7}$$

- Current to best/1 :

$$v_{i,G+1} = x_{i,G} + F.(x_{r_1,G} - x_{r_2,G}) + F.(x_{best,G} - x_{i,G}) \tag{1.2.8}$$

- Best/2 :

$$v_{i,G+1} = x_{best,G} + F.(x_{r_1,G} - x_{r_2,G}) + F.(x_{3,G} - x_{4,G}) \tag{1.2.9}$$

- Rand/2 :

$$v_{i,G+1} = x_{r_1,G} + F.(x_{r_2,G} - x_{r_3,G}) + F.(x_{r_4,G} - x_{r_5,G}) \tag{1.2.10}$$

Les indices r_1, r_2, r_3, r_4 et $r_5 \in \{1, 2, \ldots, N\}$ sont des entiers aléatoires et tous différents. Ils sont également choisis différents de l'indice courant i. $x_{best,G}$ est le meilleur individu à la $G^{\text{ème}}$ génération. $F \in [0, 2]$ est une valeur constante, appelée *differential weight*, qui contrôle l'amplification de la variation différentielle de $(x_{r_i,G} - x_{r_j,G})$.

b - Croisement

Après la mutation, une opération de croisement binaire forme le vecteur d'essai final $u_{i,G+1}$, selon le vecteur $x_{i,G}$ et le vecteur mutant correspondant $v_{i,G+1}$. L'opération de croisement est introduite pour augmenter la diversité des vecteurs de paramètres perturbés. Le nouveau vecteur $u_{i,G+1}$ est donné par la formule suivante :

$$u_{ji,G+1} = \begin{cases} v_{1i,G+1} & \text{si } (randb(j) \leq CR) \text{ ou } j = rnbr(i) \\ \\ x_{ji,G} & \text{si } (randb(j) > CR) \text{ et } j \neq rnbr(i) \end{cases} \quad \text{pour tout } j \in \{1,2,\ldots,D\} \tag{1.2.11}$$

où $randb(j)$ est la $j^{\text{ème}}$ valeur procurée un générateur de nombre aléatoire uniforme appartenant à l'intervalle [0,1]. CR est le coefficient de croisement qui appartient à l'intervalle [0,1] et est déterminé par l'utilisateur. $rnbr(i)$ est un indice choisi au hasard dans l'ensemble $\{1, 2, \ldots, N\}$.

c - Sélection

Pour décider quel vecteur, parmi $u_{i,G+1}$ ou $x_{i,G}$, doit être choisi dans la génération $G + 1$, on doit comparer les valeurs de fonction du cout de ces deux vecteurs. En effet, on garde le vecteur ayant la plus petite valeur de fonction du cout en cas de minimisation. Le nouveau vecteur $x_{i,G+1}$ est choisi selon l'expression suivante :

$$x_{i,G+1} = \begin{cases} u_{i,G+1} & \text{si } f(u_{i,G+1}) < f(x_{i,G}) \\ \\ x_{i,G} & \text{sinon} \end{cases} \tag{1.2.12}$$

Il est clair qu'un bon réglage des principaux paramètres de l'algorithme (taille de la population N, facteur de mutation F et facteur de croisement CR) contribue de façon importante à l'efficacité de la méthode. L'auto-adaptation de ces paramètres paraît donc intéressante

pour l'amélioration de l'algorithme. Une comparaison des versions adaptatives et auto-adaptatives de l'algorithme à évolution différentielle (FADE [Liu 05], DESAP [Teo 06], SaDE [Qin 05] et jDE [Bres 06]) est présentée dans [Bres 07].

1.3 Optimisation par Essaim Particulaire

1.3.1 Principe général

L'optimisation par essaim particulaire (OEP), ou *Particle Swarm Optimization* (PSO) en anglais, est un algorithme évolutionnaire qui utilise une population de solutions candidates pour développer une solution optimale au problème. Cet algorithme a été proposé par Russel Eberhart (ingénieur en électricité) et James Kennedy (socio-psychologue) en 1995 [Kenn 95]. Il s'inspire à l'origine du monde du vivant, plus précisément du comportement social des animaux évoluant en essaim, tels que les bancs de poissons et les vols groupés d'oiseaux. En effet, on peut observer chez ces animaux des dynamiques de déplacement relativement complexes, alors qu'individuellement chaque individu a une « intelligence » limitée, et ne dispose que d'une connaissance locale de sa situation dans l'essaim. L'information locale et la mémoire de chaque individu sont utilisées pour décider de son déplacement. Des règles simples, telles que « rester proche des autres individus », « aller dans une même direction » ou « aller à la même vitesse », suffisent pour maintenir la cohésion de l'essaim, et permettent la mise en œuvre de comportements collectifs complexes et adaptatifs.

L'essaim de particules correspond à une population d'agents simples, appelés *particules*. Chaque particule est considérée comme une solution du problème, où elle possède une position (le vecteur solution) et une vitesse. De plus, chaque particule possède une mémoire lui permettant de se souvenir de sa meilleure performance (en position et en valeur) et de la meilleure performance atteinte par les particules « voisines » (informatrices) : chaque particule dispose en effet d'un groupe d'informatrices, historiquement appelé son *voisinage*.

Un essaim de particules, qui sont des solutions potentielles au problème d'optimisation, « survole » l'espace de recherche, à la recherche de l'optimum global. Le déplacement d'une particule est influencé par les trois composantes suivantes :

1. Une composante *d'inertie* : la particule tend à suivre sa direction courante de dépla-

cement ;

2. Une composante *cognitive* : la particule tend à se diriger vers le meilleur site par lequel elle est déjà passée ;
3. Une composante *sociale* : la particule tend à se fier à l'expérience de ses congénères et, ainsi, à se diriger vers le meilleur site déjà atteint par ses voisins.

La stratégie de déplacement d'une particule est illustrée dans la figure 1.5.

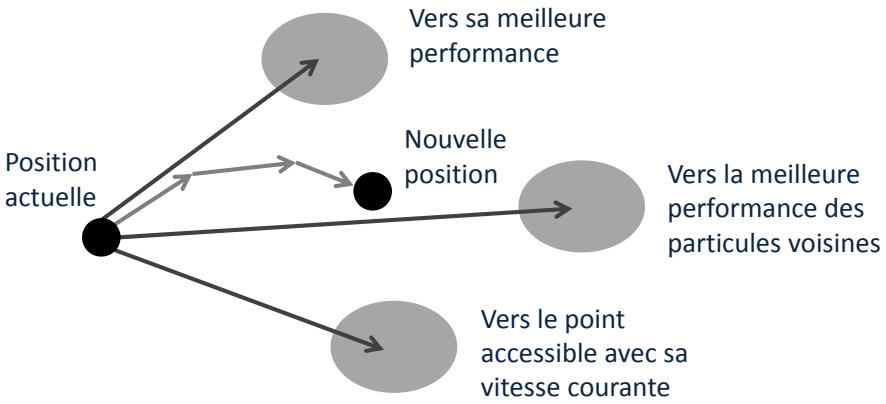


FIGURE 1.5: Déplacement d'une particule.

1.3.2 Formalisation

Dans un espace de recherche de dimension D, la particule i de l'essaim est modélisée par son vecteur position $\vec{x}_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$ et par son vecteur vitesse $\vec{v}_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$. La qualité de sa position est déterminée par la valeur de la fonction objectif en ce point. Cette particule garde en mémoire la meilleure position par laquelle elle est déjà passée, que l'on note $\vec{P}best_i = (pbest_{i1}, pbest_{i2}, \ldots, pbest_{iD})$. La meilleure position atteinte par les particules de l'essaim est notée $\vec{G}best = (gbest_1, gbest_2, \ldots, gbest_D)$. Nous nous référons à la *version globale* de PSO, où toutes les particules de l'essaim sont considérées comme voisines de la particule i, d'où la notation $\vec{G}best$ (*global best*).

Remarque : *le terme de « vitesse » est ici abusif, car les vecteurs $\vec{v}_i$ ne sont pas homogènes à une vitesse. Il serait plus approprié de parler de « direction de déplacement ». Cependant, pour respecter l'analogie avec le monde animal, les auteurs ont préféré utiliser le terme de « vitesse ».*

Au départ de l'algorithme, les particules de l'essaim sont initialisées de manière aléatoire/régulière dans l'espace de recherche du problème. Ensuite, à chaque itération, chaque particule se déplace, en combinant linéairement les trois composantes citées ci-dessus. En effet, à l'itération $t+1$, le vecteur vitesse et le vecteur position sont calculés à partir de l'équation (1.3.1) et de l'équation (1.3.2), respectivement.

$$v_{i,j}^{t+1} = wv_{i,j}^{t} + c_1 r_{1_{i,j}}^{t} \left[pbest_{i,j}^{t} - x_{i,j}^{t}\right] + c_2 r_{2_{i,j}}^{t} \left[gbest_{j}^{t} - x_{i,j}^{t}\right], j \in \{1, 2, ..., D\} \tag{1.3.1}$$

$$x_{i,j}^{t+1} = x_{i,j}^{t} + v_{i,j}^{t+1}, j \in \{1, 2, ..., D\} \tag{1.3.2}$$

où w est une constante, appelée *coefficient d'inertie* ; c_1 et c_2 sont deux constantes, appelées *coefficients d'accélération* ; r_1 et r_2 sont deux nombres aléatoires tirés uniformément dans $[0, 1]$, à chaque itération t et pour chaque dimension j.

Les trois composantes mentionnées ci-dessus (i.e. d'inertie, cognitive et sociale) sont représentées dans l'équation (1.3.1) par les termes suivants :

1. $wv_{i,j}^{t}$ correspond à la composante d'inertie du déplacement, où le paramètre w contrôle l'influence de la direction de déplacement sur le déplacement futur ;
2. $c_1 r_{1_{i,j}}^{t} \left[pbest_{i,j}^{t} - x_{i,j}^{t}\right]$ correspond à la composante cognitive du déplacement, où le paramètre c_1 contrôle le comportement cognitif de la particule ;
3. $c_2 r_{2_{i,j}}^{t} \left[gbest_{j}^{t} - x_{i,j}^{t}\right]$ correspond à la composante sociale du déplacement, où le paramètre c_2 contrôle l'aptitude sociale de la particule.

Une fois le déplacement des particules effectué, les nouvelles positions sont évaluées et les deux vecteurs $\vec{P}best_i$ et $\vec{G}best$ sont mis à jour , à l'itération $t+1$, suivant les deux équations (1.3.3) (dans le cas d'une minimisation) et (1.3.4) (dans une version globale de PSO), respectivement. Cette procédure est présentée dans l'Algorithme 1.6, où N est le nombre de particules de l'essaim.

$$\vec{P}best_i(t+1) = \begin{cases} \vec{P}best_i(t), & \text{si } f(\vec{x}_i(t+1)) \geq \vec{P}best_i(t) \\ \vec{x}_i(t+1), & \text{sinon} \end{cases} \tag{1.3.3}$$

$$\vec{G}best(t+1) = \ arg\ min_{\vec{P}best_i} f(\vec{P}best_i(t+1)), \ \ 1 \leq i \leq N. \tag{1.3.4}$$

OEP

Initialiser aléatoirement N particules : position et vitesse.
Evaluer les positions des particules
Pour chaque particule i, $\vec{P}best_i = \vec{x}_i$
Calculer $\vec{G}best$ selon (1.3.4)
tant que le critère d'arrêt n'est pas satisfait **faire**
    Déplacer les particules selon (1.3.1) et (1.3.2)
    Evaluer les positions des particules
    Mettre à jour $\vec{P}best_i$ et $\vec{G}best$ selon (1.3.3) et (1.3.4)
fin

ALGORITHME 1.6: Algorithme d'optimisation par essaim particulaire.

1.3.3 Améliorations de PSO

1.3.3.1 Confinement des particules

Pour éviter que le système n'« explose » en cas d'amplification trop grande d'oscillations (il est possible que le déplacement d'une particule soit trop rapide et la conduise à sortir de l'espace de recherche), nous pouvons introduire un nouveau paramètre V_{max}, qui permet de limiter la vitesse sur chaque dimension et ainsi de contrôler l'explosion du système [Eber 96]. Notons que cela ne restreint pas les valeurs de x_i à l'intervalle $[V_{imin}, V_{imax}]$, mais limite seulement la distance maximale qu'une particule va parcourir au cours d'une itération. Cette méthode permet de contrôler la divergence de l'algorithme et de réaliser ainsi un compromis efficace entre intensification et diversification.

De plus, une stratégie de confinement des particules peut être introduite. Une telle stratégie permet de ramener une particule sortie de l'espace de recherche à l'intérieur de celui-ci. Dans ce cadre, plusieurs méthodes peuvent être employées :

- La particule est laissée à l'extérieur de l'espace de recherche, mais on n'évalue pas sa fonction objectif. Ainsi, elle ne pourra pas attirer les autres particules en dehors de l'espace de recherche ;
- La particule est arrêtée à la frontière et les composantes associées à sa vitesse sont annulées ;
- La particule rebondit sur la frontière. La particule est stoppée sur la frontière, mais

les composantes correspondantes de la vitesse sont multipliées par un coefficient tiré aléatoirement dans l'intervalle [-1,0].

1.3.3.2 Coefficient de constriction

Des améliorations ont été apportées à l'algorithme de base, notamment du point de vue du contrôle de la divergence : en particulier, l'introduction du paramètre V_{max} que nous avons vu dans le paragraphe précédent, et qui permet de limiter la divergence des particules. En outre, beaucoup d'autres études ont été menées sur la dynamique des particules et qui s'attachent à analyser sous quelles conditions une convergence de l'essaim est assurée [Kenn 98, Ozca 99, Van 01].

La combinaison des paramètres w, c_1 et c_2 permet de régler l'équilibre entre les phases de diversification et d'intensification du processus de recherche [Kenn 01]. Dans [Cler 02], Clerc et Kennedy ont démontré qu'une bonne convergence peut être obtenue en rendant dépendants ces paramètres. L'utilisation d'un *coefficient de constriction* χ (ou *facteur de constriction*) permet de mieux contrôler la divergence de l'essaim et de s'affranchir de la définition de V_{max} [Cler 02]. Cette variante de PSO, qui a été largement utilisée dans la littérature, est connue sous le nom de *canonical PSO*. En utilisant le coefficient de constriction, l'équation (1.3.1) devient :

$$v_{ij}(t+1) = \chi \left(v_{i,j}(t) + \phi_1 \, r_1 \left(pbest_{i,j}(t) - x_{i,j}(t) \right) + \phi_2 \, r_2 \left(gbest_j(t) - x_{i,j}(t) \right) \right) \qquad (1.3.5)$$

avec :

$$\chi = \frac{2}{\phi - 2 + \sqrt{\phi^2 - 4\phi}} \qquad (1.3.6)$$

où : $\phi = \phi_1 + \phi_2$, $\phi > 4$.

Les valeurs optimales de ϕ_1 et ϕ_2 ont été déterminées dans [Cler 02], en effectuant de nombreux tests. En général, on utilise $\phi = 4{,}1$ et $\phi_1 = \phi_2$, ce qui donne un coefficient $\chi = 0{,}7298844$.

Dans [Eber 00], les auteurs ont indiqué que l'utilisation d'un coefficient de constriction donne généralement un meilleur taux de convergence, sans avoir à fixer de vitesse

maximale V_{max}. Cependant, dans certains cas, le coefficient de constriction seul ne permet pas la convergence vers la solution optimale pour un nombre d'itérations donné. Pour remédier à ce problème, il pourrait être intéressant de fixer $V_{max} = (x_{max} - x_{min})/2$ en plus du coefficient de constriction, ce qui, selon les études de Shi et Eberhart, permet d'améliorer les performances globales de l'algorithme. Ainsi, il est à noter que PSO, utilisant un coefficient de constriction, n'est pas la seule version de PSO qui garantisse la convergence vers un état d'équilibre. D'autres exemples peuvent être trouvés dans [Van 01, Trel 03, Peer 03, Zhen 03], qui permettent aussi de provoquer la convergence de l'algorithme.

1.3.3.3 Topologie de voisinage

Comme nous l'avons vu dans les deux sous-sections 1.3.1 et 1.3.2, PSO est une méthode d'optimisation stochastique inspirée d'un comportement social. Ce comportement a été modélisé par les deux équations (1.3.1) et (1.3.2) pour guider les particules durant le processus de déplacement. Le choix d'une topologie (le réseau de communication entre les particules) a donc une influence importante sur les performances de PSO.

A l'origine, dans la version de PSO résumée dans l'Algorithme 1.6, les auteurs ont défini une topologie entièrement connectée (i.e. chaque particule est reliée à toutes les autres). Cette version de PSO est appelée *version globale* (Gbest), car la particule est informée par la totalité des autres, et l'information effectivement utilisée est incarnée par le terme $\vec{G}best$ de la troisième composante de l'équation (1.3.1). Cette version a l'inconvénient majeur de ne pas donner lieu à une exploration suffisante, ce qui peut conduire à une stagnation dans un optimum local et donc à une convergence prématurée. De nombreuses variantes de la version originale, dites *versions locales* (Lbest), ont été proposées dans la littérature de PSO, afin d'améliorer sa convergence. Parmi ces variantes, nous pouvons citer celle proposée dans [Eberhart et al., 1995] et qui utilise un graphe d'information statique sous forme d'anneau (cette version est connue comme étant la version *locale classique*). Dans les versions locales, le terme $\vec{G}best$ est remplacé par les termes $\vec{L}best_i$, où, pour chaque particule i, on définit un ensemble de voisinage (i.e. l'information qui doit être partagée est la meilleure solution trouvée dans le voisinage de chaque particule ($\vec{L}best_i$)). Cette partie sera traitée et détaillée dans le chapitre 3, où nous présenterons aussi une nouvelle topologie dynamique, nommée *Dcluster*.

1.3.3.4 Coefficient d'inertie

Le coefficient d'inertie w, introduit par Shi et Eberhart [Shi 99], contrôle l'influence de la direction de la particule sur le déplacement futur. Le but de l'introduction de ce paramètre est de réaliser un équilibre entre la recherche locale (exploitation) et la recherche globale (exploration). L'intensité de l'exploration de l'espace de recherche dépend de la valeur du poids d'inertie, une grande valeur de w facilitant une exploration globale, alors qu'une petite valeur facilite l'exploration locale. Du fait de son influence sur les performances de l'algorithme PSO, le poids d'inertie a suscité un grand intérêt de la part de la communauté des chercheurs. Dans [Shi 99], les auteurs ont proposé un coefficient d'inertie dynamique qui varie au cours du temps. Il commence par une valeur proche de 0,9 et descend linéairement pour arriver à 0,4. Cette stratégie a beaucoup amélioré les performances de PSO pour plusieurs problèmes d'optimisation. Le coefficient d'inertie w varie linéairement avec le temps selon la formule suivante :

$$w = w_{min} + (w_{max} - w_{min}).(\frac{iter}{max_{iter}}) \tag{1.3.7}$$

où $iter$ est l'itération courante et max_{iter} est le nombre maximal d'itérations. w_{max} et w_{min} désignent respectivement les valeurs maximum et minimum du coefficient w (généralement, $w_{min}, w_{max} \in [0,1]$).

Dans [Chat 06], Chatterjee et Siarry ont utilisé une autre stratégie non-linéaire pour définir un coefficient d'inertie dynamique. Dans [Eber 01], Eberhart et Shi ont proposé une autre variante, dans laquelle le coefficient d'inertie est choisi au hasard, selon une distribution uniforme, dans l'intervalle $[0{,}5, 1]$. Cet intervalle a été inspirée du facteur de constriction proposé par Clerc et Kennedy (la valeur attendue du coefficient d'inertie, dans ce cas, est égale à $0{,}75 \approx 0{,}729$).

1.3.3.5 Stratégie FIPS

Kennedy et Mendes [Mend 04] ont proposé une nouvelle manière d'utiliser la topologie Gbest, appelée FIPS (*Fully Informed Particle Swarm*). FIPS utilise une partie des informations de chaque voisine, au lieu de se baser seulement sur les informations de la meilleure voisine et la meilleure expérience propre à la particule. Par conséquent, l'utilisation d'une topologie entièrement connectée ne signifie pas que l'information utilisée soit seulement

la meilleure solution trouvée par l'essaim. En effet, dans FIPS, elle est toujours utilisée, mais elle n'est pas la seule. Pour les algorithmes basés sur le principe de la topologie FIPS, l'information utilisée pour déplacer les particules est issue de toutes les autres particules. Ainsi, toutes les voisines contribuent à l'ajustement de la vitesse d'une particule :

$$v_i^{t+1} = \chi \left(v_i^t + \sum_{n=1}^{N_i} \frac{U(0,\phi)(p^t_{nbr(n)} - x_i^t)}{N_i} \right) \tag{1.3.8}$$

où N_i est le nombre de voisins de la particule i, $nbr(n)$ est la $n^{\text{ième}}$ particule voisine de la particule i et ϕ est la constante d'accélération, qui permet de contrôler la convergence des particules. Cette dernière est fixée à 4,1, suite à l'analyse de Clerc et al [Cler 02].

1.3.3.6 Algorithme TRIBES

TRIBES est un algorithme d'optimisation par essaim particulaire sans paramètres de contrôle, qui a été proposé par Clerc [Cler 03]. Cet algorithme présente la particularité d'être totalement adaptatif, c'est-à-dire que tous les paramètres de contrôle sont calculés de manière autonome par l'algorithme. En effet, TRIBES est défini comme une boîte noire, pour laquelle l'utilisateur n'a plus aucun paramètre à régler. Il doit seulement définir le problème à résoudre (i.e. la fonction objectif, l'espace de recherche, les contraintes), ainsi que son critère d'arrêt. Cependant, il est à signaler que TRIBES ne peut pas résoudre tous les problèmes. De plus, ses résultats sont probabilistes à cause de son caractère stochastique. Le but de TRIBES, d'après son auteur, est d'être efficace dans la plupart des cas et de permettre à ses utilisateurs de gagner du temps, en évitant l'étape de réglage de la métaheuristique.

Dans TRIBES, l'essaim particulaire est divisé en plusieurs sous-essaims appelés « tribus ». Les tribus sont de tailles différentes, qui évoluent au cours du temps. Le but est d'explorer simultanément plusieurs régions de l'espace de recherche, généralement des optima locaux, avant de prendre une décision globale. Dans le but de prendre un telle décision, les tribus échangent leurs résultats tout au long du traitement. Deux types de communications sont donc à définir : la communication *intra-tribu* et la communication *inter-tribus*. Chaque tribu est composée d'un nombre variable de particules. En effet, une tribu qui peine à améliorer ses résultats génère des particules plus « exploratrices ». Les particules générées par les différentes tribus forment une nouvelle tribu, qui reste en communication avec ses génitrices. Inversement, une tribu efficace tendra à supprimer celles

de ses particules qui n'ont pas contribué à sa bonne performance.

TRIBES est un algorithme compétitif, qui permet de trouver rapidement des optima locaux (très utile pour l'optimisation dynamique). Cependant, les particules ont tendance à rester dans ces optima locaux et ont du mal à en sortir.

La thèse de Yann Cooren [Coor 08] s'attache particulièrement à l'algorithme TRIBES, qui est en détail. C'est ainsi que Cooren a proposé deux idées qui permettent d'améliorer les performances de cet algorithme :

- La première idée consiste à utiliser un nouveau mode d'initialisation (initialisation régulière) pour assurer une couverture plus uniforme de l'espace de recherche par les particules. En pratique, les particules sont initialisées de manière à être les plus éloignées possible les unes des autres et les plus éloignées possible des frontières de l'espace de recherche.
- La deuxième idée consiste à utiliser une nouvelle stratégie de déplacement, basée sur une hybridation avec un algorithme à estimation de distribution, pour maintenir la diversité au sein de l'essaim, tout au long du traitement.

Les résultats obtenus montrent une réelle amélioration apportée à l'algorithme initial. Dans sa thèse, Yann Cooren a proposé aussi une version performante pour les problèmes multi-objectifs, dénommée MO-TRIBES.

1.3.3.7 PSO et hybridation

Ces dernières années, l'hybridation des algorithmes a attiré l'attention de nombreux chercheurs afin d'améliorer leurs performances. L'objectif de l'hybridation est de combiner les caractéristiques de plusieurs algorithmes pour tirer profit de leurs avantages [Talb 02]. Mais l'algorithme résultant risque d'hériter également de leurs faiblesses. De plus, un algorithme résultant de l'hybridation de plusieurs algorithmes peut avoir une complexité importante. Comme pour toutes les métaheuristiques, l'hybridation a aussi touché le domaine de PSO dans le but d'améliorer ses performances. Dans ce qui suit, nous présentons quelques exemples d'hybridations entre PSO et d'autres algorithmes.

Dans [Ange 98], Angeline a proposé la première hybridation d'un algorithme PSO. Il introduit un processus de sélection et un processus de mutation inspirés des algorithmes évolutionnaires. Le processus de sélection est utilisé pour choisir des « bonnes » particules qui vont subir une mutation, et des « mauvaises » particules qui sont éliminées.

Dans [Zhan 09], une approche originale d'hybridation, qui est une combinaison de PSO et DE (DE-PSO), est proposée. Cette approche consiste à définir une stratégie de déplacement aléatoire pour accroître la capacité d'exploration et en même temps accélérer la convergence de l'algorithme, en utilisant des opérateurs de l'algorithme DE. Dans cette approche, trois stratégies de mise à jour de la particule ont été utilisées : *DE Updating Strategy* (DEUS), *Random Updating Strategy* (RUS) et *PSO Updating Strategy* (PSOUS). Dans [Mira 02], une hybridation entre PSO et les stratégies évolutionnaires est proposée. Les paramètres *chi*, ϕ_1 et ϕ_2, ainsi que $\vec{g}$, sont perturbés selon une distribution gaussienne. La variance de cette distribution est déterminée à l'aide d'un processus de sélection. Dans [Robi 02], une hybridation entre PSO et les algorithmes génétiques est développée. Il est démontré dans cet article que PSO est favorable dans la phase de diversification, alors que les algorithmes génétiques sont plus efficaces dans la phase d'intensification. Dans [Shel 07], un algorithme de PSO est hybridé avec un algorithme de colonies de fourmis. L'idée sous-jacente consiste à utiliser PSO comme méthode de recherche globale, alors que l'algorithme de colonies de fourmis est censé améliorer le processus d'intensification, en étant utilisé comme une recherche locale. Dans [Iqba 06], un algorithme de PSO utilisant des principes des algorithmes à estimation de distribution est présenté. Les meilleures particules sont ici utilisées pour attirer les autres particules de l'essaim à l'aide de l'estimation d'une distribution de probabilité. Enfin, il existe aussi plusieurs méthodes hybridant PSO avec une recherche locale. Parmi ces méthodes, nous citons NM-PSO, qui est une combinaison de la technique de *Nelder-Mead* (NM) et de PSO. Dans NM-PSO, le processus hybride réalise l'exploration par l'algorithme PSO et l'exploitation par l'algorithme NM [Fan 04].

1.3.3.8 PSO coopérative

Les algorithmes de recherche coopératifs, notamment ceux de PSO, ont été largement étudiés ces dernières années, surtout pour résoudre les problèmes d'optimisation de grande tailles. L'approche de base consiste à avoir plus d'un module de recherche en cours d'exécution (peut-être à travers des sous-essaims) et un échange d'informations entre eux, afin d'explorer plus efficacement l'espace de recherche et d'atteindre de meilleures solutions.

Dans [Berg 04], les auteurs proposent deux variantes de PSO : *Cooperative PSO* (CPSO-S_k) et *Hybrid CPSO* (CPSO-H). La première variante CPSO-S divise l'espace n-dimensionnel (vecteur de solutions de dimension n) en sous-espaces (de plus petits vec-

teurs), où chaque sous-espace est optimisé par un essaim séparé. Le vecteur global de solutions est construit à partir des solutions trouvées par la meilleure particule de chaque essaim. Dans le cas particulier où $n = k$, l'algorithme est appelé CPSO-S et utilise n essaims 1-dimensionnel pour effectuer la recherche dans chaque dimension séparément. La deuxième variante CPSO-H consiste en deux phases de recherche séquentielle. Chaque phase s'exécute seulement pour une itération et communique la meilleure solution trouvée à la phase suivante. La première phase utilise un CPSO-S (avec des essaims 1-dimensionnel) et la seconde phase utilise l'algorithme de base de PSO.

Dans [Bask 04], l'algorithme *Concurrent PSO* (CONPSO) considère deux essaims en concurrence qui sont générés à l'aide de deux variantes de PSO : PSO de base et *Fitness-to-Distance Ratio PSO* (FDR-PSO) [Pera 03]. Ces deux essaims effectuent la recherche en même temps avec un échange fréquent d'informations. Les informations échangées sont les meilleures solutions globales $gbest_1$ et $gbest_2$ de deux essaims. Après chaque point d'échange, les particules des essaims sont obligées de suivre la meilleure solution globale trouvée entre les deux.

Dans [Yen 08], les auteurs présentent une autre méthode d'échange d'informations entre les essaims, dans laquelle la population des particules est divisée en m sous-essaims qui coopèrent entre eux. Au moment de l'échange, chaque sous-essaim prépare deux ensembles de particules : les particules à envoyer à un autre sous-essaim et les particules à remplacer par d'autres particules venant d'autres sous-essaims. L'échange d'informations se fait entre sous-essaims du même voisinage, déterminé par une distance qui change au cours de l'exécution de l'algorithme. Ceci fournit une configuration dynamique entre les sous-essaims, grâce au changement de distance, qui permet à chacun d'entre eux de communiquer avec d'autres groupes.

Un autre algorithme de PSO coopérative a été proposé dans [Niu 07], basé sur un modèle maître-esclave. Dans cet algorithme, la population se compose d'un essaim « maître » et plusieurs essaims « esclaves ». Les essaims esclaves exécutent un seul PSO indépendamment pour maintenir la diversité des particules, tandis que l'essaim maître évolue en fonction de son information propre et de celle des essaims esclaves.

1.3.3.9 Autres variantes de PSO

Dans [Hsie 09], Hsieh et al. proposent un algorithme (EPU-PSO) basé sur une population de taille variable qui utilise trois idées principales pour améliorer les performances

de PSO. La première idée introduit un gestionnaire de la population où, selon l'état de la recherche, l'algorithme ajoute ou supprime des particules. Cette dynamicité affecte considérablement les performances et augmente la capacité à trouver l'optimum global. La deuxième idée est basée sur la stratégie dite *solution-sharing strategy*, qui permet aux meilleures particules de partager leurs informations et de mettre à jour leurs vitesses. La troisième idée porte sur la technique dite *searching range sharing* (SRS), qui empêche les particules de tomber dans un optimum local. En outre, SRS étend l'exploration de l'espace de recherche à des zones inexplorées. CLPSO [Lian 06] est une variante de PSO qui utilise une nouvelle stratégie d'apprentissage (*learning strategy*) pour mettre à jour les vitesses des particules utilisant des informations historiques. UPSO [Pars 04] est un algorithme de PSO qui regroupe les variantes globale et locale de PSO, en combinant leurs capacités d'exploration et d'exploitation.

1.4 Validation des algorithmes

Dans cette section, nous allons présenter les fonctions et les problèmes réels utilisés pour comparer les résultats obtenus par les algorithmes que nous proposons dans ce manuscrit à ceux d'autres algorithmes existants dans la littérature. Nous nous appuyons, pour comparer les algorithmes, sur la procédure de test définie dans le cadre de la conférence 2005 *IEEE Congress on Evolutionary Computation* (CEC'05) [Suga 05].

Le critère d'arrêt peut être différent suivant le problème posé. Si l'optimum global est connu a priori, on peut définir une « erreur acceptable » comme critère d'arrêt. Sinon, il est habituel de fixer un nombre maximum d'évaluations de la fonction objectif ou un nombre maximum d'itérations comme critère d'arrêt. Cependant, selon le problème posé et les exigences de l'utilisateur, d'autres critères d'arrêt peuvent être utilisés.

1.4.1 Principales fonctions de test

Les principales fonctions de test sont classées en quatre groupes (voir Tableau 1.1) en fonction de leurs propriétés [Suga 05, Tu 04, Yao 99].

Dans l'annexe, nous détaillons ces quatre groupes.

Groupe	Type de fonction	Nom de fonction
A	Fonctions unimodales	f_1. *Sphere function* f_2. *Quadric function*
B	Fonctions multimodales	f_3. *Rosenbrock's function* f_4. *Ackley's function* f_5. *Rastrigin's function* f_6. *Weierstrass's function* f_7. *Generalized penalized function* f_8. *Griewank's function* f_9. *Tripod's function*
C	Fonctions pivotées	f_{10}. *Rotated Quadric function* f_{11}. *Rotated Rosenbrock's function* f_{12}. *Rotated Ackley's function* f_{13}. *Rotated Rastrigin's function* f_{14}. *Rotated Weierstrass's function* f_{15}. *Rotated Generalized penalized function*
D	Fonctions décalées	f_{16}. *Shifted Rosenbrock's function* f_{17}. *Shifted Ackley's function* f_{18}. *Shifted Rastrigin's function* f_{19}. *Shifted Griewank's function* f_{20}. *Shifted Sphere function*

TABLEAU 1.1: Les principales fonctions de test.

1.4.2 Problèmes réels

Quelques problèmes réels très fréquents dans le domaine d'optimisation et utilisés pour mesurer les performances des algorithmes, sont présentés dans le tableau 1.2 et détaillés dans l'annexe.

1.5 Conclusion

Dans ce chapitre, nous avons présenté un état de l'art sur les méthodes d'optimisation mono-objectifs statiques. Tout d'abord, nous avons donné la définition d'un problème d'optimisation. Puis, nous avons distingué les deux types de problèmes d'optimisation mono-objectif difficile : les problèmes à variables discrètes et les problèmes à variables continues. Ensuite, nous avons présenté une famille de méthodes de résolution de problèmes à variables continues : les métaheuristiques. Enfin, nous avons décrit la plupart des fonctions d'un jeu de test très utilisé dans la littérature de l'optimisation continue,

F	Problème
F_1	*Lennard-Jones atomic cluster*
F_2	*Gear train design*
F_3	*Frequency modulation sound parameter identification*
F_4	*Spread spectrum radar poly-phase code design*
F_5	*Compression spring design*
F_6	*Perm function*
F_7	*Mobile network design*
F_8	*Pressure Vessel design*
F_9	*Welded Beam design*
F_{10}	*Speed Reducer design*
F_{11}	*Problème de contraintes 1*
F_{12}	*Problème de contraintes 2*

TABLEAU 1.2: Les problèmes réels.

défini dans le cadre de la conférence (CEC'05) et quelques problèmes réels très fréquents dans ce domaine.

Parmi les métaheuristiques présentées pour résoudre ces problèmes, un intérêt particulier a été porté à la méthode d'optimisation par essaim particulaire (PSO). Cette méthode, qui est inspirée du monde du vivant, a rencontré un succès remarquable depuis sa création, grâce à sa simplicité. Elle présente l'avantage d'être efficace sur une vaste gamme de problèmes, sans pour autant que l'utilisateur ait à modifier la structure de base de l'algorithme. Cependant, PSO présente un problème majeur, qui rebute encore certains utilisateurs : le problème de la convergence prématurée, qui peut conduire les algorithmes de ce type à stagner dans un optimum local. De nombreux travaux ont été proposés pour améliorer les performances de PSO, tout en essayant de remédier à ce problème. Parmi ceux-ci, nous avons décrit les travaux basés sur l'hybridation avec d'autres méthodes, l'utilisation de plusieurs essaims, le mode d'initialisation des particules, la topologie de voisinage, etc.

Nous nous sommes inspirés de certains de ces travaux et d'idées novatrices pour concevoir, durant cette thèse, plusieurs variantes de l'algorithme d'optimisation par essaim particulaire. Ces variantes sont présentées dans les chapitres 2 et 3 et analysées en utilisant les fonctions de test et les problèmes réels donnés dans le présent chapitre et détaillés en annexe.

CHAPITRE DEUX

ÉLABORATION D'UN ALGORITHME D'OPTIMISATION PAR ESSAIM PARTICULAIRE PSO-2S

2.1 Introduction

L'algorithme d'optimisation par essaim particulaire (PSO) est caractérisé par sa convergence rapide, ce qui peut conduire les algorithmes de ce type à stagner dans un optimum local [Shi 99]. Pour remédier à ce problème et améliorer les performances de PSO standard, plusieurs tentatives ont été faites, telles que les modèles hybrides, les mécanismes inspirés de la biologie et quelques modifications de base dans les équations de mise à jour de la vitesse et de la position (le poids d'inertie w, la topologie, etc.) [Das 08].

Toujours dans le cadre de l'amélioration de l'algorithme PSO, nous proposons dans ce chapitre deux variantes pour remédier au problème de la convergence prématurée de cet algorithme. Dans la première variante, nous proposons un algorithme, nommé *PSO-2S*, qui est basé sur l'utilisation de l'approche multi-essaims. Dans la deuxième variante, nous proposons un algorithme, nommé *DEPSO-2S*, qui est une variante de *PSO-2S* basée sur l'hybridation de deux algorithmes d'optimisation, PSO et DE.

PSO-2S [El D 12a] se base sur trois idées principales. La première idée consiste à utiliser deux types d'essaims : un essaim principal, noté $S1$, et S essaims auxiliaires, notés $S2_i$, où $1 \leq i \leq S$. La deuxième idée consiste à partitionner l'espace de recherche en plusieurs zones où les essaims auxiliaires seraient initialisés (le nombre de zones est égal au nombre d'essaims auxiliaires, est donc égal à S). La dernière idée consiste à utiliser le concept de la répulsion électrostatique, qui permet de couvrir plus largement chaque zone de l'espace de recherche. Ces trois idées sont appliquées dans la première phase de l'algorithme *PSO-2S*, qui est la phase de construction de l'essaim principal.

DEPSO-2S [El D 12b] est une variante de *PSO-2S*. Elle utilise les trois idées principales sur lesquelles se base *PSO-2S* et qui sont présentées dans le paragraphe précédent. Par

contre, dans *DEPSO-2S*, l'algorithme DE est utilisé pour construire l'essaim principal au lieu de l'algorithme PSO. Pour ce faire, nous proposons une nouvelle version de DE afin de l'adapter à *PSO-2S*. Cette version est utilisée uniquement dans la première phase de l'algorithme et son rôle prend fin après la construction de l'essaim principal. Une fois, l'essaim principal construit, l'algorithme continue la recherche de la même façon que *PSO-2S*.

Dans la section 2.2, nous commençons par décrire l'algorithme *PSO-2S*. Ensuite, nous présentons une analyse expérimentale de cet algorithme et nous comparons ses performances à celles des autres algorithmes de PSO proposés dans la littérature. Dans la section 2.3, nous commençons par décrire l'algorithme *DEPSO-2S*. Ensuite, nous présentons des résultats numériques obtenus par *DEPSO-2S* et d'autres algorithmes de PSO et l'algorithme DE, et nous terminons en établissant une comparaison entre les performances des différents algorithmes.

2.2 Description de l'algorithme PSO-2S

Dans cette section, nous proposons un nouvel algorithme d'optimisation par essaim particulaire, nommé *PSO-2S*, pour résoudre des problèmes d'optimisation continue. Afin de bien comprendre le fonctionnement de cet algorithme, nous allons présenter les principales techniques utilisées, qui ont abouti à élaborer la structure finale de *PSO-2S*, en plusieurs paragraphes. Nous commencerons par décrire la structure générale de *PSO-2S*, puis nous verrons ensuite chacune des stratégies mises en œuvre.

2.2.1 Structure générale

PSO-2S est une nouvelle variante de l'algorithme d'optimisation par essaim particulaire (PSO). Cette variante utilise plusieurs sous-essaims au lieu d'un seul essaim. L'approche générale de l'algorithme multi-essaims consiste à diversifier la recherche afin d'explorer plusieurs zones de l'espace de recherche. En effet, dans cette variante, nous utilisons les sous-essaims dans différentes zones, où chaque sous-essaim est utilisé pour explorer une zone particulière. Les algorithmes multi-essaims sont adoptés spécialement pour l'optimisation de problèmes multimodaux ayant plusieurs optima locaux.

PSO-2S utilise des essaims auxiliaires et un essaim principal. Les essaims auxiliaires sont utilisés pour construire l'essaim principal. Les essaims auxiliaires sont des sous-

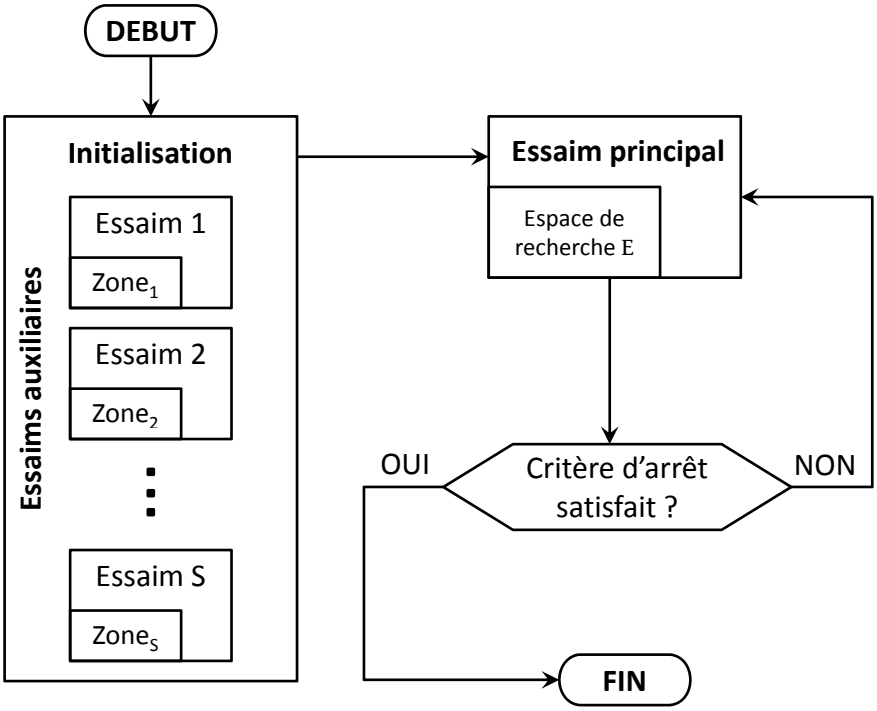


FIGURE 2.1: Schéma fonctionnel de PSO-2S.

essaims qui possèdent des rôles limités. Un essaim auxiliaire a pour but de découvrir une zone de l'espace de recherche en effectuant un certain nombre d'itérations, et son rôle se termine après la réception de l'information sur la meilleure position de la zone. Plus précisément, les populations des essaims auxiliaires sont initialisées aléatoirement dans différentes zones, selon un partitionnement déjà fait de l'espace de recherche. Puis une force de répulsion électrostatique est appliquée séparément sur les particules de chaque essaim pour les diversifier dans sa zone correspondante. Ensuite, les particules de chaque essaim effectuent $nb_{generation}$ générations (valeur prédéfinie) dans leur zone et, à la fin de ces générations, la meilleure position trouvée est sauvegardée dans un tableau. Les meilleures positions obtenues par tous les essaims auxiliaires sont donc regroupées dans un tableau de taille égale au nombre des essaims initialisés et aussi au nombre de zones. Le tableau final de toutes les meilleures positions regroupées composera ensuite la population initiale de l'essaim principal. Une fois l'initialisation de l'essaim principal est achevée, les essaims auxiliaires ne sont plus utilisés. L'essaim principal prend alors le relais et continue la recherche de la même façon que l'algorithme PSO standard jusqu'à atteindre un critère d'arrêt. La structure générale de *PSO-2S* est illustrée dans la figure 2.1.

2.2.2 Utilisation de deux essaims

L'algorithme PSO est connu pour sa convergence rapide, qui peut conduire à une stagnation dans un optimum local [Shi 99]. Nous profitons de cette convergence rapide pour détecter plusieurs optima locaux. La détection des optima locaux consiste à effectuer plusieurs recherches sur différents sites à partir de plusieurs essaims. Les optima détectés sont utilisés pour construire un essaim principal.

Comme nous l'avons indiqué dans la section précédente, *PSO-2S* consiste à utiliser des essaims auxiliaires ($S2_i$) pour construire son essaim principal ($S1$). Les essaims auxiliaires sont situés dans plusieurs zones de l'espace de recherche et ils sont de tailles différentes qui correspondent aux tailles des zones prédéfinies par la partition de l'espace de recherche. Nous en déduisons que la taille de l'essaim dépend de la taille de sa zone (zone i correspondant à l'essaim $S2_i$), où la plus petite zone de l'espace de recherche contient l'essaim contenant le moins de particules. Les essaims auxiliaires n'évoluent pas au cours du temps et leurs tailles restent donc fixes. Le but est d'explorer simultanément plusieurs régions de l'espace de recherche, généralement des optima locaux, avant que l'essaim principal ne prenne une décision globale. Les essaims auxiliaires peuvent effectuer leurs travaux de manière synchrone ou asynchrone, c'est-à-dire que les essaims peuvent travailler en parallèle (i.e. les essaims travaillent en même temps) ou en séquence (i.e. un essaim finit avant qu'un autre ne commence). Dans les deux cas, les essaims auxiliaires effectuent leurs recherches sans communiquer entre eux, c'est-à-dire que le travail de chaque essaim auxiliaire est indépendant de celui des autres essaims.

Dans *PSO-2S*, les particules de l'essaim auxiliaire sont initialisées aléatoirement dans la zone qui lui correspond. Après l'initialisation, chaque essaim exécute $nb_{generation}$ itérations de la même façon que PSO standard, et il procure à la fin de ces itérations la meilleure position trouvée pendant la recherche. Le but, dans la génération d'un petit nombre d'itérations est d'explorer uniquement la région où l'essaim a été initialisé, et donc le plus grand nombre d'itérations est laissé pour l'essaim principal. Du fait de la convergence rapide de l'algorithme PSO et de l'utilisation de l'approche multi-essaims dans un espace partitionné, les essaims auxiliaires dans *PSO-2S* convergent chacun vers une position intéressante dans les différentes zones de l'espace, et donc le cas où plusieurs essaims convergent vers la même position est évité. Ces positions sont des optima locaux des problèmes obtenus des différentes zones. Donc, à chaque fois que l'essaim auxiliaire termine son travail, la meilleure solution trouvée par cet essaim est ajoutée, comme une particule initiale, à l'es-

saim principal $S1$. La taille de population de l'essaim principal $S1$ est donc égale au nombre des essaims auxiliaires $S2_i$. La phase de construction de l'essaim $S1$ se termine après avoir généré tous les essaims $S2_i$. Après cette phase, $S1$ prend le relais et continue seul à faire une recherche globale dans l'espace. A la différence de PSO standard, la deuxième phase de *PSO-2S* (phase de l'essaim principal) commence la recherche avec des particules bien positionnées dans l'espace (c'est-à-dire des positions intéressantes dans les différentes régions). En effet, $S1$ effectue les itérations restantes (nombre d'itérations restantes = nombre total d'itérations - nombre d'itérations réservées aux essaims auxiliaires) et s'arrête lorsqu'on atteint un critère d'arrêt. La phase de construction de l'essaim principal $S1$ est illustrée dans la figure 2.2.

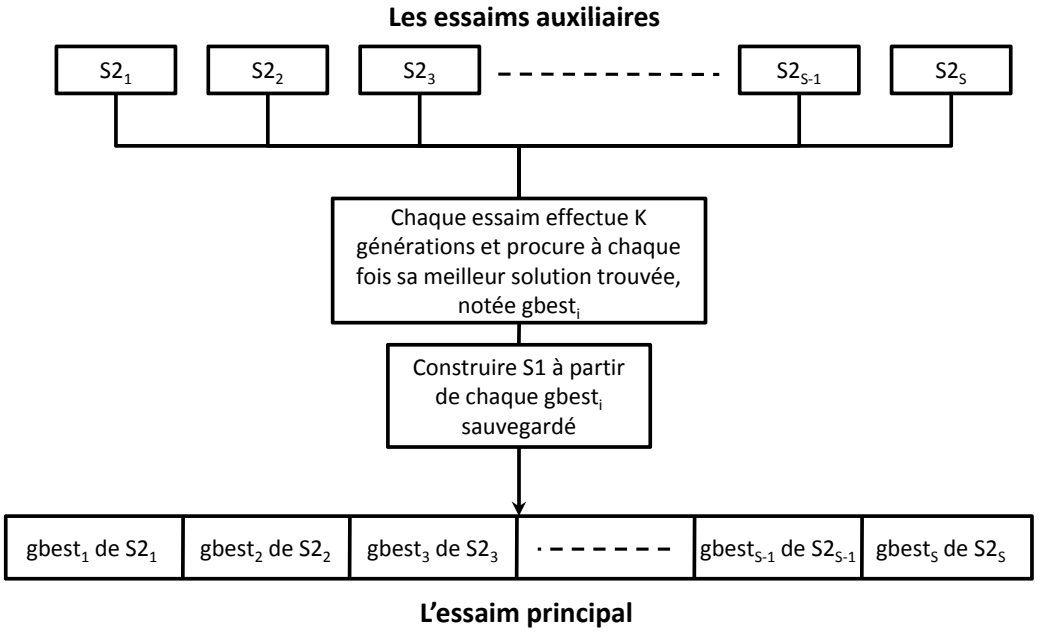


FIGURE 2.2: Construction de l'essaim principal à partir des essaims auxiliaires.

2.2.3 Partitionnement de l'espace de recherche

Dans la vie quotidienne, le partage des tâches aboutit dans la plupart des cas à des résultats intéressants. Par exemple, la recherche des trésors perdus ou le forage des puits de pétrole, où l'on commence la recherche en l'effectuant sur plusieurs régions prédéfinies, pour ensuite intensifier la recherche dans les zones prometteuses. Nous nous inspirons de ce modèle de recherche pour trouver une solution optimale du problème d'optimisation et, à cet effet, nous proposons de partitionner l'espace de recherche en plusieurs zones.

La recherche de *PSO-2S* est faite alors dans un espace de recherche partitionné et donc nous partitionnons l'espace de recherche E en plusieurs zones séparées, dans lesquelles les essaims auxiliaires sont initialisés. Ce partitionnement contribue à augmenter la diversification des particules dans tout l'espace de recherche, et d'assurer une bonne couverture pendant la phase d'initialisation. En pratique, nous fixons un nombre maximal de zones, dénommé max_{zone}, qui est égal au nombre des essaims auxiliaires S. Les zones qui forment l'espace de recherche E sont définies par :

$$zone_i, 1 \leq i \leq S \text{ tels que } \bigcup_{i=1}^{S} zone_i = E; \ \ (\forall\, i,j \in \{1,\ldots,S\},\ zone_i \cap zone_j = \emptyset)$$

Afin de partitionner un espace de recherche E de dimension D ($E = [min_d, max_d]^D$) en max_{zone} zones, où min_d et max_d représentent respectivement les bornes inférieure et supérieure dans chaque dimension d, $1 \leq d \leq D$, nous définissons le centre de l'espace de recherche comme un point de départ pour le partitionnement. Les centres des intervalles, qui définissent les bornes dans chaque dimension d, sont définis par l'équation suivante :

$$centre_d = \frac{(max_d + min_d)}{2} \tag{2.2.1}$$

Une fois que les centres d'intervalles sont définis, nous commençons par partitionner les intervalles de chaque dimension, et donc l'espace de recherche, en se basant sur ces centres. A cet effet, nous devons calculer un pas pas_d pour chaque intervalle $[min_d, max_d]$. Les pas_d sont utilisés pour construire les nouvelles bornes $[zone_{imin}, zone_{imax}]$ de chaque zone i, $1 \leq i \leq max_{zone}$. Le pas_d de chaque dimension est calculé selon la formule suivante :

$$pas_d = \frac{max_d - min_d}{2 * max_{zone}} \tag{2.2.2}$$

Chaque zone $[zone_{idmin}, zone_{idmax}]^D$ est déterminée en utilisant les deux équations suivantes :

$$zone_{idmin} = centre_d - i * pas_d \tag{2.2.3}$$

$$zone_{idmax} = centre_d + i * pas_d \tag{2.2.4}$$

où $zone_{idmin}$ et $zone_{idmax}$ sont, respectivement, les bornes inférieures et les bornes supérieures de chaque zone. La figure 2.3 et la figure 2.4 illustrent le partitionnement de l'espace de recherche dans les cas uniforme et non uniforme.

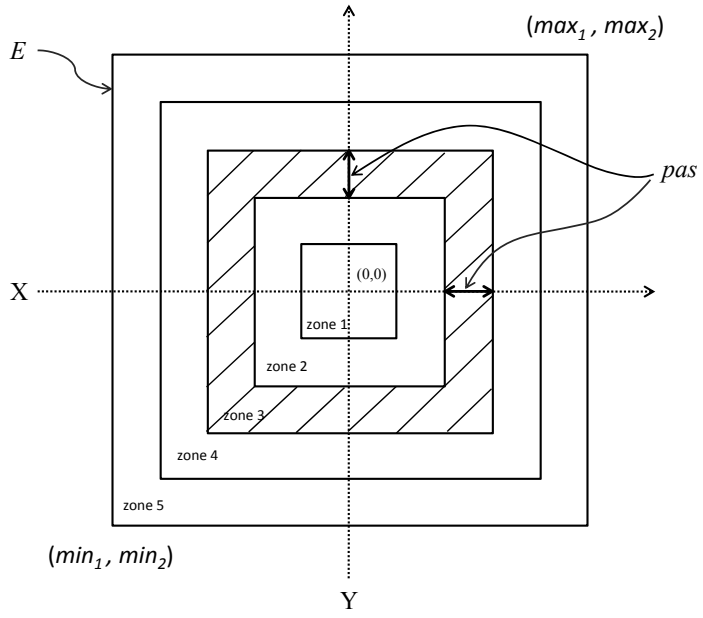


FIGURE 2.3: Partitionnement d'un espace de recherche uniforme.

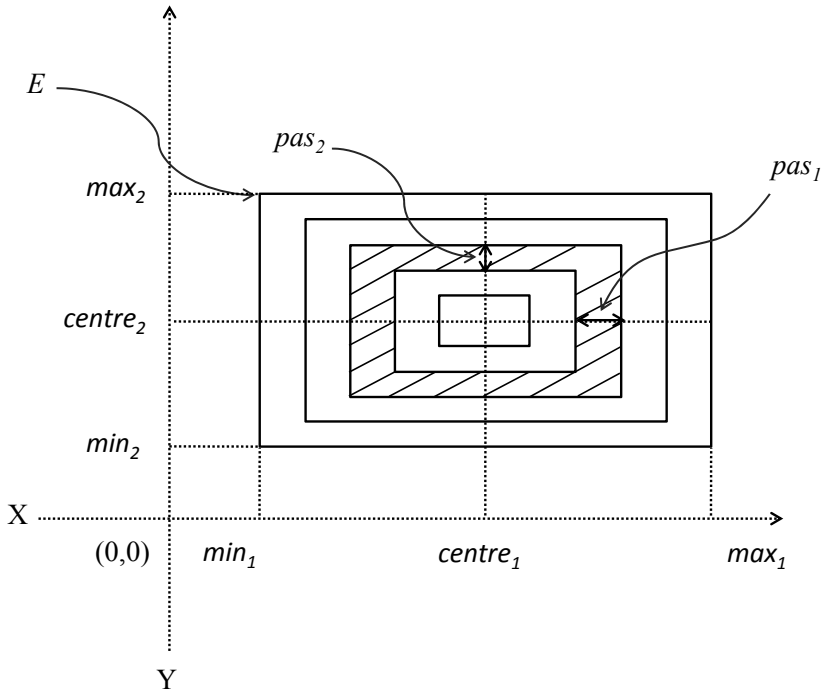


FIGURE 2.4: Partitionnement d'un espace de recherche non uniforme.

Comme nous pouvons le voir sur les figures 2.3 et 2.4, les zones partitionnées de l'espace de recherche E sont de tailles différentes ($zone_1 < zone_2 < zone_3 < \cdots < zone_S$). C'est pourquoi, nous prenons en considération leurs tailles pour définir les tailles des populations des essaims auxiliaires à initialiser dans les différentes zones. En effet, nous commençons l'initialisation par la plus petite zone, qui prend un minimum de particules, pour augmenter ensuite ce nombre en fonction de la taille de chaque zone. La taille de population de chaque essaim auxiliaire $S2_i$ est calculée comme suit :

$$S2_{isize} = num_{zone} * nb_{particule} \tag{2.2.5}$$

où, $num_{zone} \in \{1, 2, \ldots, max_{zone}\}$ est le numéro de la zone courante et $nb_{particule}$ est un nombre fixe.

Pour éviter un nombre excessif de particules lorsque l'on utilise un grand nombre de zones, le choix d'une petite valeur pour le paramètre $nb_{particule}$ sera indispensable. Donc, la valeur de $nb_{particule}$ dépend de la valeur de max_{zone}.

Le pseudo-code d'initialisation des essaims auxiliaires est présenté dans la procédure 2.1.

2.2.4 Répulsion électrostatique des particules

Dans ce paragraphe, nous allons présenter l'heuristique de répulsion électrostatique utilisée dans l'algorithme *PSO-2S*. L'utilisation de cette heuristique vise à couvrir l'espace de recherche plus largement que d'habitude, et ce, en essayant de diversifier les particules dans les différentes zones pendant la phase d'initialisation.

A l'initialisation, chaque essaim auxiliaire se compose de particules positionnées aléatoirement et qui risquent donc de l'être de manière irrégulière dans la zone concernée. Certaines particules sont alors proches les unes des autres et d'autres sont isolées. Ceci est dû à l'utilisation d'un générateur aléatoire, qui conduit parfois à un déséquilibre dans la distribution des particules. Pour remédier à ce problème, nous proposons d'utiliser une heuristique de répulsion électrostatique issue de [Conw 98]. Cette heuristique considère un ensemble de points (particules) de l'espace comme des particules chargées se repoussant mutuellement. A partir d'un ensemble arbitraire de points, chaque point est repoussé itérativement selon une force en $1/r^2$, où r représente la distance entre deux points. L'heuristique est exécutée jusqu'à la satisfaction d'un critère d'arrêt. Cette technique a été utilisée

Initialisation PSO-2S

Entrées : max, min, max_{zone}
$centre \leftarrow (max + min)/2,\ pas \leftarrow (max - min)/2 * max_{zone}$
si (p = 1) **alors**
  pour $s = 1$ à $S2_p.size$ **faire**
    $S2_p.X[s] \leftarrow alea\_X([centre - pas, centre + pas]^D)$
    $S2_p.V[s] \leftarrow alea\_V([centre - pas, centre + pas]^D)$
  fin
  sinon
    pour $s = 1$ à $S2_p.size$ **faire**
      répéter
        $S2_p.X[s] \leftarrow alea\_X([centre - pas * p,\ centre + pas * p]^D)$
      tant que (isInside($S2_p.X[s]$, $p - 1$))
    fin
    pour $s = 1$ to $S2_p.size$ **faire**
      répéter
        $S2_p.V[s] \leftarrow alea\_V([centre - pas * p,\ centre + pas * p]^D)$
      tant que (isInside($S2_p.V[s]$, $p - 1$))
    fin
    retourner $S2_p$
  fin
fin

PROCÉDURE 2.1: La procédure d'initialisation des essaims auxiliaires de PSO-2S.

dans un algorithme d'optimisation dynamique à base d'agents, nommé MADO [Lepa 10]. Dans MADO, l' heuristique est utilisée pour positionner les agents initiaux dans l'espace de recherche. Cependant, dans *PSO-2S*, elle est utilisée pour repositionner les particules des essaims auxiliaires dans chaque zone.

Dans un premier temps, nous initialisons toutes les particules de manière aléatoire dans les différentes zones, en considérant chaque particule comme un électron chargé. Puis une force de répulsion en $1/r^2$, où r est la distance entre deux particules, est appliquée sur les particules de chaque zone, jusqu'à ce que le déplacement maximal d'une particule au cours d'une itération, désigné par max_{disp}, devienne inférieur à un seuil donné ϵ. Dans un espace de recherche normalisé $[0, 1]^D$, où D est la dimension de l'espace de recherche, ce seuil est fixé à 10^{-4}. Cette valeur associée à ϵ est utilisée plusieurs fois dans la littérature, en particulier dans [Lepa 10].

Le pseudo-code de la répulsion des particules est présenté dans la procédure 2.2. A chaque itération de l'heuristique, la force de répulsion $\vec{F}$ s'appliquant à la particule $\vec{P}_i$, sous l'effet des autres particules de l'essaim auxiliaire, est calculée à la ligne 7 de ce pseudo-code. Ensuite, le coefficient *step* est adapté (augmente ou diminue) de façon à maximiser le déplacement ($step.\vec{F}$) de la particule $\vec{P}_i$, sous la contrainte d'absence d'accroissement de l'énergie e du système. Dans cet algorithme, la fonction *crop* est utilisée à la ligne 10 pour centrer les particules repoussées de la zone courante de l'espace de recherche. Elle projette chaque particule $\vec{P}_i$ sur le point le plus proche du centre de la zone, comme l'illustre la figure 2.5. On répète les opérations de répulsion et de projection (*crop*) jusqu'à ce qu'aucune particule ne se déplace d'une distance supérieure à ϵ.

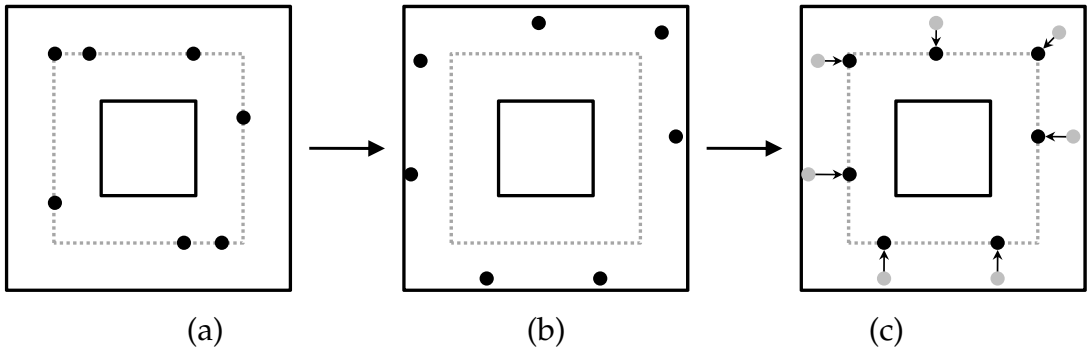


FIGURE 2.5: La fonction *crop* : (a) l'essaim auxiliaire à l'étape $n^{\text{ième}}$ de l'heuristique de répulsion après l'application de la fonction *crop*, (b) l'essaim auxiliaire à l'étape $(n+1)^{\text{ième}}$ après la répulsion, (c) l'essaim auxiliaire à l'étape $(n+1)^{\text{ième}}$ après l'application de la fonction *crop* de nouveau.

Le pseudo-code principal de l'algorithme *PSO-2S* est présenté dans la procédure 2.3. Ainsi, un exemple illustré dans la figure 2.6 montre les positions des particules avant et après l'utilisation de l'heuristique de répulsion dans la zone 3 (cas de dimension 2). En se basant sur cette figure, nous pouvons imaginer la forme initiale de la population d'un essaim auxiliaire dans *PSO-2S*. Les positions des particules obtenues après l'utilisation de l'heuristique de répulsion forment la population initiale de l'essaim auxiliaire, qui à son tour effectue $nb_{generation}$ générations et procure la meilleure solution trouvée.

2.2.5 Recherche locale de PSO-2S

Afin d'améliorer le fonctionnement de l'algorithme *PSO-2S* tout au long de la recherche, nous proposons d'intégrer une procédure de recherche locale, grâce à laquelle

Répulsion

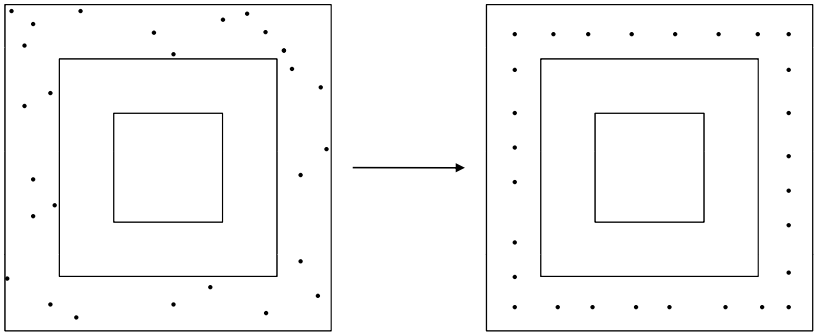
Entrées : $\epsilon = 10^{-4}$
Variables locales : $step, max_{disp}, e, \vec{F}, \vec{P}_i, \vec{P}_j, \vec{P}'_i$
$step \leftarrow 1$
répéter
  $max_{disp} \leftarrow 0$
  pour tout particule $\vec{P}_i \in P$ (population de l'essaim auxiliaire) **faire**
    $\vec{F} \leftarrow \sum_{j \neq i} \frac{\vec{P}_i - \vec{P}_j}{\left\|\vec{P}_i - \vec{P}_j\right\|^3}$
    $e \leftarrow \sum_{j \neq i} \frac{1}{\left\|\vec{P}_i - \vec{P}_j\right\|^2}$
    répéter
      $\vec{P}'_i \leftarrow crop(\vec{p}_i + step.\vec{F})$
      si $e < \sum_{j \neq i} \frac{1}{\left\|\vec{P}'_i - \vec{P}_j\right\|^2}$ **alors**
        $step \leftarrow \frac{step}{2}$
      sinon
        $step \leftarrow step \times 2$
      fin
    tant que $step$ peut être adapté et $e < \sum_{j \neq i} \frac{1}{\left\|\vec{P}'_i - \vec{P}_j\right\|^2}$
    si $\|\vec{P}_i - \vec{P}'_i\| > max_{disp}$ **alors**
      $max_{disp} \leftarrow \|\vec{P}_i - \vec{P}'_i\|$
    fin
    $\vec{P}_i \leftarrow \vec{P}'_i$
  fin
tant que $max_{disp} > \epsilon$
retourner P

PROCÉDURE 2.2: Heuristique de répulsion électrostatique utilisée pour repositionner les particules.

PSO-2S

Entrées : $max_{zone}, nb_{particule}, nb_{generation}$
$S1.size \leftarrow max_{zone}$
pour $p = 1$ à max_{zone} **faire**
    $S2_p.size \leftarrow nb_{part}$ * p
    $S2_p \leftarrow$ init_swarm($S2_p.size$, p)
    répulsion($S2_p$, p)
    pour $i = 1$ à $S2_p.size$ **faire**
        $V_i \leftarrow \omega V_i + \rho_1(Xpbest_i - X_i) + \rho_2(Xgbest_p - X_i)$
        $X_i \leftarrow X_i + V_i$
        si $f(X_i) < Pbest_i$ **alors**
            $Pbest_i \leftarrow f(X_i)$
            $Xpbest_i \leftarrow X_i$
        fin
    fin
fin
$S1.X[p] \leftarrow Xgbest[p]$
répéter
    pour $i = 1$ à max_{zone} **faire**
        Mettre à jour V_i et X_i de l'essaim S_1 selon les équations (1.3.1) et (1.3.2)
        Evaluer X_i
        Mettre à jour $Pbest_i$ et $Gbest$ de l'essaim S_1 en utilisant les équations (1.3.3) et (1.3.4)
    fin
tant que le critère d'arrêt n'est pas satisfait

PROCÉDURE 2.3: Algorithme PSO-2S.



(a) (b)

FIGURE 2.6: Procédure de répulsion : (a) $(zone_3, S2_3)$ avant répulsion, (b) $(zone_3, S2_3)$ après répulsion.

sera intensifiée à chaque itération dans la zone la plus prometteuse contenant l'optimum global, d'une manière peu couteuse en nombre d'évaluations.

La recherche locale de *PSO-2S* s'effectue dans la deuxième phase de *PSO-2S*, c'est-à-dire après la phase de construction de l'essaim principal. Cette recherche ne concerne pas les essaims auxiliaires, dont le travail est limité à découvrir des zones prometteuses et à construire l'essaim principal. Donc le rôle des essaims auxiliaires consiste à explorer et non exploiter l'espace de recherche. Ceci dit, la recherche locale est faite uniquement par l'essaim principal, où elle intervient à chaque itération en effectuant une recherche dans le voisinage de la meilleure solution trouvée *gbest* de l'essaim $S1$. Le principe général de cette recherche locale est le suivant : à partir de la meilleure solution *gbest*, où l'on connaît la valeur de la fonction objectif $f(gbest)$, on cherche la meilleure solution dans son voisinage (seulement deux voisins) en ajoutant et soustrayant du bruit sur une de ses dimensions. Ce bruit, nommé d_x, a une valeur plus petite que celle du *pas* calculé, dans l' équation (2.2.2), pour partitionner l'espace de recherche. En effet, on choisit aléatoirement une dimension $d \in \{1, \ldots, D\}$ puis, une fois, on ajoute le bruit d_x sur *gbest* et, une autre fois, on le soustrait, tout en essayant de déplacer *gbest* vers une meilleure solution voisine. Si l'on parvient à améliorer *gbest* alors on le remplace par la nouvelle solution trouvée, sinon rien ne change. Cette procédure de recherche ne nécessite pas beaucoup d'évaluations et permet à l'algorithme d'avoir plus de chances d'améliorer sa meilleure solution *gbest* tout au long de la recherche. Il est à noter que l'idée de cette recherche locale, faire un « petit » mouvement une fois « à droite » puis une fois « à gauche », n'est pas tout à fait nouvelle (voir par exemple dans [Boya 98]). Le pseudo-code de l'algorithme de la recherche locale est présenté dans la procédure 2.4.

Recherche locale

Entrées : d_x
Choisir aléatoirement une dimension $d \in \{1, \ldots, D\}$
$X_{best} \leftarrow gbest$
pour $i = 1$ à D **faire**
    si $i = d$ **alors**
        $S1.X^d_{newbest} \leftarrow S1.X^d_{best} + d_x$
    fin
    sinon
        $S1.X^i_{newbest} \leftarrow S1.X^i_{best}$
    fin
fin
évaluer $f(X^i_{newbest})$, comparer avec $f(X_{best})$ et garder la meilleure position
pour $i = 1$ à D **faire**
    si $i = d$ **alors**
        $S1.X^d_{newbest} \leftarrow S1.X^d_{best} - d_x$
    fin
    sinon
        $S1.X^i_{newbest} \leftarrow S1.X^i_{best}$
    fin
fin
évaluer $f(X^i_{newbest})$, comparer avec $f(X_{best})$ et garder la meilleure position

PROCÉDURE 2.4: La recherche locale utilisée dans l'algorithme PSO-2S.

2.2.6 Résultats et analyse

Dans cette section, nous présentons les résultats numériques obtenus par *PSO-2S*, ainsi que des comparaisons avec d'autres algorithmes, en utilisant les fonctions de test présentées en annexe. L'intérêt est de montrer que *PSO-2S* donne des résultats encourageants.

Nous commençons par une discussion sur le paramétrage de *PSO-2S* et des autres algorithmes. Ensuite, nous effectuons une analyse de la complexité de *PSO-2S*, ainsi qu'une analyse de la sensibilité de ses paramètres. Enfin, nous réalisons une analyse expérimentale de l'algorithme, ainsi qu'une comparaison de ses performances à celles des autres algorithmes de la littérature.

2.2.6.1 Paramétrage

Des expériences ont été effectuées pour comparer l'algorithme que nous avons proposé à cinq autres algorithmes de PSO (EPUS-PSO, CLPSO, CPSO-H, UPSO, SPSO2007), et ce en utilisant les fonctions (en 10-D et 30-D) présentées en annexe. Les résultats pour les deux algorithmes *PSO-2S* et SPSO2007 ont été obtenus grâce aux versions codées en C dans notre laboratoire. Les paramétrages et les résultats de toutes les autres variantes de PSO ont été tirés de [Hsie 09], où ces variantes ont été mises en œuvre en utilisant Matlab 7.1. *PSO-2S* et SPSO2007 utilisent une topologie *lbest* aléatoire pour échanger les informations entre les particules, UPSO [Pars 04] utilise une combinaison de topologies *lbest* et *gbest*, et CLPSO [Lian 06] utilise la stratégie globale d'apprentissage (*comprehensive learning strategy*) pour mettre à jour les vitesse des particules. Le tableau 2.1 récapitule les valeurs des paramètres, le coefficient d' inertie w et les coefficients d'accélération c_1 et c_2, de *PSO-2S* et des autres algorithmes. Deux types de tests sont définis : le premier étudie l'erreur moyenne et l'écart-type après un nombre fixé d'évaluations de la fonction objectif et le deuxième étudie le taux de réussite, l'erreur moyenne et l'erreur minimale après un nombre fixé d'évaluations ou après avoir atteint une précision donnée.

1. Le premier test consiste à exécuter le même algorithme 30 fois sur chacun des problèmes, une fois en 10-D et une autre en 30-D. L'erreur moyenne et l'écart-type sont calculés sur les 30 exécutions. Un critère d'arrêt unique est utilisé : l'algorithme s'arrête si le nombre d'évaluations de la fonction objectif dépasse Max_{Fes}, où Max_{Fes} est fixé à 40000 en 10-D et à 150000 en 30-D. Les paramètres de l'algorithme proposé, désignés par $nb_{generation}$ et $nb_{particule}$, sont fixés à 5 et 2 respectivement, dans les deux

Variantes de PSO	**Coefficient d'inertie**	**Coefficients d'accélération**
PSO-2S ([El D 12a])	$w = \frac{1}{2*ln(2)}$	$c_1 = c_2 = 0{,}5 + ln(2)$
EPUS-PSO ([Hsie 09])	$w = \frac{1}{2*ln(2)}$	$c_1 = c_2 = 0{,}5 + ln(2)$
CLPSO ([Lian 06])	$w(g) = 0{,}9 - \frac{0{,}5*g}{max_{gen}}$	$c_1 = c_2 = 1{,}49$
CPSO-H ([Berg 04])	$w(g) = 0{,}9 - \frac{0{,}5*g}{max_{gen}}$	$c_1 = c_2 = 1{,}49$
UPSO ([Pars 04])	$w = 0{,}72$	$c_1 = c_2 = 1{,}49$
SPSO2007 ([Cler 12])	$w = \frac{1}{2*ln(2)}$	$c_1 = c_2 = 0{,}5 + ln(2)$

TABLEAU 2.1: Paramétrages des algorithmes.

cas (10-D et 30-D). Ainsi, le paramètre max_{zone} est fixé à 20, sachant que ce paramètre définit aussi la taille d'essaim principal $S1$. La taille de population S de l'algorithme SPSO2007 est fixée à 16 et 20 (ici S est automatiquement calculé selon la formule : $S = 10 + 2\sqrt{D}$) dans les deux cas, 10-D et 30-D respectivement. La taille de population des autres algorithmes est similaire à celle utilisée dans *PSO-2S*.

2. Le deuxième test consiste à déterminer l'erreur moyenne, l'erreur minimale et le taux de réussite après avoir atteint un des deux critères d'arrêt (un nombre fixé d'évaluations ou une précision donnée). L'algorithme s'arrête si l'erreur entre la meilleure particule de l'essaim et l'objectif est inférieure à une précision donnée, ou si le nombre d'évaluations de la fonction objectif est supérieur à Max_{Fes}. Le tableau 2.2 présente les fonctions utilisées dans ce test, ainsi que leurs paramètres (*Espace de recherche*, Max_{Fes} et *Erreur acceptable*). Les autres paramètres restent les mêmes que dans le premier test. Pour ce test, les résultats sont enregistrés après avoir exécuté chaque algorithme 100 fois sur chacun des problèmes du tableau 2.2. Mais, contrairement au premier test, ce test a été effectué pour montrer l'intérêt d'utiliser l'heuristique de répulsion dans *PSO-2S*.

2.2.6.2 Analyse de complexité

L'ensemble des processus exécutés dans *PSO-2S* est présenté dans le tableau 2.3, ainsi que leurs complexités. La complexité totale de *PSO-2S* est calculée en additionnant les

f	***Fonction***	***Espace de recherche***	***Erreur acceptable***	$\mathbf{Max_{Fes}}$
f_3	*Rosenbrock*	$[-10, 10]^{30}$	0,0001	40000
f_4	*Ackley*	$[-32, 32]^{30}$	0,0001	40000
f_5	*Rastrigin*	$[-10, 10]^{30}$	0,0001	40000
f_8	*Griewank*	$[-100, 100]^{30}$	0,0001	40000
f_9	*Tripod*	$[-100, 100]^{2}$	0,0001	40000
f_{16}	*Shifted Rosenbrock*	$[-100, 100]^{10}$	0,01	100000
f_{17}	*Shifted Ackley*	$[-32, 32]^{30}$	0,0001	100000
f_{18}	*Shifted Rastrigin*	$[-5, 5]^{30}$	0,0001	100000
f_{19}	*Shifted Griewank*	$[-600, 600]^{30}$	0,0001	100000
f_{20}	*Shifted Sphere*	$[-100, 100]^{30}$	0,0001	100000

TABLEAU 2.2: Paramètres des fonctions utilisées pour le deuxième test.

complexités de chacun de ses processus. Enfin, pour un jeu de paramètres de *PSO-2S* fixé, et pour un nombre d'évaluations fixe, nous obtenons une complexité en $O(d)$ pour *PSO-2S* dans le pire des cas de calcul, où d est la dimension du problème.

2.2.6.3 Analyse de la sensibilité des paramètres

Dans cette section, nous étudions la sensibilité de *PSO-2S* en faisant varier deux de ses paramètres séparément ($nb_{generation}$ et max_{zone}), et en laissant les autres paramètres fixés à des valeurs données par défaut dans le tableau 2.1. Nous choisissons deux fonctions, f_{17} et f_{18} en 30-D, pour étudier les paramètres $nb_{generation}$ et max_{zone}. L'algorithme *PSO-2S* s'arrête lorsque 75000 évaluations sont effectuées, et l'erreur moyenne et l'écart-type sont calculés en exécutant l'algorithme 30 fois.

D'abord, nous étudions la sensibilité du paramètre $nb_{generation}$. $nb_{particule}$ est fixé à 2 et max_{zone} est fixé à 20. Le paramètre $nb_{generation}$ varie de 0 à 25 par pas de 5. La figure 2.7-a et la figure 2.7-b montrent l'erreur moyenne sur l'axe y, pour chaque valeur du paramètre $nb_{generation}$ qui varie sur l'axe x. Comme nous le constatons sur ces figures, il est préférable d'effectuer plusieurs générations des essaims $S2_i$ pour construire l'essaim $S1$, ce qui garantit que l'algorithme soit en mesure d'avoir des bonnes performances. En f_{17}, nous pouvons constater que la valeur moyenne oscille quand $nb_{generation}$ augmente, et elle devient stable

Processus	**Complexité**
Nombre total d'évaluations à l'initialisation	$O\left(nb_{generation}\ max_{zone}^2\ nb_{particule}\right)$
Nombre de générations à l'initialisation	$O\left(nb_{generation}\ max_{zone}\ nb_{particule}\right)$
Nombre de générations après l'initialisation	$O\left(\frac{Max_{Fes}}{max_{zone}} - nb_{generation}\ max_{zone}\ nb_{particule}\right)$
Génération aléatoire des particules	$O(d\ max_{zone})$
Heuristique de répulsion des particules	$O\left(\left(\log\left(\frac{1}{\epsilon}\right)\right)^2\ d\ max_{zone}^2\right)$
Une seule génération de PSO	$O(max_{zone}^2 + d\ max_{zone})$
PSO-2S pendant la phase d'initialisation	$O\left(\begin{array}{c}\left(nb_{generation}\ max_{zone}\ nb_{particule}\right) \times \\ \left(max_{zone}^2 + d\ max_{zone}\right) + \\ \left(\log\left(\frac{1}{\epsilon}\right)\right)^2\ d\ max_{zone}^2\end{array}\right)$
PSO-2S pendant la phase principale	$O\left(\begin{array}{c}\left(\frac{Max_{Fes}}{max_{zone}} - nb_{generation}\ max_{zone}\ nb_{particule}\right) \times \\ \left(max_{zone}^2 + d\ max_{zone}\right)\end{array}\right)$
Complexité totale de PSO-2S	$O\left(\ \mathbf{Max_{Fes}}\ (\mathbf{max_{zone}} + \mathbf{d}) + \left(\log\left(\frac{\mathbf{1}}{\mathbf{ffl}}\right)\right)^2\ \mathbf{d}\ \mathbf{max_{zone}^2}\ \right)$

TABLEAU 2.3: Analyse de la complexité de PSO-2S.

lorsque $nb_{generation} = 10$. Alors que, pour f_{18}, nous remarquons toujours une amélioration de l'erreur moyenne avec l'augmentation de $nb_{generation}$. Par conséquent, $nb_{generation}$ doit s'adapter avec le problème utilisé. Par ailleurs, l'écart-type ne varie pas significativement avec le changement de $nb_{generation}$.

De la même façon, nous étudions la sensibilité du paramètre max_{zone} sur les mêmes fonctions utilisées pour le paramètre $nb_{generation}$. $nb_{particule}$ est encore fixé à 2 et la valeur de $nb_{generation}$ fixée à 5. Les résultats obtenus montrent que l'erreur moyenne s'améliore en augmentant la valeur de max_{zone}, pour les deux fonctions f_{17} et f_{18}. L'écart-type diminue aussi avec l'augmentation du nombre de zones (i.e. le paramètre max_{zone}). La figure 2.7-c et la figure 2.7-d montrent les améliorations de l'erreur moyenne et de l'écart-type.

Pour conclure, nous pouvons remarquer que l'influence de max_{zone} sur les performances de *PSO-2S* est plus importante que celle du paramètre $nb_{generation}$. Par conséquent, nous pouvons donner à $nb_{generation}$ une valeur faible (environ 5) dans le but d'utiliser un petit nombre d'évaluations pour la construction de l'essaim principal $S1$. En effet, la diminution de la valeur de $nb_{generation}$ va nous permettre d'accroître le nombre de zones, sans augmenter le nombre d'évaluations, et donc d'améliorer les performances de l'algorithme avec le même nombre d'évaluations.

2.2.6.4 Comparaison avec d'autres algorithmes

Résultats des problèmes en 10-D Le tableau 3.2 présente les résultats obtenus par *PSO-2S* et cinq autres algorithmes de PSO, en termes d'erreurs moyennes et d'écarts-types sur 30 exécutions. Nous utilisons le premier test expliqué dans la section 2.2.6.1 sur les problèmes présentés dans la section 1.4.1 en dimension 10. Les meilleurs résultats parmi ceux obtenus par les six algorithmes sont présentés en gras.

Au vu de ces résultats expérimentaux, nous remarquons que l'algorithme proposé *PSO-2S* obtient les meilleurs résultats sur la plupart des fonctions utilisées. Il est suivi en deuxième place par l'algorithme EPUS-PSO. *PSO-2S* surpasse EPUS-PSO pour 7 fonctions sur 13, il obtient les mêmes résultats pour 3 fonctions et il est surpassé par EPUS-PSO sur 3 fonctions. Par ailleurs, nous pouvons remarquer que *PSO-2S* et EPUS-PSO obtiennent des résultats significativement meilleurs que les autres algorithmes sur f_5, f_7 et f_{15}. D'autre part, l'algorithme que nous avons proposé est meilleur que l'algorithme SPSO2007 sur toutes les fonctions testées, sauf sur la fonction f_1.

Le temps d'exécution pour les 30 exécutions de *PSO-2S* et SPSO2007 est présenté dans

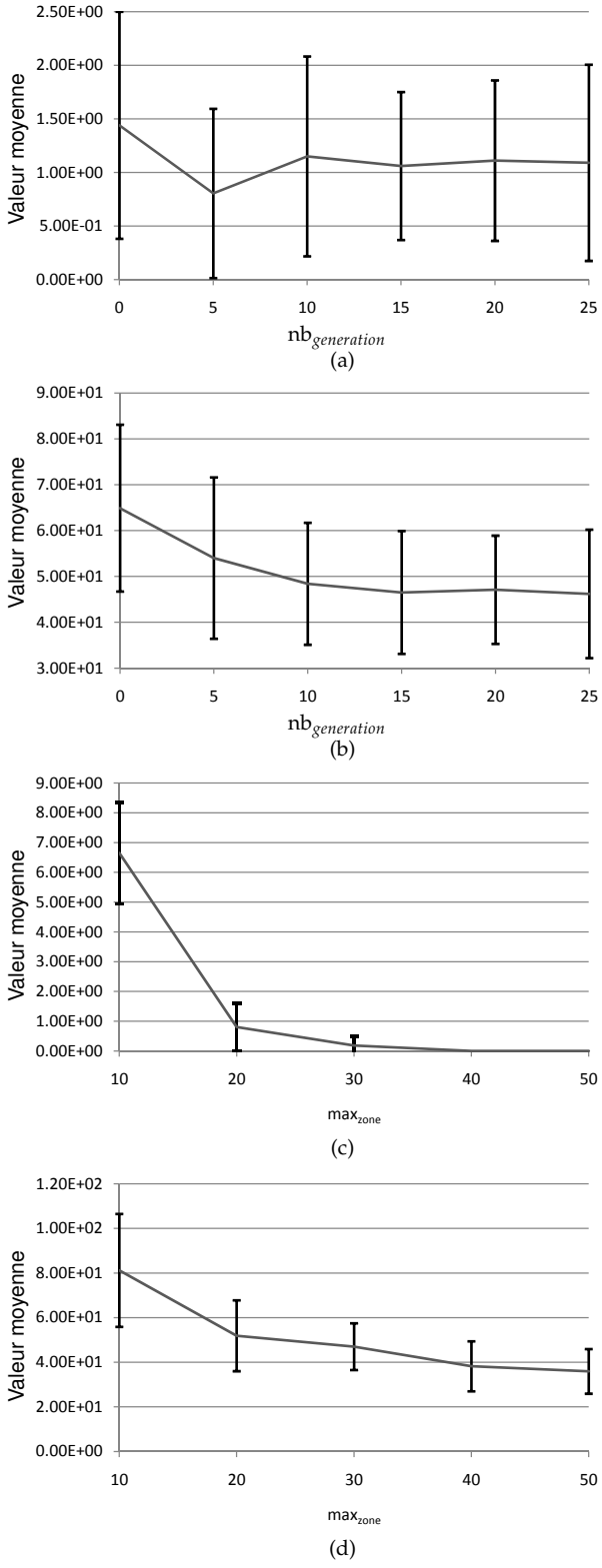


FIGURE 2.7: Analyse de la sensibilité des paramètres de PSO-2S : (a) l'évolution du paramètre $nb_{generation}$ pour f_{17}, (b) l'évolution du paramètre $nb_{generation}$ pour f_{18}, (c) l'évolution du paramètre max_{zone} pour f_{17}, (d) l'évolution du paramètre max_{zone} pour f_{18}.

le tableau 2.5. Nous pouvons remarquer que le temps d'exécution de *PSO-2S*, en utilisant l'heuristique de répulsion, est inférieur à celui de SPSO2007, pour toutes les fonctions, excepté la fonction f_6.

Résultats des problèmes en 30-D Les résultats expérimentaux pour les problèmes en 30-D sont présentés dans le tableau 3.3. Comme en 10-D, Nous comparons les algorithmes par rapport à l'erreur moyenne et l'écart-type sur 30 exécutions, et ceci, en utilisant les mêmes fonctions déjà utilisées en 10-D.

Bien que les problèmes en 30-D soient plus compliqués que ceux en 10-D, *PSO-2S* obtient toujours de bonnes performances. Les résultats montrent que *PSO-2S* se met à l'échelle avec le nombre de dimensions utilisées. Comme nous pouvons le voir, les résultats obtenus en 30-D ne sont pas significativement différents de ceux obtenus en 10-D. Par ailleurs, *PSO-2S* surpasse les autres algorithmes de PSO pour huit fonctions, et EPUS-PSO est toujours classé deuxième par rapport aux autres. *PSO-2S* surpasse de manière significative les autres algorithmes pour f_2, f_7 et f_{10}. L'algorithme proposé est meilleur que l'algorithme SPSO2007 pour toutes les fonctions, sauf pour f_{12}. Nous constatons aussi que CLPSO obtient des résultats significativement meilleurs que les autres algorithmes pour f_{12}.

Comme en 10-D, nous avons calculé le temps d'exécution de *PSO-2S* et SPSO2007 en 30-D. Les résultats sont présentés dans le tableau 2.7. Selon ces résultats, le temps d'exécution de *PSO-2S* est toujours comparable à celui de SPSO2007.

2.2.6.5 Comparaison de PSO-2S, avec et sans heuristique de répulsion, avec SPSO2007

Dans cette section, nous allons étudier l'effet de l'heuristique de répulsion sur l'algorithme *PSO-2S*, et comparer les résultats obtenus par *PSO-2S* avec et sans l'heuristique à ceux de SPSO2007. Les résultats sont exprimés en termes suivants : erreur moyenne, erreur minimale, taux de réussite et temps d'exécution de 100 exécutions pour chaque algorithme. Les fonctions de test utilisées dans cette comparaison et leurs paramètres sont présentés dans le tableau 2.2.

Les résultats obtenus sont présentés dans le tableau 2.8. Nous utilisons dans cette comparaison des fonctions qui sont translatées par rapport à leur forme d'origine pour montrer l'efficacité de *PSO-2S*. Car il existe des algorithmes qui ont la capacité de converger

f	**PSO-2S**	**EPUS-PSO**	**CLPSO**	**CPSO-H**	**UPSO**	**SPSO2007**
f_1	1,05e-086 ± 3,10e-086	**5,55e-153 ± 2,44e-134**	4,02e-021 ± 6,49e-021	9,01e-010 ± 1,38e-009	5,52e-090 ± 1,18e-089	4,00e-101 ± 1,80e-100
f_2	**6,18e-028 ± 2,13e-027**	9,32e-012 ± 4,16e-011	5,04e+002 ± 1,99e+002	1,62e+003 ± 9,35e+002	7,18e+001 ± 7,63e+001	1,83e-027 ± 4,03e-027
f_3	**2,35e-002 ± 1,56e-002**	9,13e-002 ± 1,05e-001	2,84e+000 ± 1,05e+000	1,08e+000 ± 1,18e+000	2,89e-001 ± 7,36e-002	3,10e-001 ± 9,95e-001
f_4	4,12e-015 ± 1,21e-015	**2,72e-015 ± 1,50e-015**	1,78e-011 ± 1,63e-011	7,02e-006 ± 5,77e-006	3,32e-015 ± 8,86e-016	3,85e-001 ± 2,07e-001
f_5	**0,00e+000 ± 0,00e+000**	**0,00e+000 ± 0,00e+000**	2,35e-009 ± 2,72e-009	3,83e-010 ± 7,36e-010	9,28e+000 ± 2,97e+000	5,11e+000 ± 2,29e+000
f_6	**5,85e-003 ± 2,35e-002**	3,21e-002 ± 8,11e-002	2,06e+000 ± 4,80e-001	4,12e+000 ± 1,72e+000	1,94e+000 ± 1,13e+000	4,04e-002 ± 1,78e-001
f_7	**1,35e-032 ± 5,47e-048**	**1,35e-032 ± 5,47e-048**	4,32e-001 ± 4,10e-001	1,09e+006 ± 2,14e+006	**1,35e-032 ± 5,47e-048**	7,32e-004 ± 2,74e-003
f_{10}	**3,38e-009 ± 9,02e-009**	3,99e-007 ± 1,63e-006	8,27e+002 ± 3,35e+002	2,15e+003 ± 1,19e+003	1,01e+002 ± 9,50e+001	1,02e-007 ± 4,12e-007
f_{11}	**4,21e-001 ± 7,77e-002**	8,96e-001 ± 2,15e+000	6,44e+000 ± 4,56e+000	3,01e+000 ± 2,52e+000	1,11e+000 ± 1,18e+000	5,73e-001 ± 1,05e+000
f_{12}	1,56e+001 ± 2,28e+000	**2,61e-015 ± 1,57e-015**	8,45e-008 ± 2,97e-007	1,31e+000 ± 9,68e-001	7,70e-002 ± 2,88e-001	2,03e+001 ± 8,54e-002
f_{13}	**2,65e-001 ± 9,93e-001**	7,26e+000 ± 4,31e+000	8,85e+000 ± 3,07e+000	2,81e+001 ± 1,39e+001	1,18e+001 ± 5,66e+000	1,26e+001 ± 7,00e+000
f_{14}	**6,43e-001 ± 3,45e-001**	1,12e+000 ± 1,36e+000	1,98e+000 ± 5,38e-001	3,48e+000 ± 1,41e+000	1,84e+000 ± 1,02e+000	4,20e+000 ± 1,50e+000
f_{15}	**1,35e-32 ± 5,47e-048**	**1,35e-032 ± 5,47e-048**	4,36e-001 ± 5,55e-001	1,09e+006 ± 2,37e+006	2,77e+000 ± 6,22e+000	1,10e-003 ± 3,30e-003
Performance de PSO-2S		7/10	12/13	12/13	9/12	12/13

TABLEAU 2.4: Comparaison des erreurs moyennes et des écarts-types des algorithmes en 10-D.

Function	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
PSO-2S	**10,016**	**4,731**	**6,864**	**10,657**	**5,855**	143,526	**9,250**	**5,550**	**8,276**	**11,947**	**7,737**	**103,474**	**10,224**
SPSO2007	11,259	12,559	11,042	13,356	12,801	**106,760**	16,110	12,328	12,057	14,117	13,451	107,067	16,585

TABLEAU 2.5: Temps de calcul de 30 exécutions pour chacun des algorithmes PSO-2S et SPSO2007, en 10-D.

plus rapidement si l'optimum global est situé soit à l'origine $[0, 0, \ldots, 0]$ soit au centre de l'espace de recherche [Zhon 04]. D'après ces résultats, nous pouvons constater que *PSO-2S* avec l'heuristique de répulsion donne de meilleures performances, et il surpasse SPSO2007 pour toutes les fonctions, exceptée f_{19}. De plus, les taux de réussite présentés dans ce tableau montrent l'efficacité de l'algorithme proposé.

Un test statistique de Wilcoxon-Mann-Whitney [Wilc 45, Mann 47], est appliqué sur les résultats de *PSO-2S* et SPSO2007, avec un niveau de confiance de 95%, afin de déterminer si leurs performances sont significativement différentes ou non. *PSO-2S* et SPSO2007 ont été exécutés 100 fois et les valeurs des meilleures solutions trouvées à la fin de chaque exécution ont été utilisées pour calculer les valeurs de *W* et *P-value*. Ces valeurs sont données dans le tableau 2.8 dans les deux dernières colonnes. La valeur de *P-value* est présentée en gras quand les résultats sont significativement différents. Nous pouvons conclure que *PSO-2S* surpasse SPSO2007 pour toutes les fonctions avec des résultats significativement différents, sauf pour f_{19}.

Ce travail a fait l'objet de la publication d'un article dans la revue *Computational Optimization and Applications* [El D 12a].

2.3 DEPSO-2S : hybridation de PSO-2S avec un algorithme à évolution différentielle DELG

DEPSO-2S est une nouvelle variante de *PSO-2S* issue des améliorations de cet algorithme. *DEPSO-2S* est basé sur l'hybridation de deux métaheuristiques : DE et PSO. Ces deux dernières ont connu un grand développement ces dernières années, ainsi qu'un nombre important d'applications industrielles et scientifiques. D'où l'intérêt de tenter l'hybridation de ces deux algorithmes.

f	**PSO-2S**	**EPUS-PSO**	**CLPSO**	**CPSO-H**	**UPSO**	**SPSO2007**
f_1	3,02e-119 ± 1,56e-118	**8,50e-263 ± 1,02e-262**	1,34e-025 ± 1,71e-025	1,57e-010 ± 2,99e-10	8,79e-123 ± 3,56e-122	3,36e-107 ± 1,79e-106
f_2	**6,18e-011 ± 2,63e-011**	197,e+0000 ± 4,69e+000	2,41e+004 ± 6,78e+003	8,36e+004 ± 6,58e+004	1,07e+004 ± 4,23e+003	8,25e-008 ± 9,00e-008
f_3	**9,75e+000 ± 9,59e-001**	2,55e+001 ± 3,53e-001	2,01e+001 ± 3,37e+000	1,03e+001 ± 1,37e+001	1,31e+001 ± 2,98e+000	1,23e+001 ± 1,47e+000
f_4	1,63e-001 ± 3,85e-001	**3,91e-015 ± 1,07e-015**	9,90e-014 ± 3,80e-014	3,90e-006 ± 5,98e-006	1,33e+000 ± 8,58e-001	1,36e+000 ± 1,13e+000
f_5	1,00e+000 ± 1,18e+001	**0,00e+000 ± 0,00e+000**	1,87e-009 ± 5,34e-009	3,15e-010 ± 1,05e-009	7,63e+001 ± 1,45e+001	4,88e+001 ± 1,44e+001
f_6	**3,63e-001 ± 2,79e-001**	4,08e-001 ± 1,22e+000	1,49e+001 ± 1,69e+000	1,36e+000 ± 3,38e+000	2,06e+000 ± 3,37e+000	1,55e+001 ± 1,09e+000
f_7	**1,35e-032 ± 5,47e-048**	**1,35e-032 ± 5,47e-048**	1,62e+003 ± 5,92e+003	2,58e+008 ± 4,41e+008	1,34e+001 ± 2,34e+001	4,21e-001 ± 1,10e+000
f_{10}	**7,80e-005 ± 9,25e-005**	1,20e+001 ± 2,46e+001	2,38e+004 ± 6,49e+003	3,87e+004 ± 1,71e+004	1,27e+004 ± 4,77e+003	1,09e-001 ± 1,80e-001
f_{11}	**1,39e+001 ± 1,48e-001**	1,96e+001 ± 2,91e+001	3,04e+001 ± 1,32e+001	3,03e+001 ± 2,59e+001	2,15e+001 ± 1,89e+001	2,77e+001 ± 1,31e+001
f_{12}	1,69e+001 ± 1,66e+000	6,81e-001 ± 1,02e+000	**2,22e-006 ± 8,29e-006**	1,52e+000 ± 8,30e-001	1,63e+000 ± 9,49e-001	2,09e+000 ± 1,31e+000
f_{13}	**1,40e+000 ± 2,73e-001**	3,76e+001 ± 1,48e+001	4,70e+001 ± 6,85e+000	9,52e+001 ± 2,93e+001	7,70e+001 ± 1,70e+001	9,04e+001 ± 4,03e+001
f_{14}	**3,34e+000 ± 1,22e+000**	4,26e+000 ± 2,97e+000	1,51e+001 ± 1,93e+000	1,36e+001 ± 3,56e+000	2,02e+001 ± 4,19e+000	3,27e+001 ± 3,19e+000
f_{15}	1,89e-002 ± 2,38e-002	**1,35e-032 ± 5,47e-048**	3,43e+004 ± 1,57e+005	2,44e+008 ± 4,66e+008	2,47e+001 ± 2,77e+001	5,37e+001 ± 2,92e+001
Performance de PSO-2S		7/12	10/13	10/13	11/13	12/13

TABLEAU 2.6: Comparaison des erreurs moyennes et des écarts-types des algorithmes en 30-D.

Function	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
PSO-2S	**38,178**	**36,585**	**39,852**	135,147	118,646	1472,139	**119,396**	**56,818**	**64,566**	107,005	**83,418**	1603,414	**98,412**
SPSO2007	50,625	61,446	51,445	**69,378**	**67,110**	**1038,837**	122,355	80,664	71,624	**88,712**	83,819	**1078,324**	124,757

TABLEAU 2.7: Temps de calcul de 30 exécutions pour chacun des algorithmes PSO-2S et SPSO2007, en 30-D.

Dans cette section, nous présentons tout d'abord une variante de DE, nommée DELG, puis nous décrivons les modifications proposées pour l'adapter à *PSO-2S*. Ensuite, nous décrivons la structure générique du nouvel algorithme proposé *DEPSO-2S*. Enfin, nous réalisons une étude expérimentale de l'algorithme, ainsi qu'une comparaison de ses performances à celles des autres algorithmes de la littérature.

2.3.1 Présentation de l'algorithme DELG

DELG est une variante de l'algorithme DE qui a été proposée dans [Chak 06]. Il utilise le concept de voisinage de chaque vecteur qui vise à équilibrer les capacités d'exploration et d'exploitation de l'évolution différentielle, sans avoir besoin d'évaluer la fonction. Cela se fait grâce à l'utilisation de deux opérateurs de mutation : la mutation locale et la mutation globale. Contrairement au standard DE, DELG utilise une topologie du voisinage qui définit avec qui chacune des particules va pouvoir communiquer. La topologie utilisée dans DELG est en anneau, où chaque particule est reliée à un nombre défini d'autres particules. Supposons que nous ayons une population $P = [\vec{X}_1, \vec{X}_2, \ldots, \vec{X}_{np}]$. Pour chaque vecteur $\vec{X}_i$, on définit un voisinage de rayon k, constitué des vecteurs $\vec{X}_{i-k}, \ldots, \vec{X}_i, \ldots, \vec{X}_{i+k}$. Les deux voisins immédiats du vecteur $\vec{X}_1$ sont, quant à eux, $\vec{X}_{np}$ et $\vec{X}_2$.

La mutation locale :
Pour chaque membre de la population, une mutation locale est appliquée en utilisant le meilleur vecteur $\vec{X}_{nbest}$ dans le voisinage de ce membre et de deux autres vecteurs choisis $\vec{X}_p$ et $\vec{X}_q$ dans le même voisinage. Le modèle peut être exprimé comme suit :

$$\vec{L}_i(t) = \vec{X}_i(t) + \lambda.(\vec{X}_{nbest}(t) - \vec{X}_i(t)) + F.(\vec{X}_p(t) - \vec{X}_q(t)) \tag{2.3.1}$$

où l'indice *nbest* indique le meilleur vecteur dans le voisinage de $\vec{X}_i$, et $p, q \in \{i-k, \ldots, i+k\}$.

	f	SPSO2007	PSO-2S avec répulsion	PSO-2S sans répulsion		
		Erreur moyenne Erreur minimale Taux de réussite Temps d'exécution	Erreur moyenne Erreur minimale Taux de réussite Temps d'exécution	Erreur moyenne Erreur minimale Taux de réussite Temps d'exécution	**W**	**P-value**
B	f_3	3,15e+001 1,82e-001 0,0% 46,750s	**2,23e+001** 2,13e+001 0,0% 24,938s	2,24e+001 2,14e+001 0,0% 21,021s	6957	**1,714e-06**
	f_4	9,94e-001 7,11e-001 33% 46,313s	**2,03e-001** 8,08e-005 **81%** 38,234s	7,34e-001 8,69e-005 47% 27,389s	6909	**3,024e-06**
	f_5	5,71e+001 2,49e+001 0,0% 62,835s	**2,00e+000** 6,55e-005 **24%** 64,805s	7,50e+000 8,92e-005 5% 33,160s	10000	**< 2,2e-16**
	f_8	1,31e-002 4,90e-005 43% 46,542s	**2,93e-003** 5,30e-005 **77%** 47,028s	4,24e-003 6,19e-005 61% 24,935s	6341,5	**0,001049**
	f_9	6,02e-001 3,43e-006 51% 19,047s	**2,45e-003** 1,79e-005 **98%** 9,153s	2,05e-001 7,31e-006 75% 4,572s	5946	**0,02050**
D	f_{16}	2,81e+000 7,26e-003 76% 59,282s	**2,06e+000** 9,53e-003 73% 31,202s	2,89e+000 9,57e-003 **77%** 17,615s	4070	**0,02293**
	f_{17}	1,05e+000 7,69e-005 36% 108,104s	**2,34e-001** 7,96e-005 **63%** 197,423s	1,27e+000 8,87e-005 24% 150,201s	6135,5	**0,000343**
	f_{18}	5,44e+001 2,92e+001 0,0% 154,163s	**4,32e+001** 1,39e+001 0,0% 148,035s	5,43e+001 3,08e+001 0,0% 98,578s	7824	**0,00668**
	f_{19}	**2,36e-002** 2,72e-005 36% 117,191s	3,35e-002 7,52e-005 38% 211,496s	3,78e-002 8,03e-005 **42%** 120,180s	4618	0,3512
E	f_{20}	8,39e-005 1,88e-005 **100%** 12,185s	**6,23e-005** 3,74e-005 **100%** 28,686s	9,31e-005 7,50e-005 **100%** 10,201s	3444	**0,0001442**

TABLEAU 2.8: Comparaison des résultats obtenus par PSO-2S (avec et sans) heuristique de répulsion.

La mutation globale :
De plus, pour tous les vecteurs $\vec{X}_i$, une mutation globale est appliquée en utilisant la meilleure particule $\vec{X}_{best}$ de la population, ainsi que deux autres vecteurs $\vec{X}_r$ et $\vec{X}_s$ choisis aléatoirement. La formule de la mutation globale est alors :

$$\vec{G}_i(t) = \vec{X}_i(t) + \lambda'.(\vec{X}_{best}(t) - \vec{X}_i(t)) + F'.(\vec{X}_r(t) - \vec{X}_s(t)) \tag{2.3.2}$$

où l'indice *best* indique le meilleur vecteur dans l'ensemble de la population, et $r, s \in \{1, \ldots, np\}$.

En effet, la mutation locale favorise l'exploration, car les différents membres de la population doivent en général être biaisés par des membres différents. A l'inverse, la mutation globale encourage l'exploitation, puisque tous les vecteurs de la population sont biaisés par le même vecteur (*global best*). Après avoir combiné ces deux mutations en utilisant un poids scalaire variable dans le temps, l'opérateur de mutation final de DELG permettra d'obtenir un bon équilibre entre la recherche locale et la recherche globale. L'équation de cette combinaison de mutations est la suivante :

$$\vec{V}_i(t) = w.\vec{G}_i(t) + (1 - w).\vec{L}_i(t) \tag{2.3.3}$$

Le facteur de poids w varie linéairement avec le temps selon la formule suivante :

$$w = w_{min} + (w_{max} - w_{min}).(\frac{iter}{MAXIT}) \tag{2.3.4}$$

où $iter$ est l'itération courante et $MAXIT$ est le nombre maximal d'itérations. w_{max} et w_{min} désignent respectivement les valeurs maximum et minimum du poids w, telles que : $w_{min}, w_{max} \in \{0, \ldots, 1\}$.

2.3.2 Adaptation de DELG à PSO-2S

Afin que l'hybridation de DELG avec *PSO-2S* aboutisse à l'amélioration des performances, et rende DELG compatible avec la stratégie de partage de l'espace de recherche utilisée dans *PSO-2S*, une autre modification a été proposée. L'équation (2.3.3) a été modifiée pour que la zone courante devienne dynamique : au début de la recherche, les vecteurs de chaque zone peuvent aller au-delà de la zone et donc explorent des régions périphériques, puis ils se restreignent progressivement à l'intérieur de la zone. En effet, pour

chaque vecteur courant $\vec{X}_i$ d'un essaim auxiliaire, le vecteur mutant $\vec{V}_i$ calculé dans l'équation (2.3.5) dépasse la zone où l'essaim auxiliaire est initialisé vers les zones extérieures, du fait de la valeur de w. Au début des itérations réservées aux essaims auxiliaires, le vecteur $\vec{V}_i$ va être calculé en utilisant la valeur de $w = w_{min}$, qui a pour effet d'augmenter la valeur du vecteur $\vec{V}_i$. Puis cette valeur diminue progressivement d'une itération à l'autre ; pour qu'à la fin des itérations, le vecteur $\vec{V}_i$ sorte de sa zone mais, cette fois, vers l'intérieur.

La nouvelle formule de combinaison des mutations globale et locale est la suivante :

$$\vec{V}_i(t) = (1 - w).\vec{G}_i(t) + (1 - w).\vec{L}_i(t) \tag{2.3.5}$$

En choisissant $w_{min} < 0{,}5$ et $w_{max} > 0{,}5$, le vecteur $\vec{V}_i(t)$ est alors libre de sortir de sa zone de recherche pour des valeurs de t proches de 0. $\vec{V}_i(t)$ est ensuite progressivement réduit, à mesure que le temps t augmente.

2.3.3 Algorithme DEPSO-2S

Après avoir présenté dans les deux sections précédentes l'algorithme DELG, ainsi que la modification faite pour l'adapter à *PSO-2S*, nous allons présenter dans cette section la structure générale du nouvel algorithme proposé *DEPSO-2S*. Cet algorithme utilise une hybridation de deux variantes d'algorithmes d'optimisation (une variante de DE et une variante de PSO). Nous notons que nous utilisons une variante modifiée de l'algorithme DE, nommée DELG, pour l'intégrer dans *DEPSO-2S*. Cette variante est utilisée uniquement dans la première phase de *DEPSO-2S*, qui est la phase de construction de l'essaim principal $S1$. En effet, chaque essaim auxiliaire a un nombre fixe d'itérations à effectuer en utilisant DELG à la place d'un standard PSO utilisé dans *PSO-2S*. Mais, comme dans *PSO-2S*, DELG effectue ses itérations et procure à la fin la meilleure solution trouvée pendant la recherche, et le nombre d'essaims auxiliaires utilisés est égal au nombre de zones partitionnées. La même technique est appliquée dans toutes les zones et, à chaque fois, la meilleure solution obtenue par DELG sera sauvegardée pour construire la population de l'essaim principal $S1$. Le rôle de DELG se termine une fois l'essaim $S1$ construit. Dans la deuxième phase de l'algorithme, $S1$ se comporte de la même façon que dans *PSO-2S*.

En résumé, dans la première partie de l'algorithme *PSO-2S*, la phase de construction de l'essaim principal a été modifiée pour utiliser DELG à la place d'un algorithme de PSO. La figure 2.8 illustre la structure générale de l'algorithme *DEPSO-2S*.

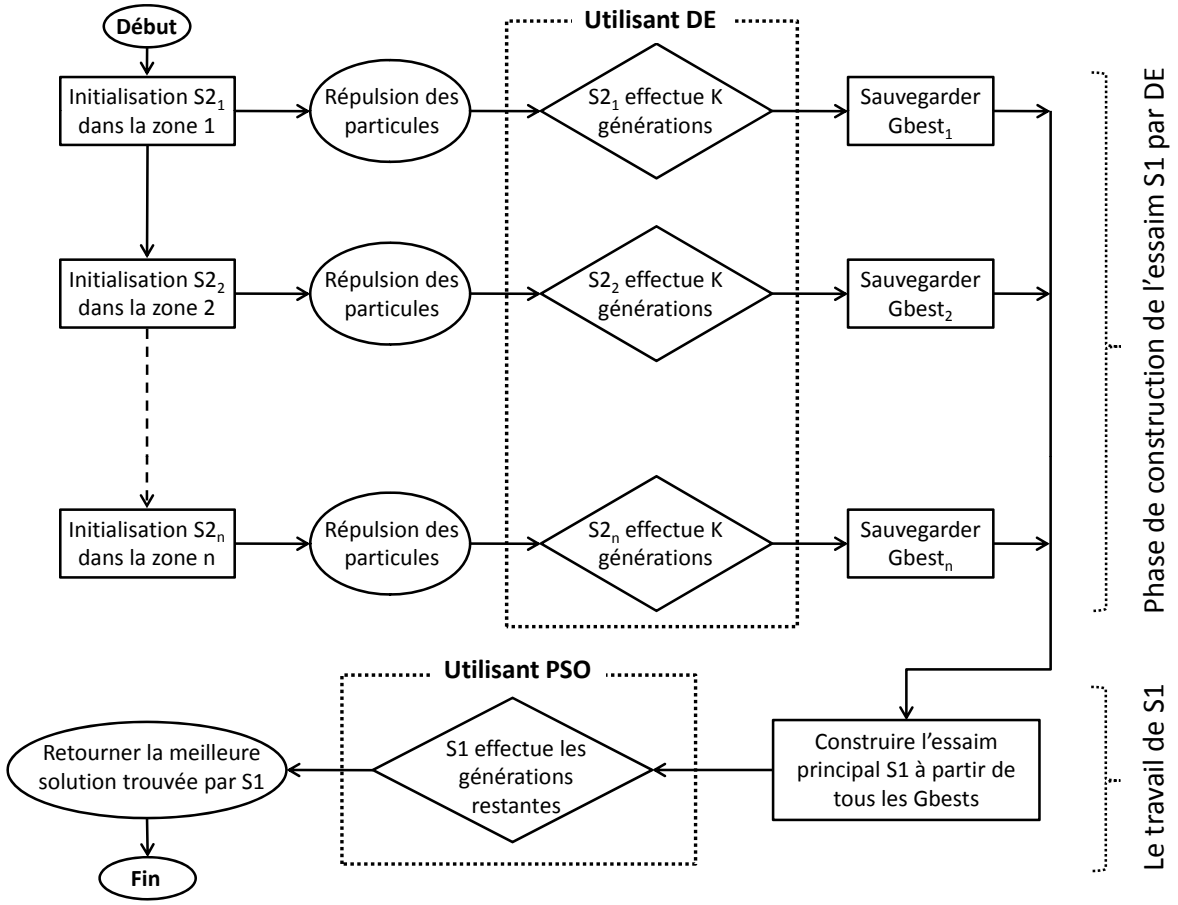


FIGURE 2.8: La structure générale de l'algorithme DEPSO-2S.

2.3.4 Résultats et discussion

Dans cette section, nous présentons les résultats numériques obtenus par *DEPSO-2S*, ainsi que ceux obtenus par *PSO-2S* et les autres algorithmes utilisés dans la section 2.2.6. L'intérêt est de comparer ces résultats et de montrer que *DEPSO-2S* améliore l'ancienne version de *PSO-2S*, et donne des résultats encourageants.

Nous commençons par une brève discussion sur le nouveau paramétrage de *DEPSO-2S*. Ensuite, nous réalisons une analyse expérimentale de l'algorithme, ainsi qu'une comparaison de ses performances à celles de *PSO-2S* et des autres algorithmes.

2.3.4.1 Paramétrage

Nous utilisons dans cette comparaison les mêmes valeurs de paramètres que dans la section 2.2.6.1, pour les algorithmes *PSO-2S*, EPUS-PSO, CLPSO, CPSO-H, UPSO et SPSO 2007. De plus, nous ajoutons les nouveaux paramètres Cr, F, F', λ et λ', utilisés dans *DEPSO-2S*. Nous fixons Cr à 0,5 et F, F', λ et λ' à 0,2, pour toutes les comparaisons. D'abord, nous allons faire des tests sur les différents algorithmes, de la même manière que ceux faits dans la section 2.2.6, en utilisant les mêmes fonctions en 10-D et 30-D. Ensuite, un autre test sera réalisé en utilisant plusieurs problèmes réels de la littérature, parmi ceux présentés en annexe.

2.3.4.2 Comparaison avec d'autres algorithmes

Dans le but de comparer *DEPSO-2S* avec d'autres algorithmes d'optimisation et de voir si nous avons réussi à améliorer l'ancienne version *PSO-2S*, nous choisissons les mêmes fonctions que dans la section 2.2.6 en 10-D et 30-D, ainsi que les mêmes algorithmes y compris *PSO-2S*. Les résultats obtenus par tous les algorithmes en 10-D sont présentés dans le tableau 2.9, et les résultats en 30-D sont présentés dans le tableau 2.10. Chaque exécution a été répétée 30 fois, en fixant le nombre maximum d'évaluations Max_{Fes} à 40000 en 10-D et 150000 en 30-D. L'erreur moyenne et l'écart-type sont calculés sur les 30 exécutions.

Nous commençons tout d'abord par analyser le tableau 2.9, où nous constatons des progrès sur la plupart des fonctions. De plus, nous pouvons voir que *DEPSO-2S* améliore significativement ses performances sur les fonctions f_6 et f_{13}, par rapport à l'ancienne version. Par contre, nous constatons une diminution des performances pour les fonctions f_1

(*Sphere*), f_3 et f_{11} (*Rosenbrock* et *Rotated Rosenbrock*, qui sont des fonctions mal conditionnées).

Nous analysons maintenant les résultats du tableau 2.10, où nous constatons que *DEPSO-2S* a amélioré les résultats pour la plupart des fonctions, par rapport aux anciens résultats obtenus par *PSO-2S*. Nous remarquons une amélioration significative sur les fonctions f_4, f_5 et surtout sur la fonction f_{13}, où *DEPSO-2S* trouve la solution optimale de cette fonction comme en 10-D. Par contre, nous remarquons une diminution des performances sur quelques fonctions qui sont les mêmes qu'en 10-D, comme f_1, f_3 et f_{11}.

Pour montrer que *DEPSO-2S* a des meilleures performances sur des problèmes divers par rapport à d'autres algorithmes incluant l'algorithme DE, nous choisissons quelques problèmes réels d'optimisation connus dans la littérature. Les résultats obtenus et présentés dans le tableau 2.11 ont été tirés de notre article [El D 12b]. Ce tableau présente les problèmes, la dimension et le nombre maximum d'évaluations Max_{Fes} qui sont fixés pour chaque problème, ainsi que les résultats obtenus par tous les algorithmes en terme d'erreur moyenne et l'écart-type sur 100 exécutions. A partir de ce tableau, nous pouvons remarquer que l'algorithme *DEPSO-2S* est plus performant que les autres algorithmes pour la plupart des problèmes testés, en termes de qualité de solution trouvée. En effet, *DEPSO-2S* obtient de meilleures résultats sur tous les problèmes, excepté pour P_4, où l'algorithme proposé est surpassé par SPSO2007.

Pour plus de détails sur les problèmes réels utilisés, on peut se référer à l'article [El D 12b].

Pour conclure, nous constatons que *DEPSO-2S* obtient de meilleurs résultats que *PSO-2S* et les autres algorithmes sur la plupart des fonctions testées, notamment sur des fonctions difficiles ou fortement multimodales (*Ackley*, *Rastrigin*, *Weierstrass*) et sur la majorité des problèmes réels testés. Par contre, pour des fonctions telles que *Sphere* et *Rosenbrock*, les performances de *DEPSO-2S* sont inférieures à celles des autres algorithmes. Néanmoins, en augmentant le nombre de zones et le nombre maximum d'évaluations Max_{Fes} autorisées, les performances de *DEPSO-2S* s'améliorent.

f	**DEPSO-2S**	**PSO-2S**	**EPUS-PSO**	**CLPSO**	**CPSO-H**	**UPSO**	**SPSO2007**
f_1	1,30e-024 ± 3,33e-002	1,05e-086 ± 3,10e-086	**5,55e-153 ± 2,44e-134**	4,02e-021 ± 6,49e-021	9,01e-010 ± 1,38e-009	5,52e-090 ± 1,18e-089	4,00e-101 ± 1,80e-100
f_2	**6,18e-028 ± 2,13e-027**	**6,18e-028 ± 2,13e-027**	9,32e-012 ± 4,16e-011	5,04e+002 ± 1,99e+002	1,62e+003 ± 9,35e+002	7,18e+001 ± 7,63e+001	1,83e-027 ± 4,03e-027
f_3	8,52e-002 ± 3,35e-002	**2,35e-002 ± 1,56e-002**	9,13e-002 ± 1,05e-001	2,84e+000 ± 1,05e+000	1,08e+000 ± 1,18e+000	2,89e-001 ± 7,36e-002	3,10e-001 ± 9,95e-001
f_4	4.79e-013 ± 2.53e-002	4,12e-015 ± 1,21e-015	**2,72e-015 ± 1,50e-015**	1,78e-011 ± 1,63e-011	7,02e-006 ± 5,77e-006	3,32e-015 ± 8,86e-016	3,85e-001 ± 2,07e-001
f_5	**0,00e+000 ± 0,00e+000**	**0,00e+000 ± 0,00e+000**	**0,00e+000 ± 0,00e+000**	2,35e-009 ± 2,72e-009	3,83e-010 ± 7,36e-010	9,28e+000 ± 2,97e+000	5,11e+000 ± 2,29e+000
f_6	**2,20e-007 ± 7,38e-006**	5,85e-003 ± 2,35e-002	3,21e-002 ± 8,11e-002	2,06e+000 ± 4,80e-001	4,12e+000 ± 1,72e+000	1,94e+000 ± 1,13e+000	4,04e-002 ± 1,78e-001
f_7	**1,35e-032 ± 5,47e-048**	**1,35e-032 ± 5,47e-048**	**1,35e-032 ± 5,47e-048**	4,32e-001 ± 4,10e-001	1,09e+006 ± 2,14e+006	**1,35e-032 ± 5,47e-048**	7,32e-004 ± 2,74e-003
f_{10}	**3,38e-009 ± 9,02e-009**	**3,38e-009 ± 9,02e-009**	3,99e-007 ± 1,63e-006	8,27e+002 ± 3,35e+002	2,15e+003 ± 1,19e+003	1,01e+002 ± 9,50e+001	1,02e-007 ± 4,12e-007
f_{11}	5,52e-001 ± 4,28e-002	**4,21e-001 ± 7,77e-002**	8,96e-001 ± 2,15e+000	6,44e+000 ± 4,56e+000	3,01e+000 ± 2,52e+000	1,11e+000 ± 1,18e+000	5,73e-001 ± 1,05e+000
f_{12}	5,66e+000 ± 5,15e-001	1,56e+001 ± 2,28e+000	**2,61e-015 ± 1,57e-015**	8,45e-008 ± 2,97e-007	1,31e+000 ± 9,68e-001	7,70e-002 ± 2,88e-001	2,03e+001 ± 8,54e-002
f_{13}	**0,00e+000 ± 0,00e+000**	2,65e-001 ± 9,93e-001	7,26e+000 ± 4,31e+000	8,85e+000 ± 3,07e+000	2,81e+001 ± 1,39e+001	1,18e+001 ± 5,66e+000	1,26e+001 ± 7,00e+000
f_{14}	**1,53e-001 ± 3,67e-002**	6,43e-001 ± 3,45e-001	1,12e+000 ± 1,36e+000	1,98e+000 ± 5,38e-001	3,48e+000 ± 1,41e+000	1,84e+000 ± 1,02e+000	4,20e+000 ± 1,50e+000
f_{15}	**1,35e-32 ± 5,47e-048**	**1,35e-32 ± 5,47e-048**	**1,35e-032 ± 5,47e-048**	4,36e-001 ± 5,55e-001	1,09e+006 ± 2,37e+006	2,77e+000 ± 6,22e+000	1,10e-003 ± 3,30e-003

TABLEAU 2.9: Comparaison des erreurs moyennes et écarts-types des algorithmes en 10-D.

f	**DEPSO-2S**	**PSO-2S**	**EPUS-PSO**	**CLPSO**	**CPSO-H**	**UPSO**	**SPSO2007**
f_1	1,25e-025 ± 3,24e-024	3,02e-119 ± 1,56e-118	**8,50e-263 ± 1,02e-262**	1,34e-025 ± 1,71e-025	1,57e-010 ± 2,99e-10	8,79e-123 ± 3,56e-122	3,36e-107 ± 1,79e-106
f_2	**3,88e-012 ± 2,50e-011**	6,18e-011 ± 2,63e-011	197,e+0000 ± 4,69e+000	2,41e+004 ± 6,78e+003	8,36e+004 ± 6,58e+004	1,07e+004 ± 4,23e+003	8,25e-008 ± 9,00e-008
f_3	1,68e+001 ± 5,43e-002	**9,75e+000 ± 9,59e-001**	2,55e+001 ± 3,53e-001	2,01e+001 ± 3,37e+000	1,03e+001 ± 1,37e+001	1,31e+001 ± 2,98e+000	1,23e+001 ± 1,47e+000
f_4	6,48e-014 ± 3,33e-014	1,63e-001 ± 3,85e-001	**3,91e-015 ± 1,07e-015**	9,90e-014 ± 3,80e-014	3,90e-006 ± 5,98e-006	1,33e+000 ± 8,58e-001	1,36e+000 ± 1,13e+000
f_5	3,98e-014 ± 2,33e-014	1,00e+000 ± 1,18e+001	**0,00e+000 ± 0,00e+000**	1,87e-009 ± 5,34e-009	3,15e-010 ± 1,05e-009	7,63e+001 ± 1,45e+001	4,88e+001 ± 1,44e+001
f_6	**1,69e-002 ± 3,36e-002**	3,63e-001 ± 2,79e-001	4,08e-001 ± 1,22e+000	1,49e+001 ± 1,69e+000	1,36e+000 ± 3,38e+000	2,06e+000 ± 3,37e+000	1,55e+001 ± 1,09e+000
f_7	**1,35e-032 ± 5,47e-048**	**1,35e-032 ± 5,47e-048**	**1,35e-032 ± 5,47e-048**	1,62e+003 ± 5,92e+003	2,58e+008 ± 4,41e+008	1,34e+001 ± 2,34e+001	4,21e-001 ± 1,10e+000
f_{10}	**7,80e-005 ± 9,25e-005**	**7,80e-005 ± 9,25e-005**	1,20e+001 ± 2,46e+001	2,38e+004 ± 6,49e+003	3,87e+004 ± 1,71e+004	1,27e+004 ± 4,77e+003	1,09e-001 ± 1,80e-001
f_{11}	2,55e+001 ± 3,92e-002	**1,39e+001 ± 1,48e-001**	1,96e+001 ± 2,91e+001	3,04e+001 ± 1,32e+001	3,03e+001 ± 2,59e+001	2,15e+001 ± 1,89e+001	2,77e+001 ± 1,31e+001
f_{12}	1,36e+000 ± 1,59e-001	1,69e+001 ± 1,66e+000	6,81e-001 ± 1,02e+000	**2,22e-006 ± 8,29e-006**	1,52e+000 ± 8,30e-001	1,63e+000 ± 9,49e-001	2,09e+000 ± 1,31e+000
f_{13}	**0,00e+000 ± 0,00e+000**	1,40e+000 ± 2,73e-001	3,76e+001 ± 1,48e+001	4,70e+001 ± 6,85e+000	9,52e+001 ± 2,93e+001	7,70e+001 ± 1,70e+001	9,04e+001 ± 4,03e+001
f_{14}	**9,70e-001 ± 5,95e-002**	3,34e+000 ± 1,22e+000	4,26e+000 ± 2,97e+000	1,51e+001 ± 1,93e+000	1,36e+001 ± 3,56e+000	2,02e+001 ± 4,19e+000	3,27e+001 ± 3,19e+000
f_{15}	**1,35e-032 ± 5,47e-048**	1,89e-002 ± 2,38e-002	**1,35e-032 ± 5,47e-048**	3,43e+004 ± 1,57e+005	2,44e+008 ± 4,66e+008	2,47e+001 ± 2,77e+001	5,37e+001 ± 2,92e+001

TABLEAU 2.10: Comparaison des erreurs moyennes et écarts-types des algorithmes en 30-D.

P	Problème	D	Max_{Fes}	DE	SPSO2007	PSO-2S	DEPSO-2S
P_1	*Gas transmission design*	3	24000	2,96438e+006 ± 0,264829	2,96440e+006 ± 4,66e-010	7,43233e+006 ± 2,28e-009	**2,96438e+006 ± 1,40e-009**
P_2	*Optimal capacity of gas production facilities*	2	16000	1,69844e+002 ± 0,000021	**1,69844e+002 ± 1,14e-013**	**1,69844e+002 ± 1,14e-013**	**1,69844e+002 ± 1,14e-013**
P_3	*Design of a gear train*	4	32000	1,76382e-008 ± 3,51577e-008	1,43626e-009 ± 5,05e-009	1,40108e-010 ± 3,35e-010	**1,39732e-010 ± 2,65e-010**
P_4	*Optimal thermohydraulic performance of an artificially roughened air heater*	3	24000	4,21422 ± 5,08471e-07	**7,26733e-016 ± 5,69e-016**	2,31987e-006 ± 1,25e-005	3,17128e-005 ± 7,54e-005
P_5	*Frequency modulation sound parameter identification*	6	144000	3,01253 ± 0,367899	9,75177e+000 ± 6,65e+000	2,58539e+000 ± 3,30e+000	**2,07431e+000 ± 3,07e+000**
P_{6a}	*The spread spectrum radar poly − phase code design* (a)	10	240000	0,626379 ± 0,0821391	5,00750e-001 ± 1,61e-001	3,50806e-001 ± 7,19e-002	**3,00490e-001 ± 7,07e-002**
P_{6b}	*The spread spectrum radar poly − phase code design* (b)	20	480000	1,07813 ± 0,0812955	8,65976e-001 ± 2,52e-001	5,39793e-001 ± 1,25e-001	**5,37990e-001 ± 8,25e-002**

TABLEAU 2.11: Les résultats obtenus pour des problèmes réels.

2.4 Conclusion

Deux nouveaux algorithmes d'optimisation, nommés *PSO-2S* et *DEPSO-2S*, ont été présentés. Le premier algorithme est basé sur l'utilisation de l'approche multi-essaims et considère les particules comme des électrons chargés dans un espace de recherche partitionné. Trois idées ont été ajoutées à l'algorithme standard de PSO, sans introduire d'opérations complexes. A partir de l'analyse et de l'expérimentation faites, nous constatons que *PSO-2S* surpasse les autres variantes pour la plupart des problèmes testés. Nous pouvons conclure que les stratégies utilisées dans cette approche ont rendu PSO plus robuste, et que les performances ont été améliorées de manière significative. Le deuxième algorithme est une variante de *PSO-2S*, basée sur l'hybridation de PSO et DE. DE a été intégré dans *PSO-2S* pour construire l'essaim principal et donc le changement est effectué uniquement dans la première phase, la suite de l'algorithme restant globalement similaire à *PSO-2S*. Cet algorithme obtient aussi de bons résultats, l'hybridation des deux algorithmes a contribué à

améliorer l'ancienne version de *PSO-2S*.

Nous soulignons que ces nouvelles idées proposées dans ce chapitre aboutissent à de nouvelles méthodes qui pourraient être appliquées à d'autres algorithmes d'optimisation, tels que les algorithmes évolutionnaires.

Dans les chapitres suivants, nous nous intéressons à la topologie de voisinage dans PSO et à l'application de *PSO-2S* à des problèmes réels relevant du traitement d'images, qui est une thématique de recherche prépondérante du laboratoire LiSSi.

CHAPITRE TROIS

TOPOLOGIE DE VOISINAGE DANS PSO : ÉTUDE ET PERFECTIONNEMENT

3.1 Introduction

Nous avons expliqué précédemment que PSO est un algorithme inspiré d'un comportement social, comme celui des individus d'un essaim d'oiseaux ou d'un banc de poissons. Ce comportement a été modélisé par les auteurs de PSO en deux équations (1.3.1) et (1.3.2), déjà détaillées au premier chapitre. À partir de ces équations, nous constatons que le déplacement d'une particule est influencé par trois composantes : la composante d'inertie, la composante cognitive et la composante sociale. Chacune de ces composantes représente une partie de l'équation (1.3.1). Dans ce chapitre, nous allons nous intéresser à la composante sociale (i.e. la troisième partie) de cette équation, où nous utilisons le terme *gbest* ou *lbest*, selon la version utilisée.

L'équation (1.3.1) montre que les relations entre les particules influencent directement leurs vitesses et donc leurs déplacements. Dans la littérature de PSO, il existe plusieurs versions qui utilisent la notion de voisinage d'une particule et ce de manières différentes. Certaines se servent de la version globale, c'est-à-dire que chaque particule est connectée à toutes les autres particules de l'essaim (i.e. le voisinage d'une particule est l'ensemble des particules de l'essaim) ; d'autres utilisent la version locale, où chaque particule n'est connectée qu'à une partie des autres particules de l'essaim (i.e. le voisinage d'une particule est un sous-ensemble des particules de l'essaim). Le voisinage d'une particule i peut donc être défini comme le sous-ensemble de particules de l'essaim que i peut interroger afin de récupérer ses informations. Pour intégrer ces notions dans l'équation (1.3.1), nous avons besoin de définir une structure de voisinage entre les particules, qui forme le réseau de communication entre elles. Cette structure est appelée "topologie de voisinage". Les auteurs ont proposé deux types de topologies : les topologies statiques et dynamiques. Chaque type étant plus performant pour certains problèmes que pour d'autres. Pour avoir une meilleure performance de l'algorithme, il est très important de bien définir la topologie

de voisinage, ainsi que les modalités d'échange d'informations entre particules. C'est pourquoi, pour accroître les performances de PSO, nous avons élaboré une nouvelle topologie dynamique, que nous avons nommée *Dcluster*, qui est une combinaison d'une topologie statique (*Four-clusters*) [Mend 04] et d'une topologie dynamique (*Fitness*) [Wang 08].

Dans ce chapitre, nous nous concentrons d'abord sur le concept de voisinage, en étudiant les caractéristiques des échanges d'informations entre les particules. Ensuite, nous procédons à une description détaillée de la topologie proposée (*Dcluster*). Enfin, nous présentons une analyse expérimentale, ainsi qu'une comparaison des performances de *Dcluster* à celles d'autres algorithmes utilisant différentes topologies connues dans la littérature de PSO.

3.2 Notion de voisinage de PSO

Comme nous l'avons vu dans le chapitre 1, l'algorithme PSO est inspiré du comportement collectif des essaims lors des études et des simulations informatiques de groupes d'animaux (vols d'oiseaux ou bancs de poissons). Ces études mettent en évidence la capacité d'un individu à rester à une distance optimale par rapport aux autres dans le même groupe et à suivre un mouvement global affecté par les mouvements locaux de ses voisins. À partir de ces simulations, les auteurs ont modélisé le comportement des individus par les équations (1.3.1) et (1.3.2). Dans la pratique, en utilisant l'équation (1.3.1), un réseau doit être défini afin d'établir des connexions entre les particules et leur permettre d'échanger des informations entre elles. Ce réseau de communication entre les particules est appelé *topologie de voisinage*. Cette topologie contribue à définir un groupe d'informatrices pour chaque particule, c'est ce que l'on appelle le voisinage d'une particule. Le voisinage d'une particule peut donc être défini comme le sous-ensemble de particules de l'essaim avec lesquelles celle-ci a une communication directe, i.e. chaque particule peut interroger les particules dans son voisinage (ses informatrices) qui, à leur tour, lui transmettent leurs informations.

Dans notre description de PSO dans le chapitre 1, nous avons décrit brièvement la version globale de PSO, dans laquelle toutes les particules ont accès à l'information globale *gbest* de la population. Toutefois, ce n'est pas le seul choix possible, et il existe des essaims dans lesquels les particules ont accès à des informations locales, ou à des informations à la fois globales et locales. Dans ce chapitre, nous discutons des différentes possibilités

d'interaction des particules entre elles, qui sont modélisées au niveau de la composante sociale de l'équation (1.3.1).

En 1995, lorsque le premier algorithme PSO (la version globale) a été proposé, les auteurs ont choisi une topologie entièrement connectée, c'est-à-dire que chaque particule se trouve reliée à toute les autres particules de l'essaim. Dans cette topologie, l'information utilisée dans l'équation (1.3.1) est globale, i.e. chaque particule vole dans l'espace de recherche du problème avec une vitesse adaptative, qui se modifie dynamiquement, selon sa propre expérience du vol et l'expérience du vol de toutes les autres particules dans l'essaim. La faiblesse de cette version est qu'elle ne fournit pas une exploration suffisante (recherche globale) de l'espace de recherche et peut conduire à une stagnation dans un optimum local et donc à une convergence prématurée [Shi 99]. De nombreuses variantes de la version originale ont alors été proposées afin d'améliorer sa convergence. Ces variantes intègrent notamment de nouvelles topologies pour l'essaim de particules. Nous allons traiter dans la suite ces différentes topologies, telles que les topologies locales [Watt 98, Suga 99, Watt 99], les topologies dynamiques et d'autres topologies plus sophistiquées, qui ont été proposées pour établir un équilibre entre la recherche locale et la recherche globale [Kenn 99, Pera 03, Rich 03, Cler 06, Wang 08].

3.2.1 Voisinage géographique et social

La notion de voisinage peut avoir deux types principaux, géographique et social, qui donnent aux particules une capacité de perception de leur environnement proche, selon le type utilisé (i.e. l'environnement proche peut être défini par un réseau social ou un réseau géographique).

3.2.1.1 Voisinage géographique

Le voisinage géographique d'une particule i se compose des particules proches. Pour considérer cette proximité, la distance entre la particule i et les autres particules de l'essaim est utilisée, sachant que le nombre de voisins doive être défini à l'avance. Dans [Lane 08, Safa 09], les auteurs ont utilisé l'approche spatiale pour définir le voisinage géographique des particules. Les voisins sont ainsi calculés avec la triangulation de *Delaunay* ou le diagramme de *Voronoï*. Un autre modèle d'extension spatiale des particules a été proposé dans [Krin 02] pour augmenter la diversité des particules, lorsqu'elles commencent à se regrouper. Pour remédier au problème du regroupement, un rayon r a

été ajouté à chaque particule afin de vérifier s'il y a rapprochement entre particules puis collision. Dans ce cas, les auteurs proposent plusieurs techniques pour écarter les particules et éviter la collision et donc le regroupement. Toutefois, le voisinage doit être calculé à chaque itération car les distances entre les particules peuvent changer d'une itération à l'autre au cours de la recherche. Cette méthode est donc couteuse en termes de temps de calcul, mais elle permet par ailleurs d'apporter un certain dynamisme à la topologie.

3.2.1.2 Voisinage social

Le voisinage social d'une particule est le sous-ensemble des voisins virtuels défini au début de l'algorithme PSO ; il n'est pas nécessairement fixe et peut être adaptatif. Il peut être représenté sous la forme d'un graphe, où les particules sont des sommets reliés à leurs voisines par les arêtes du graphe. Le but d'utiliser ce type de voisinage est d'améliorer la rapidité de convergence de l'algorithme. Cela signifie qu'il est nécessaire de contrôler la propagation de l'information dans le graphe. Pour ce faire, il s'agit donc de trouver quels sont les bons paramètres (par exemple, la taille du voisinage, la forme du réseau social, la ou les sources d'influence) qui structurent le graphe et qui sont responsables de la vitesse de propagation du flux d'information. Savoir choisir la topologie est donc fondamental et demande une adaptation au problème traité. Lorsque la topologie est dense, les particules ont beaucoup de voisins, de sorte que l'information (meilleure position) est rapidement partagée. Inversement, dans une topologie éparse, l'information se propage plus lentement.

Pour définir une topologie de ce type, qui peut être statique ou dynamique, on doit créer une structure au début de l'algorithme qui définit le voisinage de chaque particule (e.g. les topologies : anneau ou *Ring*, étoile ou *Wheel*, *Von Neumann*, *Four-clusters*, etc.). Il y a aussi des topologies qui n'ont pas besoin de définir leurs structures, comme *Fitness-Distance-Ratio* [Pera 03] et une autre, aléatoire, qui a été proposée par Clerc [Cler 06] et qui est utilisée dans SPSO 2006, SPSO 2007 et SPSO 2011. D'autres modèles sociaux ont été proposés par Watts [Watt 99, Watt 98], et se caractérisent par deux facteurs, le degré de connectivité K, qui mesure le nombre de voisins d'une particule, et la quantité de regroupement C, qui mesure le nombre de voisins d'une particule appartenant également à d'autres voisinages. Le voisinage social est le plus utilisé dans la littérature de PSO. Il est le plus simple à programmer et le moins couteux en temps de calcul. La convergence de l'algorithme peut conduire un voisinage social à devenir un voisinage géographique. La

figure 3.1 montre la différence entre un voisinage géographique et un voisinage social.

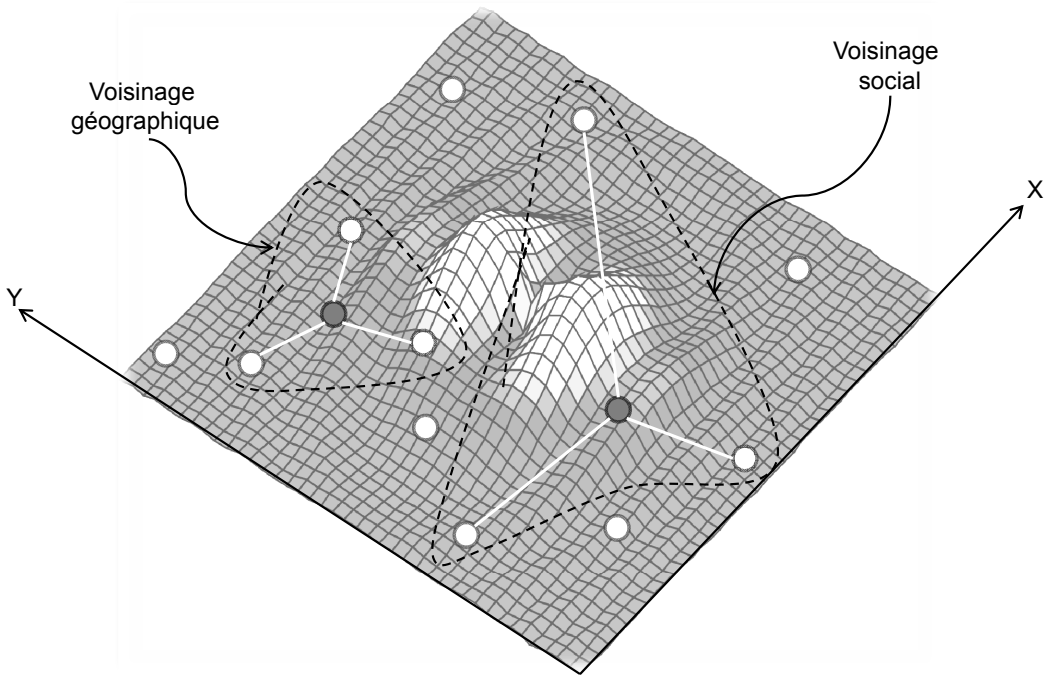


FIGURE 3.1: Voisinage géographique et voisinage social.

3.2.2 Échange des informations entre les particules dans l'algorithme PSO

Après la définition de la topologie du voisinage, chaque particule peut obtenir des informations sur une partie des particules de l'essaim ou sur toutes les particules de l'essaim, selon la topologie choisie. Pour donner une vision du mécanisme d'échange d'informations, il convient de répondre à plusieurs questions sur la coopération entre les particules.

– Quelle est l'information à échanger ? Dans la plupart des versions de PSO (globale et locale), l'information à échanger est la meilleure expérience des particules. Dans la version globale, l'information qui doit être partagée est la meilleure solution trouvée par l'essaim (*gbest*) ; pour ce qui est de la version locale, c'est la meilleure solution trouvée dans le voisinage de chaque particule (*lbest*) qui est considérée. Il y a certaines versions de PSO qui échangent à la fois l'information locale (*lbest*) et globale (*gbest*) [Pars 04], afin d'améliorer la convergence de l'algorithme et ne pas perdre l'information globale de l'essaim. Il existe également des versions qui échangent la

mauvaise expérience des particules [Chen 10], afin d'éviter l'exploration des zones non prometteuses de l'espace de recherche.

- Quand l'échanger ? Le moment d'échange de l'information dépend de la stratégie de communication utilisée. Dans la littérature, deux stratégies sont connues, les stratégies *synchrones* et les stratégies *asynchrones*, qui indiquent le moment où les particules doivent partager l'information [Vent 05]. La stratégie synchrone est la plus utilisée dans la littérature de PSO, elle oblige les particules à communiquer à chaque itération, ou suite à un nombre prédéterminé d'itérations, et ce après avoir mis à jour l'information à échanger (*gbest* ou *lbest*). La deuxième stratégie, asynchrone, implique l'échange de l'information lorsque certaines conditions se produisent (par exemple, si la solution trouvée ne s'améliore pas pendant un certain nombre d'itérations).
- Qui échange avec qui ? Une particule peut partager son information avec ses voisines, qui sont définies selon la topologie utilisée. Dans la version globale, l'information peut être transmise à toute la population de l'essaim, car le voisinage de chaque particule est l'essaim tout entier. En revanche, dans la version locale, chaque particule reçoit l'information de son voisinage, qui n'est qu'une partie de l'essaim. Il existe également des versions qui se basent sur la notion de tribus [Bast 09], où il y a une particule *leader*, qui a le droit exclusif de recevoir l'information et de la communiquer aux autres particules, après l'avoir examinée.
- Que faire de l'information échangée ? Une fois l'information est reçue, elle doit être traitée afin d'améliorer la performance de l'algorithme. L'importance de cette information est révélée par son utilisation dans l'équation (1.3.1), elle sert à mettre à jour les vitesses qui vont permettre, par la suite, de déplacer les particules en améliorant leurs positions dans l'espace de recherche. D'autre part, l'information peut être utilisée pour réinitialiser une ou plusieurs particules qui n'ont pas amélioré leurs positions durant plusieurs itérations, ou l'essaim tout entier.

3.2.3 Différents modèles de topologies

PSO a connu deux versions originales, juste après son développement en 1995, qui sont la version globale (*Gbest*) et la version locale classique en anneau (*Lbest* classique). En plus de ces deux modèles originaux, un grand nombre de topologies ont été proposées et beaucoup d'entre elles ont laissé un impact positif sur l'algorithme PSO en termes

de performances. Nous distinguons les topologies statiques des topologies dynamiques, avant de décrire, ci-dessous, une sélection de ces topologies, qui ont été citées plusieurs fois dans la littérature et ont eu un certain succès.

3.2.3.1 Topologies statiques

A l'origine, l'algorithme PSO n'utilisait pas de topologies dynamiques. Celles utilisées étaient structurées d'une manière régulière ou sur des formes géométriques. Ces topologies sont donc appelées graphes statiques, car elles sont définies une fois pour toutes. Les versions statiques les plus connues (celles qui sont principalement utilisées) sont la version globale *Gbest* et la version locale classique sous forme d'un anneau (*Lbest* classique), où chaque particule n'a que deux particules voisines. Dans le modèle *Lbest* classique, l'essaim converge plus lentement que dans *Gbest*, mais il a plus de chances de localiser l'optimum global. En général, l'utilisation du modèle *Lbest* permet de limiter les risques d'une convergence prématurée de l'algorithme. Ce qui a un effet positif sur les performances de l'algorithme, surtout pour les problèmes multimodaux. Kennedy et Mendes proposent plusieurs topologies statiques dans [Kenn 02, Kenn 99, Mend 04], nous présentons ci-dessous quelques-unes d' entre elles :

- *Von Neumann* [Kenn 02] : elle est communément adoptée sur les deux modèles *Gbest* et *Lbest* classique. Dans *Von Neumann*, la topologie prend la forme d'une grille (réseau rectangulaire), où chaque particule est reliée à ses quatre particules voisines (à gauche, à droite, au dessus et en dessous). Comme le modèle Lbest classique, cette topologie ralentit la vitesse de convergence de l'algorithme, ce qui est prouvé par Kennedy et Mendes dans [Kenn 02], en raison de sa structure particulière, qui ralentit la propagation de l'information à toutes les particules de l'essaim. Une représentation graphique du modèle de *Von Neumann* est illustrée dans 3.2.
- *Wheel* ou *focal* [Kenn 02] : Cette topologie a la forme d'une roue, où la particule du centre de l'essaim, nommée focale (principale), est responsable du flux des informations. Dans cette topologie, toutes les particules sont isolées les unes des autres et elles sont liées uniquement à la particule focale. Les informations doivent être communiquées à travers cette particule, qui va ensuite utiliser ces informations pour adapter sa trajectoire. Si le résultat des ajustements montre une amélioration des performances de la particule principale, alors cette amélioration sera communiquée au reste de la population. Donc la particule principale sert de filtre, qui ralentit la vi-

tesse de transmission de l'information sur la meilleure solution trouvée. La figure 3.2 illustre une représentation graphique du modèle *Wheel*.

– *Four-clusters* [Mend 04] : La topologie *Four-clusters* utilise quatre groupes de particules reliés entre eux par plusieurs passerelles. Du point de vue sociologique, cette topologie ressemble à quatre communautés isolées. Dans chaque communauté, quelques particules peuvent communiquer avec une particule d'une autre communauté. Ce modèle a été étudié dans [Mend 04, Mend 03], en prenant une taille de population égale à 20. Les particules sont divisées en quatre groupes, chaque groupe est composé de cinq particules, qui échangent l'information avec les trois autres groupes via les passerelles. Chaque passerelle n'est liée qu'à un seul groupe. Cette topologie est caractérisée par le nombre d'arêtes qui relient indirectement deux particules. Le diamètre du graphe est de trois, ce qui signifie que l'information peut être transmise d'une particule à une autre en empruntant au maximum trois arêtes. Grâce à ce diamètre, chaque particule peut obtenir une vision sur l'information globale, en effectuant au maximum trois itérations de l'algorithme. La figure 3.2 montre la représentation graphique de ce modèle. Dans [Mend 04], les auteurs ont aussi proposé un autre modèle, nommé *Pyramid*. Ce modèle est représenté par un rendu en fil de fer triangulaire en 3-D (ou *triangular wire-frame*).

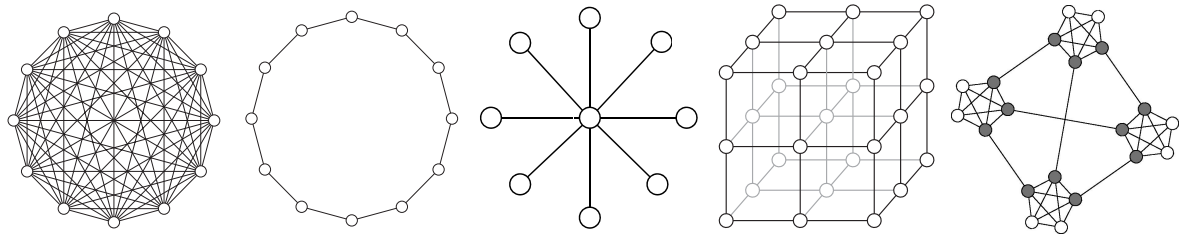


FIGURE 3.2: Topologies statiques : (a) Topologie *Gbest*, (b) Topologie en anneau (ou *Ring*) utilisée dans *Lbest*, (c) Topologie en étoile (ou *Wheel*), (d) Topologie *Von Neumann* et (e) topologie *Four-clusters*.

3.2.3.2 Topologies dynamiques

En général, les différentes topologies statiques proposées dans la littérature de PSO ne résolvent pas le problème de la convergence prématurée de PSO, même si beaucoup d'entre elles améliorent ses performances. La recherche s'est donc poursuivie, visant tou-

jours à élaborer une topologie robuste permettant d'améliorer les performances de l'algorithme. Les auteurs ont alors essayé de mettre en place des topologies dynamiques. Ils ont trouvé que ces topologies pouvaient avoir de meilleures performances que les topologies statiques, et ce, au moyen d'un changement de leurs structures, d'une itération à l'autre. Ce changement permet aux particules de changer leurs trajectoires de déplacement, afin de pouvoir s'échapper des optima locaux. Nous allons dans la suite évoquer un ensemble de topologies dynamiques :

– *Fitness* [Wang 08] : Wang et Xiang ont proposé une topologie dynamique dans laquelle les particules sont disposées dans un anneau régulier, comme dans le modèle classique en anneau du Lbest. Mais, contrairement au modèle classique qui possède des liens bidirectionnels, les particules dans *Fitness* sont connectées par des liens unidirectionnels, une particule ne peut donc communiquer qu'avec une seule autre particule située après elle. Toutefois, les particules sont triées par ordre décroissant, en fonction de leurs valeurs de $fitness$, et non pas selon leurs indices prédéfinis. En effet, la particule qui a la plus grande valeur de $fitness$ (la mauvaise particule) est placée dans la première position de l'anneau, suivie par la particule qui a la deuxième valeur, et ainsi de suite (cet arrangement a lieu en cas de minimisation). Les auteurs ont utilisé deux stratégies, « *Learn From Far and Better Ones* » et « *Centroid of Mass* », qui permettent la communication entre une particule et son voisinage. Dans cette topologie, l'ensemble du voisinage N_i d'une particule i est défini par :

$$N_i = \left\{ \begin{array}{l} \{n : n \in [i+1+\gamma, N]\}, \quad if\ i+1+\gamma \leq N \\ \\ \{n : n \in [(N\ mod\ i+1+\gamma), i-1]\}, \text{sinon} \end{array} \right\} \tag{3.2.1}$$

où N est le nombre de particules et γ est un paramètre défini par l'utilisateur.

– *Fitness-Distance-Ratio* (FDR) [Pera 03] : est une variante de PSO se basant sur un réseau social spécial, qui a été inspiré par l'observation du comportement animal. Ce réseau social n'utilise pas une structure géométrique spécifique pour effectuer la communication entre les particules. Toutefois, il utilise une nouvelle façon de communiquer, qui est modélisée par l'ajout d'un terme à la composante sociale de l'équation (1.3.1). Dans FDR, la position et la vitesse d'une particule sont mises à jour dimension par dimension. Pour chaque dimension, la particule n'a qu'une seule informatrice. On évite ainsi que plusieurs informatrices se communiquent des infor-

mations qui s'annulent entre elles. Le choix de cette informatrice, nommée *nbest* (*best nearest*), repose sur les deux critères suivants :

– elle doit être proche de la particule courante ;

– elle doit avoir visité une position ayant une meilleure $fitness$.

Pour ce faire, les auteurs ont proposé de choisir comme informatrice de la particule courante i, pour chaque dimension d, celle qui, parmi toutes les particules de l'essaim, à l'exception de la particule i, maximise le rapport (*Fitness-Distance-Ratio*) suivant :

$$\frac{Fitness(P_j) - Fitness(X_i)}{|P_{jd} - X_{id}|}$$

où P_j et X_i sont respectivement la meilleure position de la particule j et la meilleure position de la particule courante i.

Le déplacement de la particule est influencé alors par trois facteurs : sa propre expérience *pbest*, l'expérience globale de l'essaim *gbest* et l'expérience de la particule *nbest*. L'équation (1.3.1) de mise à jour de la vitesse devient la suivante :

$$v_{i,d}^{t+1} = wv_{i,d}^{t} + c_1 r_{1_{i,d}}^{t} \left[pbest_{i,d}^{t} - x_{i,d}^{t}\right] + c_2 r_{2_{i,d}}^{t} \left[gbest_{d}^{t} - x_{i,d}^{t}\right] + c_3 r_{3_{i,d}}^{t} \left[nbest_{d}^{t} - x_{i,d}^{t}\right]. \tag{3.2.2}$$

– *R*2006 [Cler 06] : C'est une topologie dynamique aléatoire qui a été proposée par Clerc. Elle consiste à utiliser la notion de modèle *Lbest*, dans lequel chaque particule est influencée par son propre voisinage défini à l'avance. Toutefois, dans *R*2006, la taille du voisinage est variable d'une particule à l'autre, elle peut être de n'importe quelle valeur entre 1 et S (où S est la taille de la population) avec une forte probabilité d'être proche d'un nombre fixe k. Chaque particule choisit son voisinage aléatoirement en utilisant une distribution représentée graphiquement par une courbe ayant la forme d'une cloche non symétrique (avec une valeur de moyenne proche de k). À chaque itération, si la meilleure solution trouvée jusque-là n'a pas été améliorée, la topologie change à l'itération suivante. La topologie *R*2006 est utilisée dans SPSO 2006, SPSO 2007 et SPSO 2011 [Cler 12].

– *Hierarchy* [Jans 05] : Janson et Middendorf ont proposé une autre topologie dynamique. Dans cette topologie, les particules sont organisées dans une hiérarchie dynamique afin de définir le voisinage de chaque particule, qui change au fil des itérations. La structure de cette topologie est un arbre descendant de nœuds où

chacune des branches contient aussi d'autres nœuds, pour pouvoir continuer à créer d'autres branches. L'arbre de cette topologie est défini par sa hauteur h, le degré de branchement d (i.e. le nombre maximum d'enfants des nœuds internes) et le nombre total de nœuds m (i.e. le nombre de particules). La figure 3.3 montre un exemple d'un essaim de 21 particules disposées dans un arbre descendant régulier de hauteur $h = 3$, et un degré de branchement $d = 3$. Dans le cas où il n'est pas possible de construire un arbre régulier avec un degré de branchement uniforme pour tous les nœuds, toute incompatibilité sur les nœuds internes sera poussée au niveau le plus profond de l'arbre, à condition que la différence de branchement entre deux nœuds de ce niveau soit égale, au plus, à un, comme dans l'exemple illustré par la figure 3.4. Les particules changent leurs places dans l'arbre, vers le haut ou vers le bas, selon leurs valeurs de $fitness$ évaluées à chaque itération. La valeur de $fitness$ d'une particule du nœud courant est comparée à celles des particules des nœuds enfants (une par une et de la gauche vers la droite). Les particules échangent leurs places à chaque fois que l'on trouve une particule d'un nœud enfant qui est meilleure que celle du nœud courant. On commence ces comparaisons par la particule en tête avec l'enfant de gauche (du haut vers le bas, et de la gauche vers la droite).

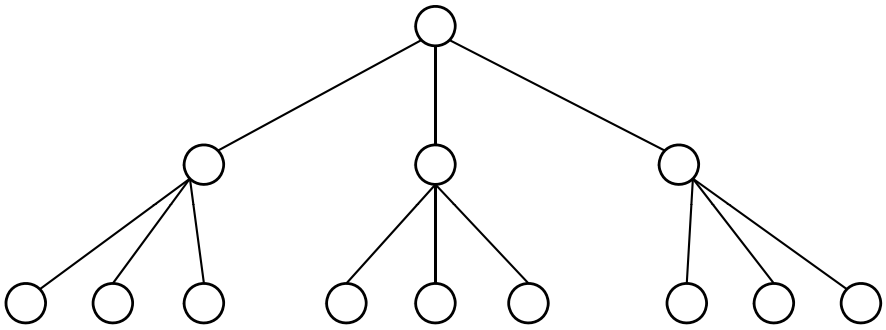


FIGURE 3.3: Un arbre régulier.

Une autre topologie dynamique a été proposée dans [Rich 03]. L'essaim est initialisé sous la forme d'une topologie en anneau. A l'initialisation, chaque particule est connectée à une autre particule unique de la population. Au fil du temps, des connexions supplémentaires entre les particules sont ajoutées, jusqu'à ce que l'on ait un réseau entièrement connecté comme dans la version globale *Gbest*. La stratégie implémentée ici est d'avoir un essaim entièrement connecté (i.e. une topologie globale) après 80% des évaluations al-

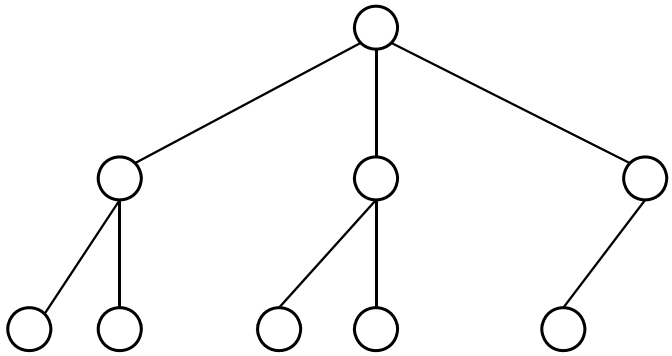


FIGURE 3.4: Un arbre non-régulier.

louées pour la fonction utilisée. Sachant qu'une nouvelle connexion est ajoutée à chaque particule après qu'un nombre prédéfini d'évaluations aient été effectuées, jusqu'à atteindre 80% du nombre total d'évaluations. Par exemple, supposons que la taille de l'essaim soit de 12 particules et que le nombre maximum d'évaluations soit égal à 9600. Dans un premier temps, chaque particule P_i est reliée à P_{i+1} (P_{11} liée à P_0). Au fil du temps, l'essaim ajoute progressivement des connexions, de telle sorte que toutes les particules deviennent interconnectées entre elles, après avoir atteint $9600 * 0{,}8 = 7680$ évaluations. Après la configuration initiale en anneau, chaque particule doit être connectée à 10 autres particules, si l'on ajoute une nouvelle connexion après chaque série de 768 évaluations (64 itérations de l'algorithme de 12 particules). Donc, après 64 itérations, chaque P_i est connecté à P_{i+2} ; après 128 itérations, chaque P_i est connecté à P_{i+3}, et ainsi de suite.

3.3 Élaboration de la topologie *Dcluster* : une combinaison de deux topologies (statique et dynamique)

Une topologie statique n'est pas appropriée pour tous les problèmes d'optimisation, et donc une topologie dynamique est parfois nécessaire. En effet, l'utilisation d'une topologie dynamique permet à l'algorithme PSO d'améliorer ses performances et de limiter les risques d'une convergence prématurée. De ce fait, nous avons élaboré une nouvelle topologie dynamique, nommée *Dcluster*. Cette topologie est une combinaison de deux topologies, l'une statique et l'autre dynamique. Dans cette section, nous décrivons la topologie *Dcluster*, puis nous en présentons une analyse expérimentale, ainsi qu'une comparaison de ses performances avec celles des autres algorithmes PSO, qui utilisent différentes topologies.

3.3.1 Présentation de la topologie proposée

La topologie proposée *Dcluster* utilise une nouvelle structure spécifique pour former son réseau de communication entre les particules. L'idée générale de cette topologie s'inspire du point de vue moral selon lequel le fort doit aider le faible, chacun devant donner suivant ses facultés, et recevoir, suivant ses besoins. Prenant par exemple une population d'individus que nous partitionnons en sous-ensembles homogènes de tribus (par exemple, en mettant ensemble les individus ayant une « résistance » similaire). Chaque sous-ensemble d'individus a son propre monde, complètement interconnecté et il peut échanger des informations avec les autres, à travers des canaux de communication définis par toutes les tribus. Ces communications permettent aux tribus de communiquer les besoins de chacune et aux faibles individus d'améliorer leurs situations. Dans certains cas, si un individu devient plus fort qu'un autre appartenant à une autre tribu, il peut migrer vers cette autre tribu plus forte, en essayant de maintenir la parité entre les individus dans chaque tribu.

Pour rapprocher l'idée de communication entre les individus des tribus avec les particules de l'essaim de PSO, nous considérons l'individu comme une particule, et la tribu comme un sous-essaim de particules. Puis l'ensemble des sous-essaims forme l'essaim principal de l'algorithme PSO.

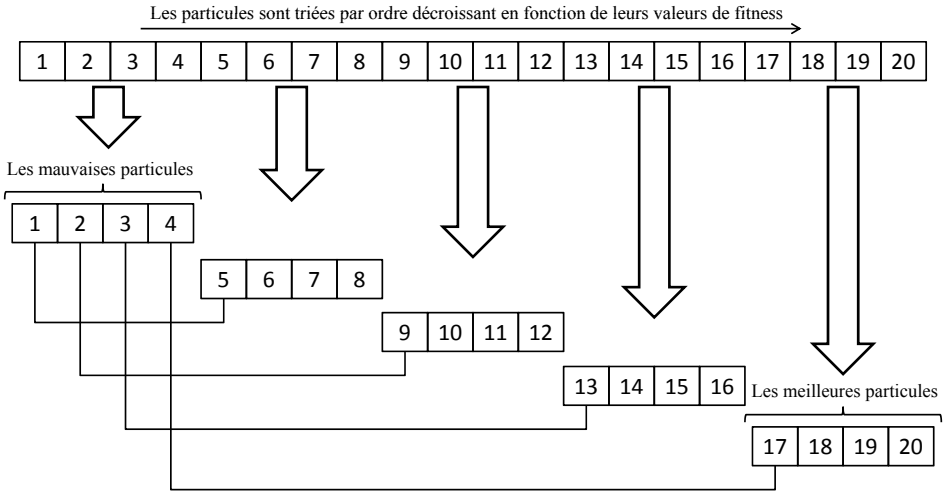


FIGURE 3.5: Partitionnement en sous-liste.

La topologie proposée fonctionne comme suit : à chaque itération, après la phase d'évaluation de la fonction objectif, on trie les particules dans une liste, suivant l'ordre décrois-

sant de leurs valeurs de $fitness$ (en cas de minimisation). De sorte que la particule qui obtient la plus grande valeur de $fitness$ (la plus mauvaise particule) est placée dans la première case de la liste (case n° 1), suivie par la particule qui obtient la deuxième plus grande valeur dans la case suivante (case n° 2), et ainsi de suite, et donc la dernière case dans la liste contient alors la meilleure particule de l'essaim. Une fois le classement des particules terminé, la liste se divise en plusieurs sous-listes, correspondant chacune à un groupe (ou *cluster*) de particules dans la topologie proposée. La dernière étape consiste à lier les groupes entre eux, afin de créer la structure finale de la topologie de l'essaim. Ainsi, le premier groupe contenant les « mauvaises » particules est appelé groupe central (ou *central cluster*), il est placé au centre de la topologie. Chaque particule de ce groupe est reliée aux autres groupes comme suit : la mauvaise particule du groupe central est liée à la mauvaise particule du deuxième groupe, la seconde est liée au troisième groupe aussi par sa mauvaise particule, et ainsi de suite. Ceci est illustré dans la figure 3.5. Tous les groupes dans cette topologie, y compris le groupe central, sont entièrement connectés. La figure 3.6 montre la structure finale de la topologie proposée.

Dcluster est définie par le nombre de particules dans chaque groupe np. Pour avoir une topologie de forme régulière, la taille de l'essaim doit être égale à $np + np^2$ et $ng = np + 1$, où ng est le nombre de groupes. Prenons, par exemple, un essaim de taille égale à 20, la topologie sera composée de 5 groupes ($ng = 5$), qui contiennent chacun 4 particules ($np = 4$).

3.3.2 Résultats et discussion

Pour tester *Dcluster*, nous allons utiliser quelques fonctions de *CEC05* et 12 problèmes réels, tels que décrits au paragraphe 1.4. Nous commençons ici par la présentation des valeurs des paramètres utilisées par *Dcluster* et par d'autres versions de PSO utilisant différentes topologies de la littérature. Ensuite, nous effectuons une analyse expérimentale de tous les algorithmes, ainsi qu'une comparaison des performances obtenues avec *Dcluster* par rapport aux autres topologies. Enfin, nous réaliserons une étude de la convergence de l'algorithme.

3.3.2.1 Paramétrage

Afin de montrer l'efficacité de notre topologie, nous comparons, dans les lignes suivantes, les performances de l'algorithme PSO, qui utilise la topologie *Dcluster*, aux

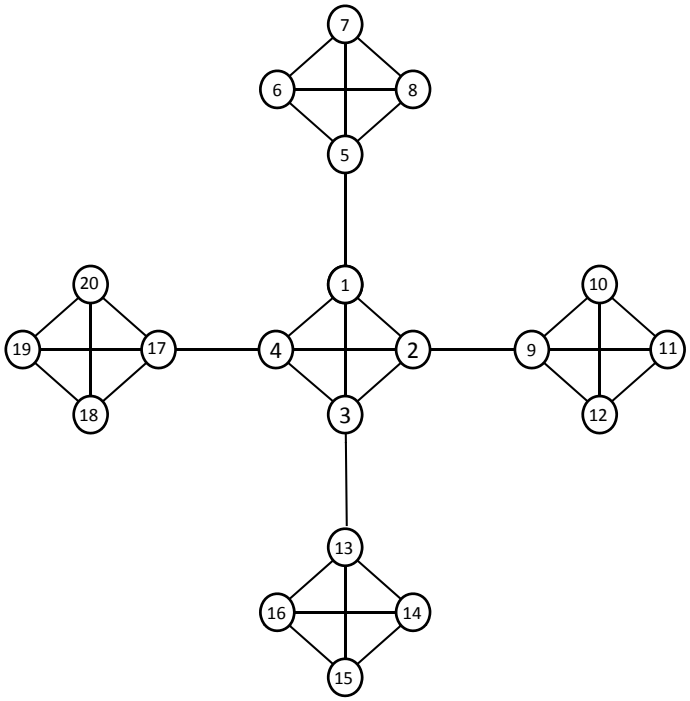


FIGURE 3.6: Structure finale de *Dcluster*.

six autres algorithmes utilisant d'autres topologies (*R*2006, *FDR*, *Fitness*, *Four-clusters*, *Geometric* et *Hierarchy*).

Pour que la comparaison soit équitable, plusieurs éléments doivent être pris en compte : le générateur de nombres aléatoires et les paramètres des algorithmes (par exemple : taille de la population, c_1, c_2, w, etc.). Dans notre travail, nous utilisons, pour les sept topologies, le même générateur de nombres aléatoires *KISS* (*Random Number Generator* (RNG)) qui est utilisé dans les algorithmes SPSO 2006, SPSO 2007 et SPSO 2011 [Cler 12]. La taille de la population est fixée à 20 pour les sept algorithmes. Les coefficients d'accélération c_1 et c_2 sont choisis égaux à 1,19 et le coefficient d'inertie w est fixé à 0,72 pour chaque algorithme. La taille du voisinage de *FDR*, *Fitness* et *Geometric* est fixé à 5. Elle est variable entre 1 et 20 pour *R*2006, avec une forte probabilité d'être proche de 5. Pour la topologie *Hierarchy*, la hauteur h est fixée à 4 et le degré de branchement d est fixé à 2. Dans ces expériences, *Dcluster* utilise une forme régulière, comme dans la figure 3.6, où le nombre de particules dans chaque groupe np est fixé à 4, alors que le nombre de groupes ng est égal à 5.

Nous utilisons tout d'abord une sélection des fonctions décalées et des fonctions pivotées (*Shifted functions* et *Rotated functions*), définies dans la section 1.4.1. Le nombre maximal d'évaluations (Max_{Fes}) est fixé à 50000 en 10-D pour les huit fonctions, et à 150000 en 30-D.

Nous utilisons ensuite les 12 problèmes réels présentés dans la section 1.4.2. Le Max_{Fes} pour chaque problème est donné dans le tableau 3.4. Dans les expériences, chaque algorithme est exécuté 100 fois et la moyenne et l'écart-type des valeurs de la fonction objectif sont calculés. Les sept algorithmes sont implémentés en langage C dans notre laboratoire, en utilisant un ordinateur équipé d'un processeur de 64-bits.

Problème	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}
Dimension	27	4	6	10	3	5	42	4	4	7	13	10
Max_{Fes}	150000	50000	150000	240000	50000	50000	150000	300000	50000	50000	15000	100000

TABLEAU 3.1: Dimension et Max_{Fes} des problèmes réels.

3.3.2.2 Analyse expérimentale et comparaison avec d'autres topologies

Dans cette section, les résultats numériques obtenus par *Dcluster* sont présentés, ainsi que des comparaisons avec d'autres algorithmes utilisant différentes topologies. L'intérêt est de montrer que la topologie proposée donne des résultats encourageants. Une discussion sur les avantages et inconvénients de *Dcluster* est ensuite engagée.

Les tableaux 3.2 et 3.3 présentent les résultats de simulation obtenus en 10-D et 30-D respectivement, sur les huit fonctions citées dans le paragraphe précédent. Plus précisément, la moyenne et l'écart-type (*Mean best value* $\pm$ *Standard deviation*) obtenus, pour chaque algorithme et pour chaque fonction, sur 100 exécutions en 10-D et 30-D, sont présentés dans ces tableaux. Nous avons ajouté une colonne indiquant la $P - value$ donnée par le test statistique de Friedman, afin de confirmer que les résultats sont significativement différents (des échantillons non appariés). Les meilleurs résultats sont indiqués en gras dans les tableaux.

De ces expériences, on peut remarquer que la topologie *Geometric* donne les résultats les moins bons sur toutes les fonctions testées, quoique cette topologie soit la plus couteuse en termes de temps de calcul. Les résultats des autres topologies sont mitigés, mais on peut constater que la topologie *Dcluster* obtient de meilleurs résultats que les autres topologies, pour la majorité des fonctions testées.

Nous commençons par analyser les résultats en 10-D. Les résultats de ces analyses sont présentés dans le tableau 3.2. Nous constatons que PSO utilisant la topologie *Dcluster* surpasse les autres algorithmes sur 3 fonctions décalées (*Shifted Rosenbrock, Shifted Griewank* et

Fonction	***R2006***	***FDR***	***Dcluster***	***Fitness***	***Four clusters***	***Geometric***	***Hierarchy***	***P-value***
Shifted Rosenbrock	2,81e+004 ± 2,79e+005	1,41e+001 ± 3,16e+001	**4,99e+000 ± 1,55e+001**	1,29e+001 ± 4,77e+001	7,44e+000 ± 3,10e+001	8,12e+003 ± 8,07e+004	2,25e+001 ± 6,27e+001	5,09e-32
Shifted Ackley	1,16e-002 ± 1,15e-001	**0,00e+000 ± 0,00e+000**	1,79e-014 ± 1,54e-014	1,16e-002 ± 1,15e-001	1,14e-001 ± 3,67e-001	1,62e+000 ± 1,48e+000	1,70e-014 ± 1,50e-014	1,55e-70
Shifted Griewank	5,92e-002 ± 3,38e-002	6,98e-002 ± 4,83e-002	**2,71e-002 ± 2,08e-002**	3,78e-002 ± 2,84e-002	7,15e-002 ± 5,02e-002	1,03e-001 ± 6,89e-002	4,84e-002 ± 4,11e-002	2,78e-30
Shifted Rastrigin	6,06e+000 ± 3,07e+000	6,09e+000 ± 5,13e+000	**2,81e+000 ± 1,62e+000**	3,83e+000 ± 1,76e+000	9,49e+000 ± 5,31e+000	1,57e+001 ± 6,94e+000	4,08e+000 ± 1,77e+000	3,35e-59
Rotated Rosenbrock	**3,55e+002 ± 1,28e+003**	5,85e+002 ± 1,80e+003	7,62e+002 ± 2,63e+003	1,53e+003 ± 4,14e+003	5,16e+002 ± 1,90e+003	2,51e+003 ± 1,69e+004	2,56e+003 ± 5,02e+003	1,32e-15
Rotated Ackley	2,33e+001 ± 8,50e-002	2,40e+001 ± 8,48e-002	**2,28e+001 ± 7,93e-002**	2,31e+001 ± 1,05e-001	2,30e+001 ± 8,93e-002	2,29e+001 ± 8,21e-002	2,33e+001 ± 1,06e-001	5,35e-19
Rotated Griewank	1,12e-001 ± 7,68e-002	2,24e-001 ± 1,45e-001	**1,04e-002 ± 8,86e-002**	8,78e-002 ± 7,79e-002	1,87e-001 ± 1,17e-001	6,86e-001 ± 7,02e-001	5,27e-002 ± 3,61e-002	6,20e-59
Rotated Rastrigin	1,14e+001 ± 5,81e+000	1,58e+001 ± 7,86e+000	**4,96e+000 ± 3,40e+000**	7,87e+000 ± 4,00e+000	1,35e+001 ± 4,99e+000	1,23e+001 ± 3,78e+000	5,61e+000 ± 4,29e+000	1,31e-38

TABLEAU 3.2: Les résultats en 10-D.

Shifted Rastrigin) et sur 3 fonctions pivotées (*Rotated Ackley, Rotated Griewank* et *Rotated Rastrigin*), mais ce n'est pas le cas pour la fonction (*Shifted Ackley*), pour laquelle *FDR* obtient des résultats significativement meilleurs, et pour la fonction (*Rotated Rosenbrock*), pour laquelle *Dcluster* est surpassée par *R*2006. Ainsi, *Dcluster* obtient de meilleurs résultats que les autres topologies pour 6 fonctions sur 8.

Fonction	***R2006***	***FDR***	***Dcluster***	***Fitness***	***Four clusters***	***Geometric***	***Hierarchy***	***P-value***
Shifted Rosenbrock	2,83e+001 ± 6,66e+001	5,92e+001 ± 7,91e+001	5,72e+001 ± 9,32e+001	8,21e+001 ± 1,37e+002	**1,50e+001 ± 3,32e+001**	6,70e+007 ± 9,17e+007	9,10e+001 ± 1,65e+002	6,95e-69
Shifted Ackley	1,73e+000 ± 1,03e+000	3,71e-001 ± 1,73e+000	**9,00e-014 ± 2,45e-014**	2,00e-001 ± 4,92e-001	2,66e+000 ± 1,64e+000	1,47e+001 ± 2,26e+000	5,98e-001 ± 1,30e+000	9,28e-104
Shifted Griewank	6,26e-002 ± 3,14e-001	3,93e-001 ± 2,68e+000	**4,43e-003 ± 9,38e-003**	1,50e-002 ± 2,18e-002	1,10e-001 ± 3,44e-001	1,39e+001 ± 1,12e+001	4,87e-002 ± 8,43e-002	1,80e-58
Shifted Rastrigin	5,65e+001 ± 1,85e+001	4,36e+001 ± 1,74e+001	**3,20e+001 ± 1,14e+001**	4,51e+001 ± 1,57e+001	9,49e+001 ± 2,53e+001	1,55e+002 ± 3,15e+001	4,12e+001 ± 1,59e+001	6,09e-85
Rotated Rosenbrock	1,87e+003 ± 4,62e+003	2,03e+003 ± 4,82e+003	2,13e+003 ± 4,72e+003	1,82e+005 ± 1,78e+006	**1,32e+003 ± 3,88e+003**	5,67e+008 ± 5,69e+008	3,07e+003 ± 6,36e+003	2,73e-48
Rotated Ackley	2,92e+001 ± 7,09e-002	2,96e+001 ± 5,23e-002	**2,83e+001 ± 5,98e-002**	2,91e+001 ± 7,85e-002	2,90e+001 ± 6,92e-002	2,90e+001 ± 8,23e-002	2,92e+001 ± 6,85e-002	1,51e-28
Rotated Griewank	2,11e-002 ± 1,65e-002	1,64e-002 ± 1,44e-002	**1,04e-002 ± 1,49e-002**	1,63e-002 ± 1,36e-002	2,37e-002 ± 2,20e-002	2,50e+001 ± 1,50e+001	1,95e-002 ± 1,89e-002	2,07e-48
Rotated Rastrigin	7,82e+001 ± 4,69e+001	9,29e+001 ± 2,78e+001	1,03e+002 ± 2,08e+001	5,66e+001 ± 3,80e+001	7,21e+001 ± 2,01e+001	1,09e+002 ± 2,44e+001	**3,63e+001 ± 2,53e+001**	3,54e-63

TABLEAU 3.3: Les résultats en 30-D.

Nous analysons maintenant les résultats du tableau 3.3, obtenus en 30-D. Ce tableau indique les erreurs moyennes et les écarts-types sur les mêmes fonctions que celles utilisées en 10-D. Bien que l'on ait augmenté la dimension et donc la complexité du problème, les performances de PSO, en utilisant la topologie *Dcluster*, ne sont pas affectées par ce changement. Cette topologie permet ainsi à PSO d'obtenir des bons résultats, quelle que

soit la dimension. Nous pouvons remarquer que les résultats obtenus par *Dcluster* en 30-D ne sont pas significativement différents de ceux obtenus en 10-D, où *Dcluster* surpasse les autres topologies pour cinq fonctions sur huit. *Four-clusters* possède de meilleures performances pour les fonctions *Shifted Rosenbrock* et *Rotated Rosenbrock*, et *Hierarchy* pour la fonction *Rotated Rastrigin*. Dans le tableau 3.3, nous avons ajouté une colonne qui contient les $P - value$ obtenues par le test statistique de Friedman. Ces valeurs montrent que les résultats obtenus par tous les algorithmes sont significativement différents. Pour assurer les performances et la robustesse de la topologie *Dcluster*, nous évaluons les sept topologies sur 12 problèmes réels (présentés dans la section 1.4.2) de différentes dimensions. Les résultats de ces problèmes sont présentés dans le tableau 3.4, et ils sont calculés de la même façon que pour les fonctions en 10-D et en 30-D. Les $P - value$ obtenues par le test statistique de Friedman sont également données dans ce tableau.

Pour les problèmes avec contraintes, nous avons utilisé l'approche de pénalité statique, afin de préserver la faisabilité des solutions [Homa 94]. Nous avons défini plusieurs niveaux de violation, chacun ayant un coefficient de pénalité choisi, de façon que ce coefficient augmente avec le niveau de violation. Ainsi les coefficients les plus grands sont attribués aux niveaux les plus élevés. Cette approche commence par une population aléatoire de particules (faisables ou infaisables). La mise à jour des meilleures positions des particules est réalisée en appliquant cette approche, dans laquelle les facteurs de pénalité ne dépendent pas du nombre de générations effectuées et restent constants pendant toute la recherche. Plus précisément, la particule modifie sa position courante uniquement si la nouvelle position candidate est meilleure, sinon elle ne se déplace pas. Les positions des particules sont évaluées en utilisant la formule suivante, proposée dans [Homa 94] :

$$fitness(\vec{x}) = f(\vec{x}) + \sum_{i=1}^{m} \left(R_{k,i} * max[0, g_i(\vec{x})]^2 \right) \tag{3.3.1}$$

où $R_{k,i}$ sont les coefficients de pénalité utilisés, m est le nombre de contraintes, $f(\vec{x})$ est la fonction objectif, et $k = 1, 2, \ldots, l$ où l est le nombre de niveaux de violation défini par l'utilisateur.

Nous constatons dans le tableau 3.4 que les meilleurs résultats sont obtenus par l'algorithme de PSO utilisant la topologie *Dcluster*. En effet, *Dcluster* surpasse les autres topologies pour la majorité des problèmes testés, à l'exception de F_1 et F_2. Les résultats obtenus par les autres topologies sont mitigés, et nous ne pouvons pas déterminer quelle topologie est la meilleure. La topologie *Hierarchy* surpasse la topologie proposée pour F_1 et F_2, et elle

obtient les mêmes résultats pour F_9 et F_{10}.

Pour faciliter la lecture du tableau 3.4, nous avons ajouté la figure 3.7 qui montre clairement les classements des topologies sur les 12 problèmes. Sur cette figure, l'axe y correspond aux problèmes testés, l'axe x correspond aux classements en fonction de l'erreur moyenne de chaque topologie. On peut remarquer que *Dcluster* est classée première et obtient les meilleurs résultats pour la majorité des problèmes (10 problèmes sur 12). En revanche, les autres topologies sont instables et leurs classements changent d'un problème à l'autre, c'est ce qui nous amène à les caractériser comme mitigées.

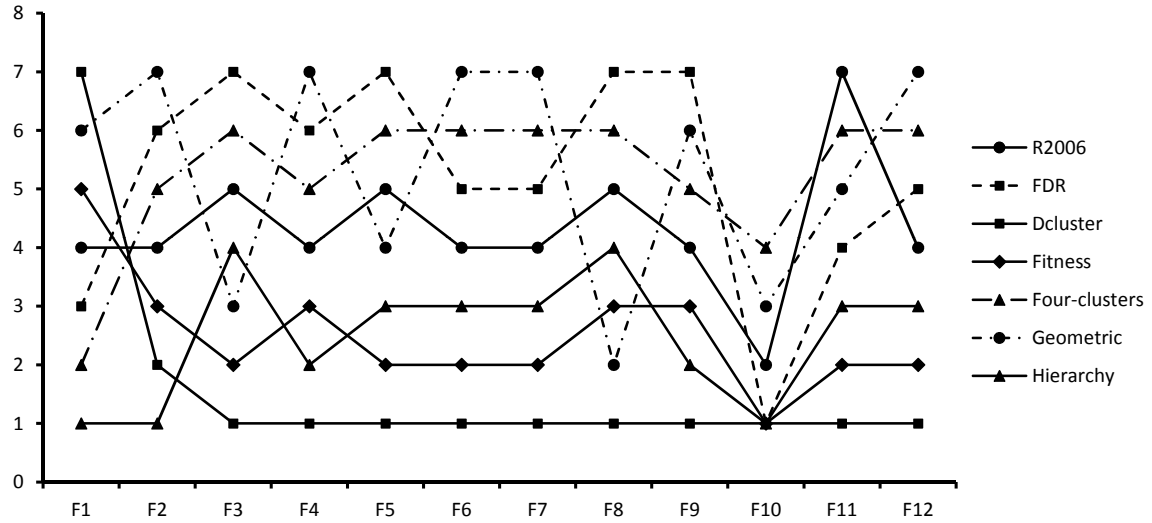


FIGURE 3.7: Le classement des sept algorithmes sur les 12 problèmes réels.

3.3.2.3 Analyse de convergence

La figure 3.8 montre les graphes de convergence pour les fonctions *Shifted Ackley*, *Shifted Griewank*, *Shifted Rastrigin* et *Shifted Rosenbrock*. Ces graphes représentent l'erreur médiane sur 100 exécutions, en fonction du nombre d'évaluations de la fonction objectif (Max_{Fes}).

Les quatre premiers graphes représentent les graphes de convergence des fonctions citées ci-dessus, en 10-D. Ces graphes montrent que l'algorithme PSO utilisant *Dcluster* obtient de meilleurs résultats et évite la convergence prématurée. *Dcluster* continue à améliorer sa meilleure solution jusqu'à la fin de l'exécution de l'algorithme, excepté pour *Shifted Ackley*. Pour cette fonction, *Dcluster* a pu limiter la convergence prématurée de PSO. Par contre, ce graphe montre qu'en utilisant les topologies (*Fitness*, *Four-clusters*, *R*2006 et *Geometric*), l'erreur décroît assez rapidement jusqu'à approximativement 1000

Pb.	***R2006***	***FDR***	***Dcluster***	***Fitness***	***Four clusters***	***Geometric***	***Hierarchy***	***P-value***
F_1	1,26e+000 ± 1,26e+000	1,22e+000 ± 7,07e-001	6,63e+000 ± 1,83e+000	1,49e+000 ± 1,57e+000	1,08e+000 ± 6,98e-001	5,74e+000 ± 1,97e+000	**1,00e+000 ± 6,19e-001**	3,41e-76
F_2	3,61e-010 ± 7,79e-010	3,36e-009 ± 5,78e-009	2,89e-010 ± 6,27e-010	3,17e-010 ± 6,01e-010	9,20e-010 ± 2,87e-009	4,37e-009 ± 7,69e-009	**2,56e-010 ± 4,91e-010**	1,17e-46
F_3	8,89e+000 ± 6,36e+000	1,18e+001 ± 7,80e+000	**2,21e+000 ± 3,98e+000**	4,50e+000 ± 4,97e+000	1,14e+001 ± 6,73e+000	5,74e+000 ± 1,97e+000	6,53e+000 ± 6,14e+000	1,59e-27
F_4	4,69e-001 ± 1,64e-001	6,45e-001 ± 1,40e-001	**3,90e-001 ± 1,22e-001**	4,23e-001 ± 1,41e-001	6,34e-001 ± 1,26e-001	7,21e-001 ± 9,03e-002	4,17e-001 ± 1,62e-001	1,12e-58
F_5	8,54e-003 ± 2,99e-002	7,28e-002 ± 1,17e-001	**6,53e-005 ± 6,50e-004**	2,67e-004 ± 1,28e-003	2,21e-002 ± 5,36e-002	5,29e-003 ± 3,23e-002	2,34e-003 ± 1,19e-002	1,70e-44
F_6	5,29e+002 ± 5,03e+002	5,30e+002 ± 5,29e+002	**1,14e+002 ± 1,96e+002**	1,62e+002 ± 2,41e+002	5,99e+002 ± 5,49e+002	6,69e+002 ± 5,66e+002	1,98e+002 ± 2,92e+002	4,46e-36
F_7	1,04e+002 ± 1,37e+001	1,07e+002 ± 2,35e+001	**9,06e+001 ± 1,02e+001**	9,49e+001 ± 1,23e+001	1,09e+002 ± 1,81e+001	1,12e+002 ± 2,90e+001	9,79e+001 ± 9,65e+000	4,03e-33
F_8	6,68e+000 ± 2,68e+001	3,35e+001 ± 7,49e+001	**3,59e-001 ± 3,58e+000**	3,03e+000 ± 1,90e+001	9,92e+000 ± 4,25e+001	3,86e-001 ± 3,84e-001	5,37e+000 ± 2,30e+001	1,09e-40
F_9	1,15e-002 ± 5,52e-002	1,61e-001 ± 2,59e-001	**3,08e-007 ± 6,48e-017**	5,51e-004 ± 5,25e-003	6,98e-002 ± 1,30e-001	1,00e-001 ± 1,33e-001	**3,08e-007 ± 3,79e-017**	1,87e-90
F_{10}	3,92e-001 ± 3,91e+000	**0,00e+000 ± 0,00e+000**	**0,00e+000 ± 0,00e+000**	**0,00e+000 ± 0,00e+000**	1,08e+000 ± 5,71e+000	3,92e-001 ± 3,91e+000	**0,00e+000 ± 0,00e+000**	1,19e-12
F_{11}	6,17e+000 ± 2,28e+000	9,78e-001 ± 1,24e+000	**2,00e-002 ± 1,99e-001**	3,57e-001 ± 8,29e-001	6,06e+000 ± 2,55e+000	5,06e+000 ± 2,15e+000	5,97e-001 ± 9,68e-001	5,11e-99
F_{12}	7,06e-001 ± 6,83e-001	1,09e+000 ± 1,12e+000	**3,26e-001 ± 2,92e-001**	4,11e-001 ± 3,86e-001	1,21e+000 ± 1,12e+000	7,43e+000 ± 8,97e+000	5,93e-001 ± 5,69e-001	4,06e-55

TABLEAU 3.4: Les résultats obtenus pour des problèmes réels.

évaluations de la fonction objectif, mais qu'ensuite elle a tendance à ne plus trop évoluer. On peut constater que la topologie *Geometric* conduit l'algorithme à une convergence prématurée, sur toutes les fonctions, et donc à de mauvais résultats.

Les quatre graphes restants sont les graphes de convergence des mêmes fonctions étudiées non pas en 10-D, mais en 30-D. Ces graphes, comme ceux présentés plus haut, montrent une amélioration significative de la convergence de PSO utilisant *Dcluster*. Pour les fonctions *Shifted Ackley* et *Shifted Griewank*, toutes les topologies convergent rapidement vers un optimum local, sauf la topologie *Dcluster*, qui converge plus lentement, ce qui permet aux particules de ne pas être piégées précocement dans un optimum local.

La bonne convergence de *Dcluster* peut être expliquée par l'utilisation de cinq groupes interconnectés dans une structure dynamique. En effet, chaque ensemble de particules (groupe) peut découvrir une région de l'espace de recherche ; ainsi la topologie contribue à accélérer la convergence vers l'optimum global, et permet aux particules piégées dans un optimum local de s'en échapper.

3.4 Conclusion

Dans ce chapitre, nous avons défini le voisinage d'une particule dans l'algorithme PSO. Nous avons également répondu aux principales questions sur les échanges des informations entre les particules. Nous avons présenté ensuite plusieurs types de topologies, utilisés dans la littérature de PSO, tels que les modèles statiques et dynamiques.

Par ailleurs, nous avons proposé une nouvelle topologie dynamique, appelée *Dcluster*, basée sur le comportement entre les individus des tribus. Cette topologie a été structurée comme une combinaison d'une topologie statique et d'une topologie dynamique (*Fousclusters* et *Fitness*). Notre travail a été évalué en utilisant différents problèmes (plusieurs fonctions de CEC'05, et problèmes réels, avec et sans contraintes). Les résultats obtenus par *Dcluster* ont été comparés à ceux obtenus par six autres topologies. Les expériences ont montré qu'aucune des topologies étudiées ne surpasse complètement les autres, mais la topologie proposée obtient les meilleures performances pour la majorité des problèmes testés. En outre, nous avons montré que *Dcluster* a pu limiter le phénomène de convergence prématurée de l'algorithme PSO.

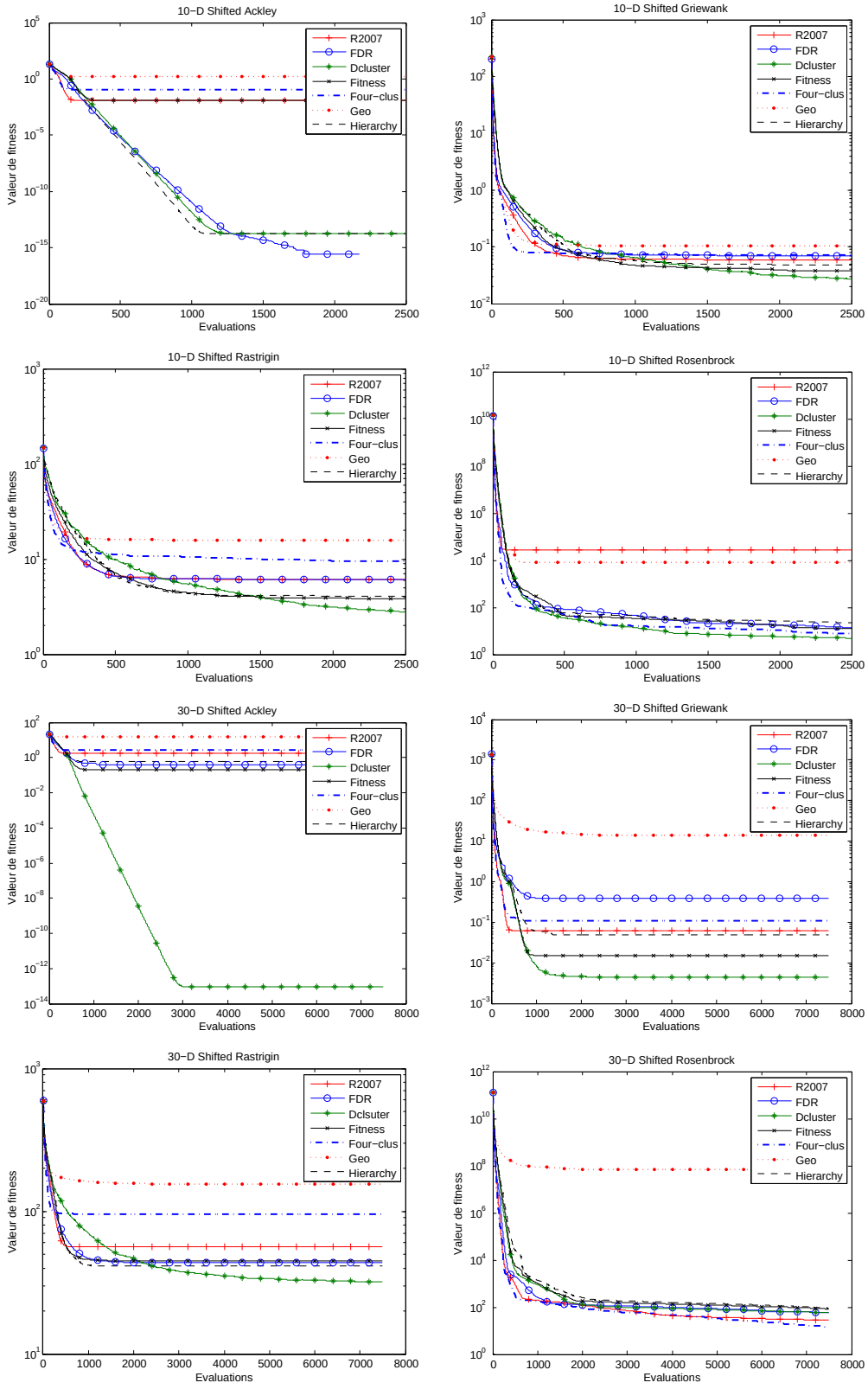


FIGURE 3.8: Les courbes de convergence de l'algorithme pour certains problèmes.

CHAPITRE QUATRE

APPLICATION DE PSO-2S EN SEGMENTATION D'IMAGES ET EN ÉLECTRONIQUE

4.1 Introduction

Ce chapitre se divise en deux parties correspondant aux applications de *PSO-2S* à des problèmes « réels ». Dans la première partie (section 4.2), nous présentons la première application de *PSO-2S* en segmentation d'images, dans laquelle *PSO-2S* est utilisé comme méthode d'optimisation pour minimiser un critère qui permet de déterminer les différents seuils d'une image multi-classes. Dans la deuxième partie (section 4.3), nous présentons la deuxième application de *PSO-2S* en électronique (plus précisément, dans le cadre de la conception de composants des circuits électroniques). Le but ici est de trouver les dimensions des transistors qui permettent de faire fonctionner de manière optimale les circuits étudiés.

4.2 Application de PSO-2S en segmentation d'images

4.2.1 Introduction

Le traitement d'images numériques est devenu prépondérant dans de nombreux domaines, tels que la surveillance, la médecine ou encore la prospection pétrolière. Les images pouvant contenir des informations très diverses, difficiles à extraire et à séparer visuellement, il est devenu nécessaire de disposer d'algorithmes de traitement automatique des images. Un système de traitement automatique d'images est globalement composé de deux niveaux : le premier est consacré au traitement numérique, au sens large, avec des opérations telles que le codage, l'amélioration ou encore la restauration ; le deuxième est dédié à l'analyse des images, dans le but d'extraire et d'interpréter l'information contenue dans l'image.

La segmentation occupe une place importante, car elle est située à l'articulation entre le

traitement et l'analyse des images. La complexité du problème, la diversité des images et l'évaluation assez empirique des résultats en font un axe de recherche très actif. Le but de la segmentation est de partitionner une image en plusieurs régions homogènes, au sens d'un critère fixé a priori. On peut ainsi passer d'une information de type « intensité lumineuse » à une information spatiale, permettant de mieux interpréter l'image traitée. D'un point de vue purement pratique, l'opération de segmentation revient à donner à chaque pixel de l'image un label d'appartenance à une région donnée.

Plusieurs approches de segmentation d'images existent dans la littérature [Naki 07]. Dans le cadre de nos travaux, nous nous sommes limités à l'approche par seuillage d'images. Le seuillage d'image est une méthode de segmentation supervisée, c'est-à-dire que le nombre de régions et leurs propriétés sont fournis au préalable par l'utilisateur. La segmentation est effectuée en déterminant, pour chaque pixel, la classe dont les propriétés se rapprochent le plus de celles observées en ce pixel. La technique de seuillage est basée sur l'hypothèse que les différentes régions de l'image peuvent être différenciées par leurs niveaux de gris. Cette méthode repose donc sur l'utilisation de l'histogramme de l'image traitée. Le seuillage de l'image en N classes revient à trouver les $N-1$ seuils qui vont partitionner l'histogramme en N zones. Le problème de seuillage est défini plus précisément dans la sous-section 4.2.2.

Dans cette partie de ce chapitre, nous montrons l'intérêt d'utiliser la métaheuristique *PSO-2S*, décrite dans le chapitre 2, pour la segmentation d'images IRM cérébrale. Les performances de *PSO-2S* sont tout d'abord comparées à celles de SPSO-07 (*Standard Particle Swarm Optimization* dans sa version 2007) [Cler 12], en utilisant des images de référence, communément utilisées dans la littérature en segmentation. Ensuite, les performances de ces deux algorithmes seront comparées sur des images issues d'une base de données générée par simulation d'IRM cérébrale [Bran 12, Coll 98]. Cette base de données, appelée *BrainWeb*, fournit des images pour lesquelles une segmentation « optimale » est connue. En effet, les simulations d'IRM cérébrale disponibles sur BrainWeb se basent sur un modèle anatomique prédéfini du cerveau. Les images générées par ces simulations peuvent ainsi servir pour valider une méthode de segmentation, ou comparer les performances de différentes méthodes.

Il est à noter ici que cette application est faite en collaboration avec Julien Lepagnot, Maître de Conférences à l'Université de Haute-Alsace, à Mulhouse.

4.2.2 Segmentation d'images par seuillage

4.2.2.1 Formulation du problème

Le problème du seuillage consiste à diviser l'image en N régions (N défini au préalable par l'utilisateur) non forcément connexes. Le processus de segmentation est uniquement basé sur l'hypothèse que les différentes régions de l'espace sont différentiables par leurs niveaux de gris. Segmenter l'image en N régions revient donc à trouver $N-1$ seuils qui vont diviser l'histogramme en N régions.

Soit f l'image à segmenter, l'image segmentée g est définie par :

$$g(x,y) = k \text{ si } T_k \leq f(x,y) < T_{k+1}, k \in \{0,...,N-1\} \tag{4.2.1}$$

où x et y sont les coordonnées du pixel courant, N est le nombre de classes et $T_0, ..., T_N$ sont les bornes des différentes régions en niveaux de gris. Le but d'une méthode de seuillage est de trouver les seuils optimaux $T_1, ..., T_{N-1}$, T_0 et T_N étant les bornes de l'ensemble des niveaux de gris possibles.

Le seuillage manuel d'images se déroule globalement en quatre étapes :

1. observation de l'histogramme ;
2. choix des seuils dans les vallées de l'histogramme ;
3. définition des classes de régions par intervalles de niveaux de gris ;
4. classement des pixels.

Les performances du seuillage manuel dépendent de l'aptitude de l'opérateur à trouver les bons seuils de segmentation. Cette tâche ne pose pas de problème lorsque l'histogramme de l'image n'est pas difficile à interpréter. Toutefois, ce n'est pas toujours le cas avec les différentes classes d'images. Le but d'un algorithme de seuillage est donc de proposer une méthode automatique qui permette à l'utilisateur de s'affranchir de l'ensemble des quatre étapes du seuillage manuel, afin de gagner en précision et en rapidité de traitement.

4.2.2.2 Méthodes de seuillage

Les méthodes de seuillage peuvent être divisées en deux catégories : les méthodes *non-paramétriques*, qui reposent sur l'optimisation d'un ou plusieurs critères a posteriori, et les méthodes *paramétriques*, qui sont basées sur l'hypothèse que les niveaux de gris des différentes classes de l'image suivent une certaine fonction de densité de probabilité.

Les deux méthodes non-paramétriques de référence sont la méthode d'Otsu [Otsu 79] et la méthode de Kapur et al. [Kapu 85]. Beaucoup de techniques parues ensuite sont des variantes de ces deux méthodes. La méthode d'Otsu repose sur la maximisation de la *variance interclasse*. Celle de Kapur et al. est basée sur la maximisation de l'entropie de Shannon. Cette méthode a ouvert le champ à la définition de nombreuses mesures d'information pour mieux segmenter les images, telles que l'entropie de Tsallis ou l'entropie de Renyi [Sezg 04].

Les méthodes paramétriques de seuillage supposent que les différentes classes de l'image suivent une *fonction de densité de probabilité* (fdp) prédéfinie. L'histogramme est alors approché par une somme de fdp et les seuils sont choisis aux intersections de celles-ci. Généralement, ces fdp sont supposées suivre un modèle gaussien. Le problème se décompose ici en deux étapes : estimer les paramètres des fdp et déterminer les seuils de segmentation. La plupart des méthodes existantes sont présentées dans [Saho 88], [Sezg 04], [Gonz 07] et [Cuev 10].

Dans cette section, nous nous intéressons à une méthode de cette catégorie pour segmenter des images médicales. L'histogramme est ainsi approché par une somme de fonctions gaussiennes. Une telle opération nous permet d'obtenir une expression analytique approchée de l'histogramme, et d'en déduire les seuils optimaux.

4.2.2.3 Seuillage par apprentissage

La première étape permet d'approcher l'histogramme, noté h, d'une image quelconque codée sur une échelle de L niveaux de gris, par une somme de d fdp [Synd 90]. Dans notre cas, des fdp gaussiennes sont utilisées, et l'approximation de h est donnée par :

$$h_{approx}(x) = \sum_{i=1}^{d} P_i \exp\left[-\frac{(x-\mu_i)^2}{\sigma_i^2}\right] \tag{4.2.2}$$

où P_i est l'amplitude de la $i^{\text{ième}}$ gaussienne, μ_i est la moyenne et σ_i^2 est la variance.

L'histogramme h est normalisé de la manière suivante :

$$h_{norm}(i) = \frac{h(i)}{\sum_{j=0}^{L-1} h(j)} \tag{4.2.3}$$

Approcher l'histogramme h avec la somme de gaussiennes h_{approx} revient donc à minimiser le critère J en fonction du jeu de paramètres Θ. L'expression du critère J est donnée par :

$$J(\Theta) = \sum_{i=0}^{L-1} \left| h_{norm}(i) - h_{approx}(\Theta, i) \right|^2 \tag{4.2.4}$$

Le jeu de paramètres Θ est donné par :

$$\Theta = \{P_i, \mu_i, \sigma_i \; ; \; i = 1, ..., d\} \tag{4.2.5}$$

Si l'on suppose que l'histogramme est correctement approché par la somme de gaussiennes, alors les seuils optimaux sont calculés en minimisant la probabilité de recouvrement des différentes gaussiennes. Pour deux gaussiennes successives, l'expression de cette probabilité est :

$$E(T_i) = P_i \int_{-\infty}^{T_i} p_i(x)dx + P_{i+1} \int_{T_i}^{+\infty} p_{i+1}(x)dx \tag{4.2.6}$$

où T_i est le $i^{\text{ième}}$ seuil de l'image, p_i est la $i^{\text{ième}}$ gaussienne de l'approximation, et $i = 1, ..., d-1$.

Minimiser cette erreur revient à différencier $E(T_i)$ selon le critère de Leibniz en fonction de T_i et à l'égaler à zéro. Cela donne :

$$P_i \; p_i(x) = P_{i+1} \; p_{i+1}(x) \tag{4.2.7}$$

Dans le cas de fdp gaussiennes, on aboutit à la résolution d'une équation du second ordre, dont l'expression est donnée par :

$$A\ T_i^2 + B\ T_i + C = 0 \tag{4.2.8}$$

avec :

$$A = \sigma_i^2 + \sigma_{i+1}^2 \tag{4.2.9}$$

$$B = 2\left(\mu_i\ \sigma_{i+1}^2 - \mu_{i+1}\ \sigma_i^2\right) \tag{4.2.10}$$

$$C = \mu_{i+1}^2\ \sigma_i^2 - \mu_i^2\ \sigma_{i+1}^2 + 4\ \sigma_i^2\ \sigma_{i+1}^2 \ln\left(\frac{\sigma_{i+1}\ P_i}{\sigma_i\ P_{i+1}}\right) \tag{4.2.11}$$

Cette expression a deux solutions possibles, mais seule l'une des deux est acceptable dans le cas présent [Kitt 86].

4.2.3 Protocole expérimental et paramétrage

Pour comparer les performances de *PSO-2S* et de SPSO-07, le critère (4.2.4) est minimisé pour les quatre images de la figure 4.1. Le critère de stagnation utilisé est satisfait si aucune amélioration significative (supérieure à 1E−10) n'est observée pour la meilleure solution courante, durant 1E+4 évaluations successives de la fonction objectif. De plus, le nombre maximum d'évaluations autorisées est fixé à $Max_{Fes} = 300000$.

Sur cette figure, les images (a) et (b) sont des images de référence utilisées pour la validation de méthodes de segmentation, dans la littérature. Les images (c) et (d) correspondent, quant à elles, à des images obtenues depuis BrainWeb [Bran 12]. Les paramètres utilisés pour la simulation d'IRM cérébrale sont une séquence pondérée en T1, une épaisseur de coupe de 1 mm, un bruit gaussien de 3% calculé par rapport au tissu d'intensité maximale, et un niveau d'inhomogénéité d'intensité du champ radio-fréquence de 20%.

Les valeurs des paramètres de *PSO-2S* et de SPSO-07 utilisées pour ce problème de segmentation, sont présentées ci-dessous :

– *PSO-2S* (variante utilisant une recherche locale telle que décrite dans le chapitre 2) en utilisant 30 zones et $\frac{p^4}{7000} + 10$ particules dans chaque zone, avec p le numéro de la zone. La valeur du paramètre K utilisé pour engendrer le voisinage des particules est fixée à $K = 3$. Le paramètre désigné par $nb_{generation}$ est fixé à 15.

(a) (b) (c) (d)

FIGURE 4.1: Les images tests : (a) LENA, (b) BRIDGE, (c) IRMCT (coupe transversale), (d) IRMCC (coupe coronale).

– SPSO-07 (*Standard Particle Swarm Optimization* dans sa version 2007) [Cler 12] en utilisant le paramétrage recommandé par son auteur (le paramètre K utilisé pour engendrer le voisinage des particules ($K = 3$) et la taille de l'essaim S est calculée automatiquement par la formule : $S = 10 + 2\sqrt{D}$).

4.2.4 Résultats et discussion

Nous présentons maintenant les résultats expérimentaux obtenus avec *PSO-2S* et SPSO-07. Les résultats de segmentation sont présentés à la figure 4.2. Sur cette figure, les images originales et leurs histogrammes sont illustrés en (a) et en (b), respectivement. Les histogrammes approchés sont illustrés en (c), et les images segmentées (en 5 classes) sont illustrées en (d). Pour chaque image test, nous pouvons constater que l'approximation de son histogramme, illustrée en détail pour LENA à la figure 4.3, permet d'obtenir une bonne segmentation de l'image.

Les résultats d'estimation d'histogramme, pour chaque image test, sont présentés dans le tableau 4.1. Dans ce tableau, les paramètres de chacune des cinq gaussiennes utilisées pour approcher l'histogramme de chaque image sont donnés. Les paramètres de la $i^{\text{ième}}$ gaussienne d'une image sont notés P_i, μ_i et σ_i. Les valeurs de seuils séparant les différentes classes de pixels, calculées pour chaque image via l'approximation de son histogramme, sont ensuite données dans le tableau 4.2.

Pour chaque image test, le nombre d'évaluations effectuées par chaque algorithme, moyenné sur 100 exécutions, est donné dans le tableau 4.3. Le taux de succès (le pourcentage sur les 100 exécutions de solutions acceptables trouvées, *i.e.* le pourcentage de

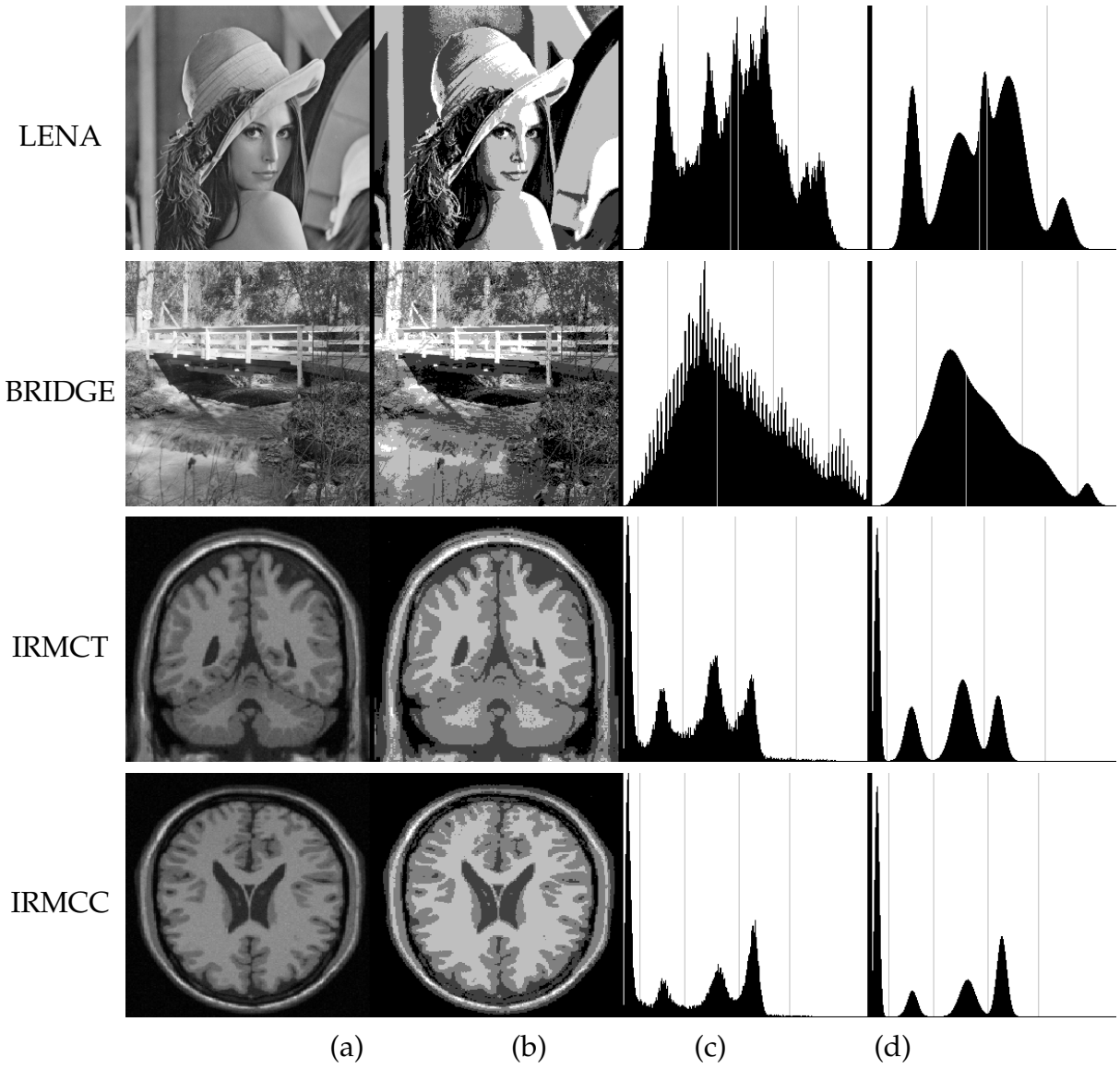
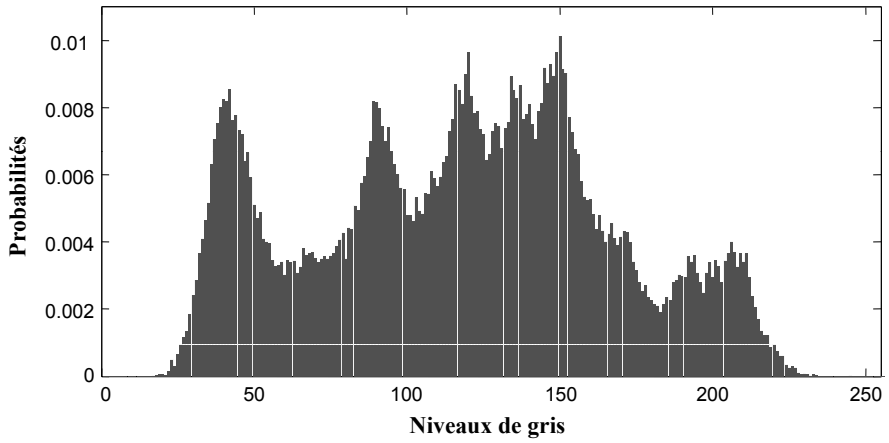


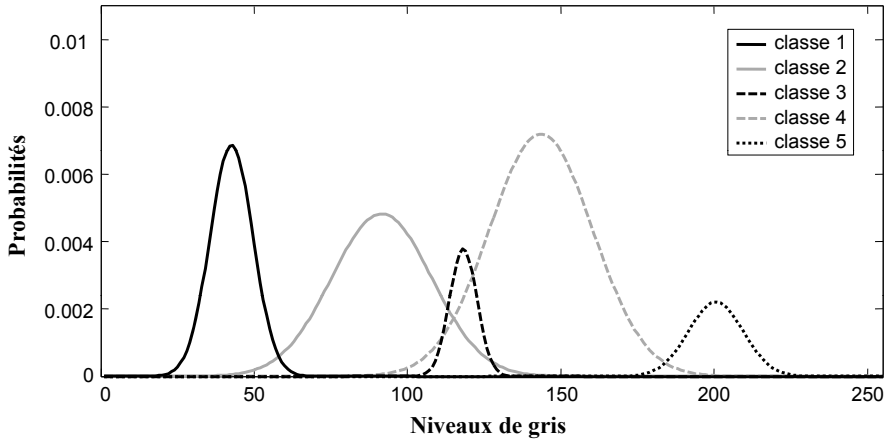
FIGURE 4.2: Illustration de la procédure de segmentation. (a) Images originales. (b) Images segmentées. (c) Histogrammes originaux. (d) Histogrammes approchés.

Image	μ_1	P_1	σ_1	μ_2	P_2	σ_2	μ_3	P_3	σ_3	μ_4	P_4	σ_4	μ_5	P_5	σ_5
LENA	41,62	0,17	9,87	90,57	0,28	23,14	117,21	0,06	6,33	142,54	0,43	23,82	199,56	0,07	12,64
BRIDGE	41,13	0,06	15,36	76,75	0,39	25,97	119,25	0,38	34,73	173,38	0,14	28,43	225,53	0,02	9,09
IRMCT	4,71	0,28	3,78	40,79	0,17	10,05	94,50	0,36	14,26	131,55	0,19	9,17	225,60	0,00	252,92
IRMCC	4,66	0,41	3,66	41,76	0,09	7,64	99,57	0,21	11,92	135,53	0,29	7,46	192,73	0,00	239,82

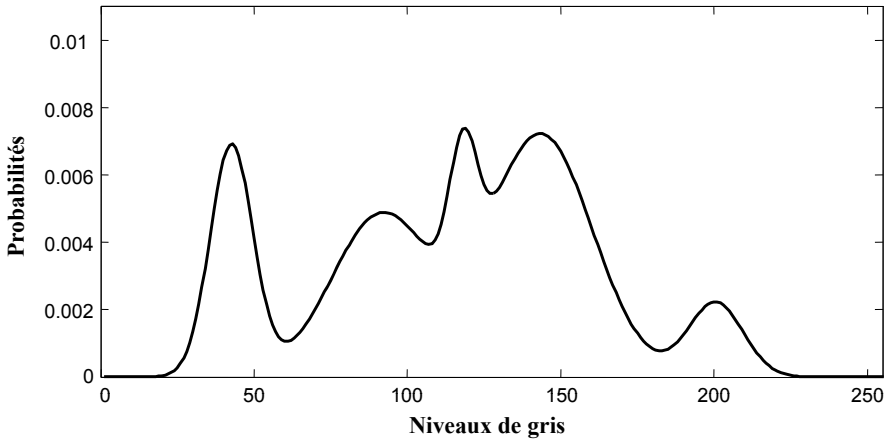
TABLEAU 4.1: Paramètres des gaussiennes utilisés pour estimer l'histogramme de chaque image test.



(a)



(b)



(c)

FIGURE 4.3: Illustration de la procédure d'approximation d'histogramme pour LENA. (a) Histogramme original. (b) Fonctions gaussiennes de chaque classe de pixels. (c) Somme des fonctions gaussiennes (histogramme approché).

Image	Seuils
LENA	57 112 120 183
BRIDGE	46 98 156 215
IRMCT	15 62 117 184
IRMCC	17 64 121 177

TABLEAU 4.2: Valeurs des seuils pour chaque image test.

solutions ayant une valeur de fonction objectif inférieure ou égale à 5,14E−4 ; 5,09E−4 ; 7,51E−4 ; 7,99E−4 pour LENA ; BRIDGE ; IRMCT ; IRMCC, respectivement), ainsi que l'erreur moyenne d'approximation (la valeur moyenne de la fonction objectif pour la meilleure solution trouvée) de l'histogramme d'une image sont aussi donnés dans ce tableau, pour 100 exécutions d'un algorithme.

Image	Algorithme	Evaluations	Erreur d'approximation	Taux de succès
LENA	PSO-2S	119721,7 ± 64844,3	5,44E−4 ± 3,06E−5	41 %
	SPSO-07	67057,1 ± 64316,9	5,52E−4 ± 3,08E−5	25 %
BRIDGE	PSO-2S	241537,8 ± 70707,8	5,27E−4 ± 9,54E−5	48 %
	SPSO-07	125524,4 ± 63277,9	5,16E−4 ± 6,01E−6	24 %
IRMCT	PSO-2S	81080,9 ± 66569,8	7,69E−4 ± 1,26E−4	95 %
	SPSO-07	44212,0 ± 57321,5	8,79E−4 ± 2,73E−4	77 %
IRMCC	PSO-2S	72922,5 ± 35894,4	8,68E−4 ± 1,18E−4	70 %
	SPSO-07	28185,4 ± 16188,4	9,46E−4 ± 2,71E−4	54 %

TABLEAU 4.3: Nombre moyen d'évaluations pour segmenter une image, erreur d'approximation et taux de succès obtenus par chaque algorithme, pour chaque image test.

Dans ce tableau, nous constatons que *PSO-2S* nécessite plus d'évaluations que SPSO-07 pour converger vers une solution acceptable. En revanche, son taux de succès est significativement supérieur à celui de SPSO-07 pour toutes les images, d'après le test exact de Fisher [Fish 22] avec un niveau de confiance de 95%. En effet, *PSO-2S* est conçu pour éviter une convergence prématurée de l'algorithme PSO. Il permet d'améliorer significativement la stabilité de la méthode de segmentation utilisée. Nous montrons ainsi l'intérêt d'utiliser *PSO-2S* pour ce type de problème.

4.2.5 Conclusion

Dans cette section, nous avons présenté une méthode de segmentation par estimation d'histogramme permettant d'identifier plusieurs classes de pixels, aussi bien dans des images standard que dans des images médicales. Cette méthode comporte une étape d'optimisation dans laquelle nous avons intégré notre algorithme *PSO-2S*. Nous avons également testé la méthode en utilisant l'algorithme SPSO-07.

Les résultats de segmentation obtenus sur plusieurs images tests communément utilisées dans la littérature en traitement d'images, ainsi que sur des images synthétiques issues de simulations paramétrables d'IRM cérébrale, sont satisfaisants. Ensuite, nous avons montré que l'utilisation de *PSO-2S* confère une plus grande stabilité à la méthode de segmentation, par rapport à SPSO-07. Les résultats montrent ainsi l'intérêt d'utiliser *PSO-2S* sur ce type de problème.

4.3 Application de PSO-2S au dimensionnement de circuits électroniques

4.3.1 Introduction

L'intégration à très grande échelle (VLSI - *Very Large Scale Integration*) est une technologie de circuit intégré (CI) dont la densité d'intégration permet de supporter plus de 100 000 composants électroniques sur une même puce et, donc, de réaliser des circuits intégrés de haute complexité [Giel 91]. Les composants analogiques prennent une part importante dans la conception de tels circuits, tant pour les systèmes à signal mixte que pour les systèmes numériques [Giel 91, Toum 93]. En effet, les paramètres des composants analogiques influencent directement les performances de ces circuits.

La conception de circuits analogiques est un processus complexe, qui consiste à réaliser de nombreuses étapes de développement, pour concevoir correctement et sans erreurs un tel circuit. Elle ne concerne pas seulement le placement et le routage des composants, mais aussi leur dimensionnement [Grae 01]. Normalement, le dimensionnement des composants analogiques est un processus lent, itératif et qui nécessite beaucoup de temps. Ce processus peut être réalisé grâce à l'expérience et l'intuition d'un expert [Conn 96]. Donc, trouver une manière efficace et rapide d'accélérer le processus de conception des circuits intégrés est indispensable. Les approches qui sont basées sur des

topologies fixes et/ou des techniques stochastiques sont les plus connues dans la littérature [Mede 94, Silv 96, Conn 96, Loul 02]. Ces méthodes sont initialisées avec une bonne solution fournie par un expert en conception de circuits analogiques. Le problème de ces méthodes est qu'elles sont souvent très lentes et qu'elles ne garantissent pas la convergence vers un optimum global.

En effet, le problème de dimensionnement peut être considéré comme un problème d'optimisation à variables continues soumis à des contraintes. Les variables de décision sont alors les largeurs et les longueurs des transistors du circuit, les contraintes étant imposées par le cahier des charges du circuit.

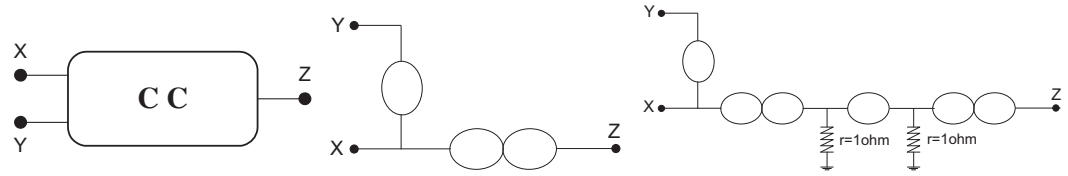
Dans ce qui suit, nous présentons une application de *PSO-2S* au dimensionnement d'un circuit analogique. Plus précisément, dans cette application, nous cherchons à optimiser les caractéristiques importantes d'un *convoyeur de courant de seconde génération positif à boucle translinéaire CMOS* afin d'améliorer sa performance. Ces caractéristiques sont la résistance parasite et la fréquence de coupure en courant [Toum 93, Fabr 98].

Il est à noter ici que cette application est faite en collaboration avec Mourad Fakhfakh, Maître de Conférences à l'École Nationale d'Ingénieurs de Sfax, en Tunisie.

4.3.2 Convoyeur de courant de seconde génération

Les convoyeurs de courant (CCs, *Current Conveyors*) ont été introduits en 1970 [Sedr 70]. Ces convoyeurs constituent à ce jour l'élément le plus courant des circuits analogiques en mode courant (CMCs, *Current Mode Circuits*) [Sedr 90, Toum 93]. Il est notoire que les circuits en mode tension (VMCs, *Voltage Mode Circuits*), tels que les amplificateurs opérationnels (*Op-amps*), les convertisseurs tension-fréquence (VFCs, *Voltage-to-Frequency Converters*), les comparateurs de tension, etc., ne sont pas adaptés aux opérations à haute fréquence, en raison de leurs contraintes dues aux faibles bandes passantes. De tels problèmes se posent dans les VMCs, en raison de la présence de capacités parasites [Rajp 07]. Les CMCs se sont avérés plus performants que les VFCs parce que le fonctionnement du circuit dépend principalement des courants et, donc, il est possible de concevoir des circuits utilisant les CMCs qui peuvent fonctionner sur une large gamme dynamique. Parmi les configurations les plus populaires de CMCs, nous citons les convoyeurs de courant, spécialement ceux de deuxième génération (CCIIs), qui ont reçu beaucoup d'attention et présentent de nombreux avantages par rapport à leurs homologues VMCs.

Un convoyeur de courant est un dispositif électronique composé de trois ports actifs. Sa représentation conventionnelle est illustrée dans la figure 4.4 (a). Les figures 4.4 (b) et 4.4 (c) illustrent, respectivement, les schémas équivalents nullateurs-norateurs qui reproduisent les comportements idéaux des convoyeurs de courant CCII+ et CCII− [Tlel 10, Schm 00].



(a) (b) (c)

FIGURE 4.4: (a) Représentation générale d'un convoyeur de courant, (b) Schéma équivalent nullateur-norateur d'un CCII- idéal, et (c) Schéma équivalent nullateur-norateur d'un CCII+ idéal.

Les différents courants et tensions aux trois bornes X, Y et Z sont liés par la relation suivante [Toum 93, Ben 06] :

$$\begin{pmatrix} I_Y \\ V_X \\ I_Z \end{pmatrix} = \begin{pmatrix} 0 & \alpha & 0 \\ 1 & 0 & 0 \\ 0 & \beta & 0 \end{pmatrix} \begin{pmatrix} V_Y \\ I_X \\ V_Z \end{pmatrix}$$

Dans l'équation ci-dessus, α spécifie le type de convoyeur de courant utilisé et peut prendre deux valeurs 0 ou 1 :

- Si $\alpha = 0$, le circuit est considéré comme un convoyeur de courant de seconde génération (CCII),
- Si $\alpha = 1$, alors le circuit est considéré comme un convoyeur de courant de première génération (CCI).

et β caractérise le transfert de courant entre les ports X et Z et peut prendre deux valeurs -1 ou 1 :

- Si $\beta = -1$, le circuit est classé comme un convoyeur de courant *négatif*,
- Si $\beta = 1$, le circuit est considéré comme un convoyeur de courant *positif*.

Un convoyeur de courant de seconde génération assure les deux fonctionnalités suivantes :

– suiveur de courant entre les ports X et Z ;

– suiveur de tension entre les ports X et Y.

Dans le cas idéal, et pour assurer les bonnes fonctionnalités du circuit, un convoyeur de courant de deuxième génération est caractérisé par une faible impédance sur le port X et de hautes impédances sur les ports Y et Z.

Le convoyeur de courant de seconde génération est le convoyeur le plus souvent utilisé, tant pour ses bonnes performances, que pour l'intérêt apporté par l'utilisation d'une *boucle translinéaire* [Seev 00]. La figure 4.5 montre l'exemple d'un convoyeur de courant de seconde génération positif à boucle translinéaire CMOS [Ben 06]. Les transistors M_1 à M_4 constituent la boucle translinéaire qui assure $V_X = V_Y$. I_0 est le courant de polarisation et les transistors M_9 à M_{13} constituent des *miroirs de courant*. Les transistors M_5 à M_8 servent à reproduire au port Z le courant appliqué au port X.

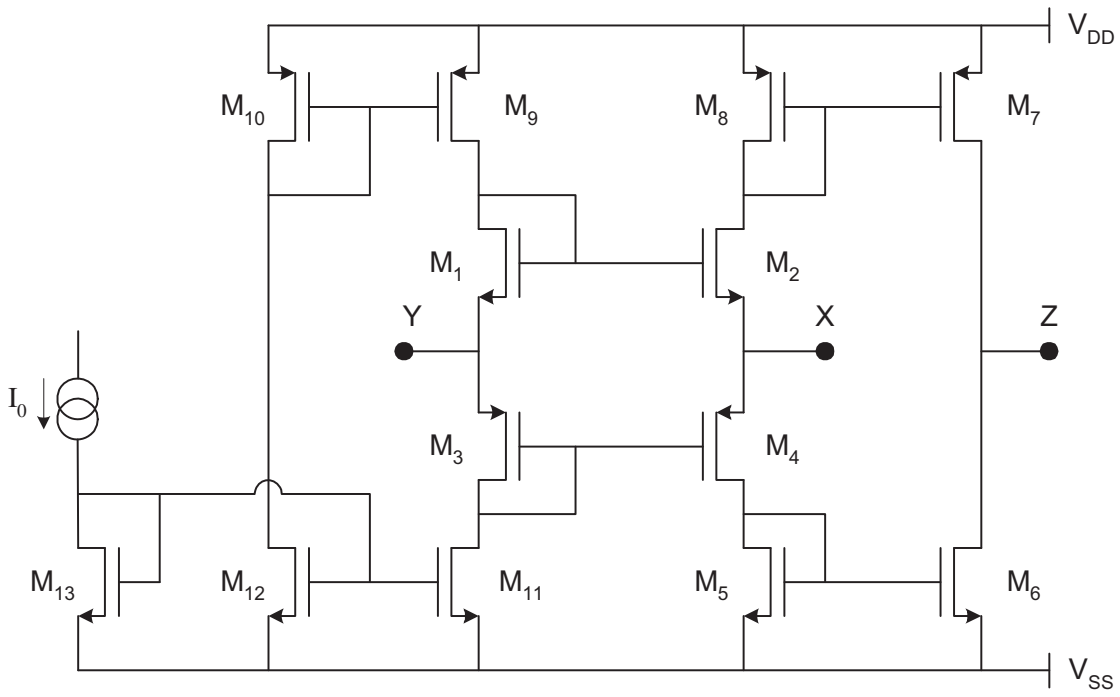


FIGURE 4.5: Convoyeur de courant de seconde génération positif (CCII+).

Lorsqu'il est conçu au niveau microélectronique, le comportement d'un convoyeur de courant peut différer du cas idéal. La Figure 4.6 montre les différents composants parasites d'un convoyeur de courant qui sont souvent modélisés par trois impédances Z_X, Z_Y et Z_Z et reliés aux ports correspondants. Ces composants affectent de manière significative le

comportement du circuit. Il a été prouvé que, parmi ces éléments, Z_X est le composant parasite dont les effets sont les plus importants sur le comportement du convoyeur de courant. Plus précisément, c'est la résistance parasite R_X du port X qui est le plus dominant et qui affecte les performances de la façon la plus significative du CC.

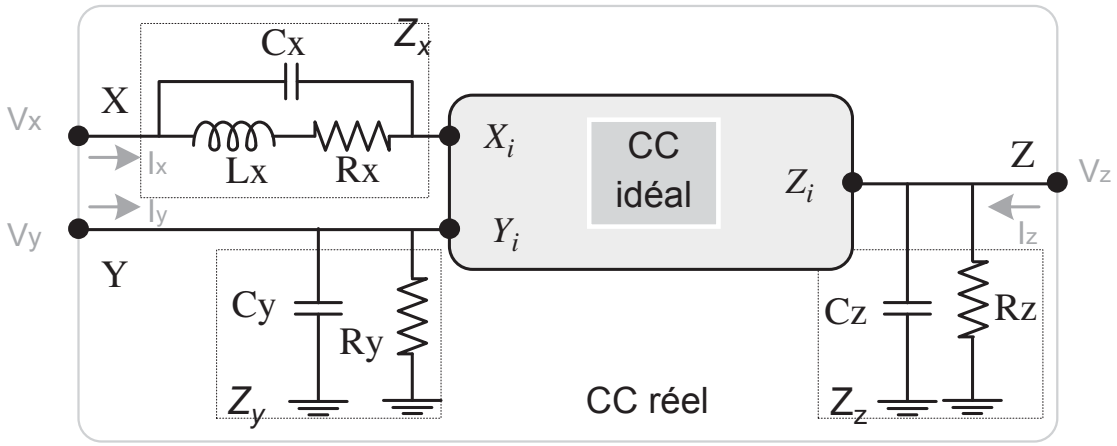


FIGURE 4.6: Schéma équivalent avec composants parasites.

Dans le cas d'applications hautes fréquences, ces circuits doivent être précisément dimensionnés si l'on veut que les performances soient en accord avec les souhaits du concepteur. La bande passante en courant limite le domaine d'application du circuit, car la bande passante en courant est moins large que celle en tension. Donc, dans cette application, nous cherchons à optimiser les caractéristiques importantes du circuit, i.e. la résistance parasite R_X du port X et la fréquence de coupure en courant f_{ci}.

4.3.3 Les fonctions objectifs de composants

L'expression de la résistance parasite R_X est donnée par [Ben 06] :

$$R_X = \frac{1}{\sqrt{2K_N(\frac{W_N}{L_N})(1+\lambda_N V_{DS})I_0} + \sqrt{2K_P(\frac{W_P}{L_P})(1+\lambda_P V_{DS})I_0}} \tag{4.3.1}$$

où :

- $\lambda_N, \lambda_P, K_N = 0{,}9386.10^{-8}$ et $K_P = 0{,}3476.10^{-6}$ sont des paramètres liés à la technologie des transistors (AMS 0,35 µm),
- I_0 est le courant de polarisation et V_{DS} est la tension drain-source des transistors,

– W_N et L_N sont respectivement la largeur et la longueur des transistors NMOS,

– et W_P et L_P sont respectivement la largeur et la longueur des transistors PMOS.

L'expression de la fréquence de coupure en courant f_{ci} est calculée à partir d'une approche symbolique. Son expression n'est pas donnée ici, du fait de son très grand nombre de termes.

Tous les transistors doivent opérer en mode saturation. Les contraintes de saturation sont données par :

$$K_N \frac{W_N}{L_N} = K_P \frac{W_P}{L_P} \tag{4.3.2}$$

$$W_N L_N = W_P L_P \tag{4.3.3}$$

$$\frac{V_{DD}}{2} - V_{TN} - \sqrt{\frac{I_0}{K_N \frac{W_N}{L_N}}} > \sqrt{\frac{I_0}{K_P \frac{W_P}{L_P}}} \tag{4.3.4}$$

$$\frac{V_{DD}}{2} - V_{TP} - \sqrt{\frac{I_0}{K_P \frac{W_P}{L_P}}} > \sqrt{\frac{I_0}{K_N \frac{W_N}{L_N}}} \tag{4.3.5}$$

où $V_{TN} = 0{,}4655V$ et $V_{TP} = 0{,}617V$ sont des paramètres inhérents à la technologie utilisée. V_{DD} est la source de tension de 2,5 V.

Le but est de minimiser R_X et de maximiser f_{ci} en fonction de W_N, L_N, W_P et L_P , en respectant les conditions imposées par les contraintes présentées ci-dessus. Les limites technologiques imposent que :

$$35.10^{-8} \leq L_N, L_P \leq 13.10^{-6} \tag{4.3.6}$$

$$1.10^{-6} \leq W_N, W_P \leq 30.10^{-6} \tag{4.3.7}$$

Comme notre objectif est de minimiser R_x et de maximiser f_{ci} simultanément, et comme ces deux quantités sont essentiellement positives, nous configurons le problème comme un problème d'optimisation *bi-objectif* à minimiser. La nouvelle fonction objectif à minimiser, notée f_{obj}, est alors donnée comme suit :

$$f_{obj} = \left(\frac{1}{f_{ci}}\right) + R_X \tag{4.3.8}$$

4.3.4 L'évaluation des performances

Les valeurs optimales pour les dimensions physiques des transistors MOS (i.e les valeurs de W_N et L_N pour chaque transistor NMOS et les valeurs de W_P et L_P pour chaque transistor PMOS) sont obtenues en minimisant f_{obj}, pour une série du courant de polarisation (i.e. avec différentes valeurs de I_0). Les simulations sont réalisées en utilisant la technologie CMOS AMS 0,35 µm, une alimentation de 2,5 V et l'ensemble des valeurs du courant de polarisation est considéré $I_0 \in \{50, 100, 150, 200, 250, 300\}$ µA. Les valeurs des paramètres de *PSO-2S* utilisées pour ce problème, désignés par $nb_{generation}$et $nb_{particule}$, sont fixées à 40 et 2, respectivement. En outre, le paramètre max_{zone} est fixé à 30, sachant que ce paramètre signifie aussi la taille de l'essaim principal $S1$. Le critère d'arrêt pour *PSO-2S* est $Max_{Fes} = 100000$.

Les tableaux $4.4 - 4.9$ montrent les valeurs optimales des paramètres de conception (i.e. les valeurs optimales de dimensions obtenues pour CCII+), ainsi que les valeurs associées obtenues pour R_x et f_{ci}, en utilisant l'algorithme *PSO-2S*. Chacun de ces tableaux correspond à une valeur de I_0, et présente les résultats obtenus par *PSO-2S* sur cinq exécutions. Les meilleurs résultats sont indiqués en gras ; la ligne en gras dans chaque tableau désigne la meilleure exécution.

N° exéc.	L_N(µm)	W_N(µm)	L_P(µm)	W_P(µm)	f_{ci} (GHz)	R_X(ohm)
#1	0,52	15,98	0,35	26,38	0,909	594
#2	**0,52**	**15,30**	**0,35**	**25,10**	**0,925**	**607**
#3	0,52	17,48	0,35	28,85	0,868	567
#4	0,52	17,47	0,35	28,82	0,869	568
#5	0,52	17,70	0,35	28,99	0,859	565

TABLEAU 4.4: Valeurs optimales des paramètres W_N, L_N, W_P et L_P de transistors NMOS et PMOS obtenues par PSO-2S sur cinq exécutions, avec un courant de polarisation $I_0 = 50$µA.

Pour vérifier les résultats obtenus par *PSO-2S*, nous avons mis en œuvre des simulations à l'aide de SPICE [1] pour le modèle CCII à l'étude. Pour ce faire, nous choisissons les meilleures valeurs optimales des dimensions (i.e. la meilleure exécution parmi les cinq)

1. SPICE (*Simulation Program with Integrated Circuit Emphasis*) est un logiciel de simulation généraliste de circuits électroniques analogiques.

N° exéc.	$L_N(\mu m)$	$W_N(\mu m)$	$L_P(\mu m)$	$W_P(\mu m)$	f_{ci} (GHz)	R_X(ohm)
#1	**0,52**	**14,30**	**0,35**	**23,62**	**1,358**	**444**
#2	0,52	17,70	0,35	29,47	1,215	399
#3	0,54	17,19	0,35	27,41	1,211	408
#4	0,57	15,33	0,35	23,27	1,240	437
#5	0,52	17,69	0,35	29,17	1,220	399

TABLEAU 4.5: Valeurs optimales des paramètres W_N, L_N, W_P et L_P de transistors NMOS et PMOS obtenues par PSO-2S sur cinq exécutions, avec un courant de polarisation $I_0 = 100\mu A$.

N° exéc.	$L_N(\mu m)$	$W_N(\mu m)$	$L_P(\mu m)$	$W_P(\mu m)$	f_{ci} (GHz)	R_X(ohm)
#1	0,58	12,14	0,35	18,22	1,664	362
#2	**0,53**	**14,41**	**0,35**	**23,53**	**1,761**	**404**
#3	0,53	13,92	0,35	22,64	1,671	368
#4	0,53	11,30	0,35	18,41	1,855	409
#5	0,56	12,26	0,35	19,08	1,646	397

TABLEAU 4.6: Valeurs optimales des paramètres W_N, L_N, W_P et L_P de transistors NMOS et PMOS obtenues par PSO-2S sur cinq exécutions, avec un courant de polarisation $I_0 = 150\mu A$.

N° exéc.	$L_N(\mu m)$	$W_N(\mu m)$	$L_P(\mu m)$	$W_P(\mu m)$	f_{ci} (GHz)	R_X(ohm)
#1	0,52	17,19	0,35	28,28	1,749	286
#2	0,52	13,07	0,35	21,43	1,999	329
#3	0,53	15,77	0,35	25,41	1,800	300
#4	**0,52**	**16,36**	**0,35**	**27,02**	**1,793**	**293**
#5	0,55	12,88	0,35	19,95	1,940	335

TABLEAU 4.7: Valeurs optimales des paramètres W_N, L_N, W_P et L_P de transistors NMOS et PMOS obtenues par PSO-2S sur cinq exécutions, avec un courant de polarisation $I_0 = 200\mu A$.

N° exéc.	$L_N(\mu m)$	$W_N(\mu m)$	$L_P(\mu m)$	$W_P(\mu m)$	f_{ci} (GHz)	R_X(ohm)
#1	0,54	16,73	0,35	26,54	1,932	262
#2	**0,53**	**15,34**	**0,35**	**24,84**	**2,045**	**272**
#3	0,54	15,65	0,35	25,49	1,993	271
#4	0,53	17,51	0,35	29,52	1,915	255
#5	0,53	15,78	0,35	25,61	2,021	268

TABLEAU 4.8: Valeurs optimales des paramètres W_N, L_N, W_P et L_P de transistors NMOS et PMOS obtenues par PSO-2S sur cinq exécutions, avec un courant de polarisation $I_0 = 250\mu A$.

N° exéc.	$L_N(\mu m)$	$W_N(\mu m)$	$L_P(\mu m)$	$W_P(\mu m)$	f_{ci} (GHz)	R_X(ohm)
#1	**0,52**	**18,04**	**0,35**	**29,79**	**2,095**	**228**
#2	0,52	17,76	0,35	29,25	2,097	230
#3	0,53	18,62	0,35	30,00	2,027	226
#4	0,54	18,89	0,35	30,00	1,996	225
#5	0,54	18,83	0,35	30,00	2,003	225

TABLEAU 4.9: Valeurs optimales des paramètres W_N, L_N, W_P et L_P de transistors NMOS et PMOS obtenues par PSO-2S sur cinq exécutions, avec un courant de polarisation $I_0 = 300\mu A$.

obtenues par *PSO-2S* pour chacun des six cas. Ces valeurs sont utilisées dans SPICE pour déterminer les valeurs exactes de R_x et f_{ci}, que nous devrions nous attendre à obtenir si la configuration de CCII+ est physiquement réalisée avec les dimensions physiques optimales correspondant à celles des transistors NMOS et PMOS. En outre, pour permettre une comparaison correcte de ces résultats obtenus en utilisant *PSO-2S*, les résultats sont comparés avec ceux d'algorithmes d'optimisation stochastique concurrents, qui peuvent être potentiellement utilisés pour résoudre ce genre de problème.

Afin de réaliser ces comparaisons, nous nous sommes basés sur des résultats présentés dans [Chat 10]. Ces résultats sont obtenus à partir de trois algorithmes : DE, PSO et BFO (*bacterial foraging optimization* [Pass 02]). Le tableau 4.10 présente les résultats de ces trois algorithmes, ainsi que les résultats obtenus par *PSO-2S*. Plus précisément, ce tableau montre les résultats directs de R_x et f_{ci} obtenus par chacun de ces quatre algorithmes, ainsi que les résultats de simulations obtenus par SPICE, qui sont associés aux valeurs de di-

$I_0(\mu A)$	Résultats	f_{ci}(GHz)				R_X(ohm)			
		BFO	PSO	DE	PSO-2S	BFO	PSO	DE	PSO-2S
50	Direct	**1,245**	0,866	1,184	0,925	860	1376	865	**607**
	Simulation	**1,228**	0,840	1,181	0,982	1044	1821	1060	**671**
100	Direct	**1,812**	1,802	1,729	1,358	608	633	607	**444**
	Simulation	**1,629**	1,620	1,543	1,539	815	848	821	**538**
150	Direct	1,736	1,721	1,735	**1,761**	405	435	406	**404**
	Simulation	1,581	1,564	1,569	**1,746**	546	597	548	**471**
200	Direct	1,846	**2,027**	1,686	1,793	305	338	305	**293**
	Simulation	1,639	1,774	1,543	**1,810**	410	471	413	**388**
250	Direct	2,036	1,940	2,006	**2,045**	272	272	273	**272**
	Simulation	1,772	1,750	1,763	**2,016**	**377**	382	382	381
300	Direct	**2,121**	2,042	2,055	2,095	231	230	232	**228**
	Simulation	1,825	1,788	1,748	**2,053**	324	324	324	**318**

TABLEAU 4.10: Une comparaison des meilleurs résultats obtenus par les algorithmes : BFO, PSO, DE et PSO-2S.

mensions obtenues par chaque algorithme (i.e. nous avons utilisé les valeurs de W_N, L_N, W_P et L_P de chaque algorithme pour faire les simulations SPICE, par lesquelles nous obtenons les valeurs de R_x et f_{ci}). L'intérêt de ces comparaisons est de montrer que *PSO-2S* donne des résultats encourageants par rapport aux algorithmes concurrents, et de vérifier que les résultats directs et ceux des simulations ne sont pas très différents. Il est à noter que la présence de différences, entre les valeurs théoriques (directes) et celles des simulations (SPICE), est tout à fait normal, car les modèles mathématiques utilisés présentent des erreurs provoquées par les simplifications.

Une étude attentive du tableau 4.10 montre que l'algorithme *PSO-2S* obtient de meilleurs résultats que les autres algorithmes en compétition sur cette application, et ce dans la majorité des cas (i.e. pour les différentes valeurs de I_0). En effet, sur ce tableau, nous pouvons remarquer que *PSO-2S* obtient les plus petites valeurs de R_x (la fonction à minimiser) dans les six cas étudiés. En revanche, pour f_{ci} (la fonction à maximiser), *PSO-2S* donne de meilleurs résultats dans deux cas de I_0 ($I_0 = 150$ µA et $I_0 = 250$ µA). Pour cette fonction, BFO surpasse les autres algorithmes dans trois cas ($I_0 = 50$ µA, $I_0 = 100$ µA et $I_0 = 300$ µA), pendant que PSO obtient de meilleurs résultats dans un seul cas ($I_0 = 200$ µA). Le fait de ne pas avoir les meilleurs résultats pour f_{ci} par *PSO-*

2S ne montre pas une faiblesse de l'algorithme, mais ceci s'éxplique par la minimisation d' une fonction bi-objectif. En effet, nous pouvons voir sur ce tableau qu'en minimisant la fonction objectif f_{obj} ($f_{obj} = (\frac{1}{f_{ci}}) + R_X$), *PSO-2S* obtient les meilleurs résultats dans les six cas. De plus, les simulations SPICE prouvent que *PSO-2S* obtient les meilleurs valeurs pour R_x et f_{ci} dans la plupart des cas.

Les figures 4.7− 4.12 montrent les résultats correspondants à la simulation SPICE, pour les évolutions de f_{ci} et R_X en fonction de la fréquence, avec les dimensions optimales déterminées à partir de l'algorithme *PSO-2S*.

4.3.5 Conclusion

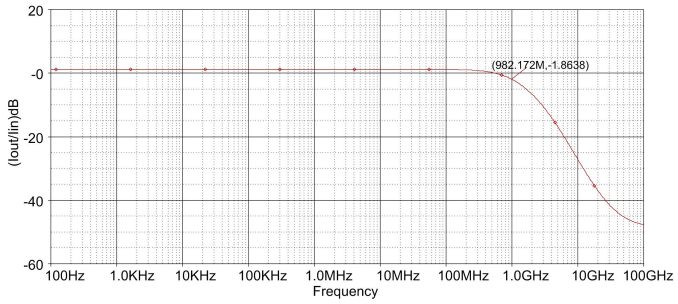
Dans cette application, nous avons réussi à démontrer comment un convoyeur de courant de seconde génération positif à boucle translinéaire CMOS (CCII+) peut être conçu de manière optimale pour donner de meilleures performances. En effet, la conception consiste à minimiser la résistance d'entrée X-port (R_X) et en même temps à maximiser la fréquence de coupure f_{ci} du CCII+. Ce problème a été traité et résolu correctement à l'aide de *PSO-2S*.

Une série d'expériences de simulation, réalisée avec plusieurs courants de polarisation I_0, a démontré que l'algorithme proposé obtient de meilleurs résultats, en comparaison de ceux obtenus par les algorithmes PSO, DE et BFO.

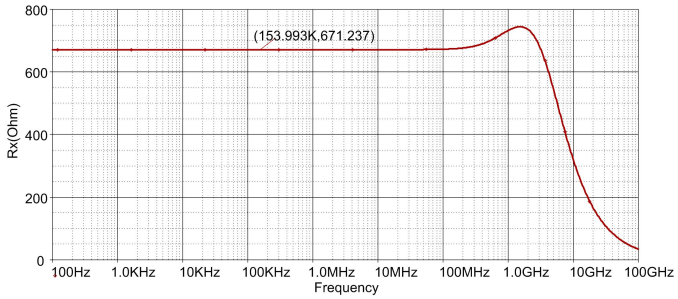
4.4 Conclusion

Dans la première partie de ce chapitre (section 4.2), une méthode permettant d'approcher l'histogramme par une somme de fonctions gaussiennes a été présentée. Une telle opération nous a permis d'obtenir une expression analytique approchée de l'histogramme, et donc de calculer beaucoup plus facilement les seuils optimaux. Puis, nous avons montré comment *PSO-2S* peut être utilisé pour résoudre un problème de segmentation d'images. Les résultats proposés pour des images de test standard, ainsi que pour des images médicales, montrent que cette méthode, couplée à l'utilisation de *PSO-2S*, offre une plus grande stabilité, par rapport à SPSO-07.

Dans la deuxième partie (section 4.3), nous avons présenté une application de l'optimisation au problème de dimensionnement de circuits électroniques analogiques. Ce type de problème est crucial lors de la conception de circuits analogiques, car les performances

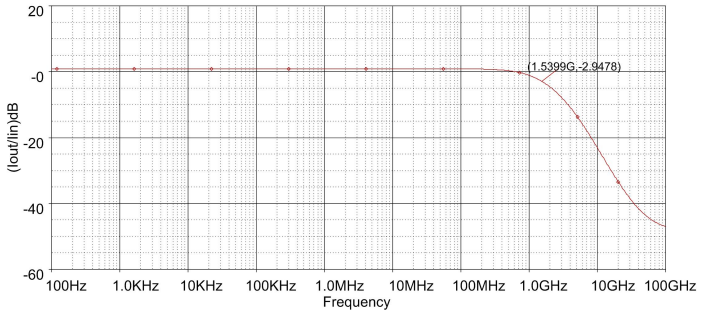


(a) Solution donnant un f_{ci} maximum

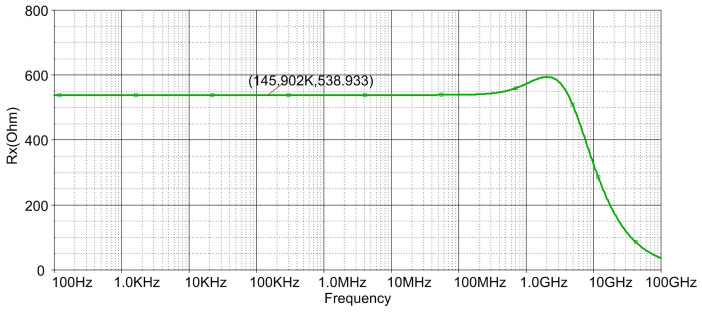


(b) Solution donnant un R_x minimum

FIGURE 4.7: Cas ($I_0 = 50$ µA) : les résultats de la simulation SPICE montrant la variation de (f_{ci} et R_x) par rapport à la fréquence, pour le dimensionnement optimal suggéré par PSO-2S.

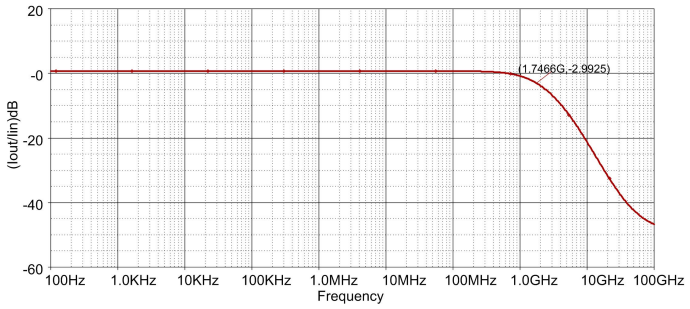


(a) Solution donnant un f_{ci} maximum

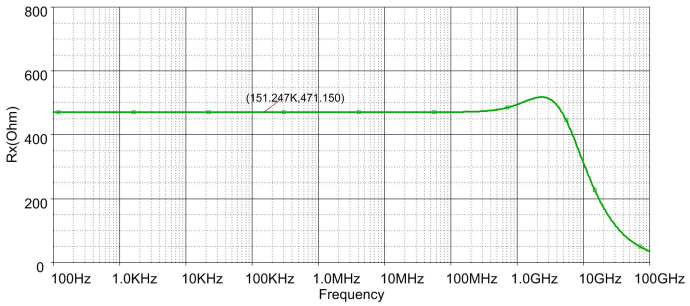


(b) Solution donnant un R_x minimum

FIGURE 4.8: Cas ($I_0 = 100$ µA) : les résultats de la simulation SPICE montrant la variation de (f_{ci} et R_x) par rapport à la fréquence, pour le dimensionnement optimal suggéré par PSO-2S.

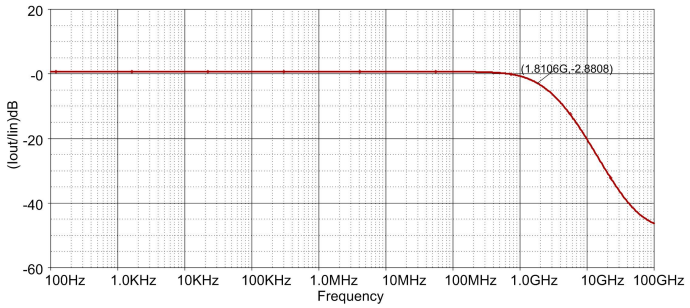


(a) Solution donnant un f_{ci} maximum

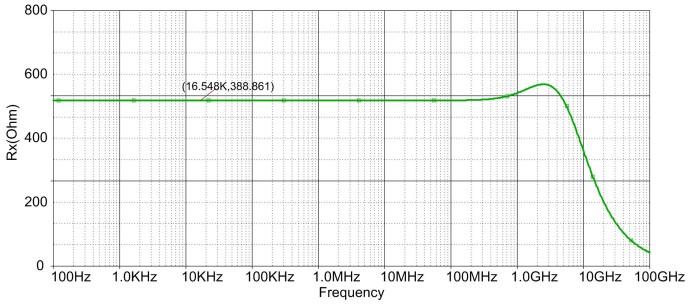


(b) Solution donnant un R_x minimum

FIGURE 4.9: Cas ($I_0 = 150$ µA) : les résultats de la simulation SPICE montrant la variation de (f_{ci} et R_x) par rapport à la fréquence, pour le dimensionnement optimal suggéré par PSO-2S.

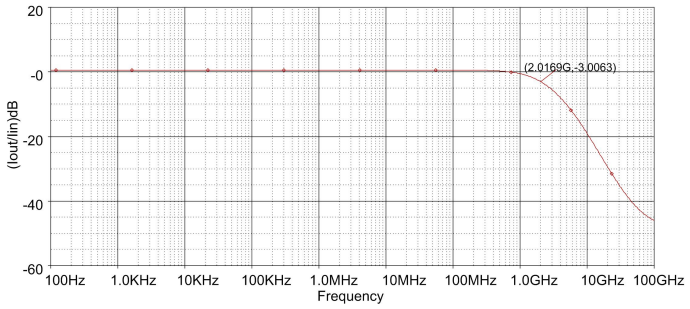


(a) Solution donnant un f_{ci} maximum

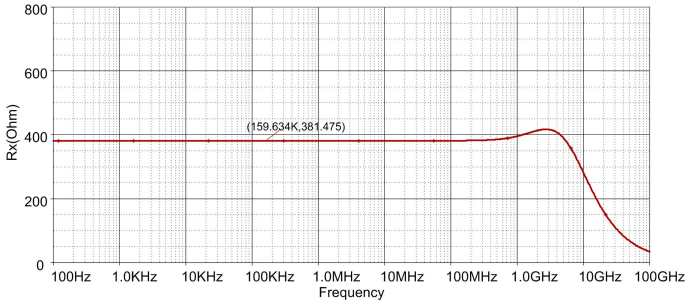


(b) Solution donnant un R_x minimum

FIGURE 4.10: Cas ($I_0 = 200$ µA) : les résultats de la simulation SPICE montrant la variation de (f_{ci} et R_x) par rapport à la fréquence, pour le dimensionnement optimal suggéré par PSO-2S.

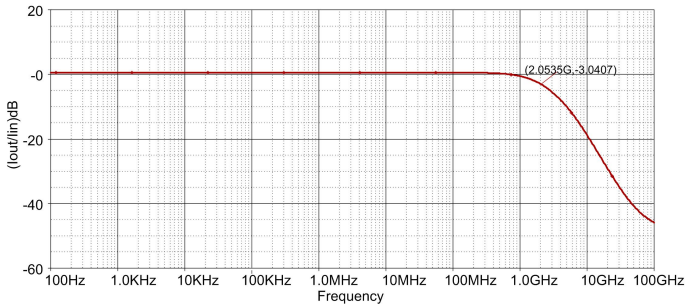


(a) Solution donnant un f_{ci} maximum

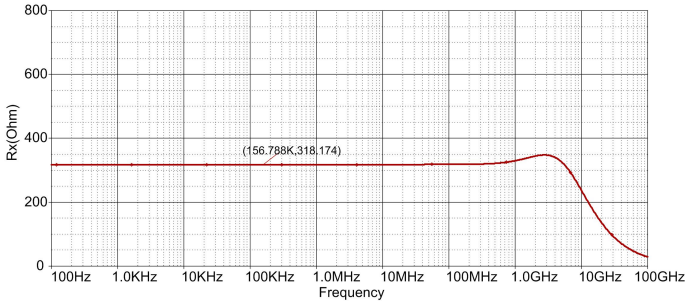


(b) Solution donnant un R_x minimum

FIGURE 4.11: Cas ($I_0 = 250$ µA) : les résultats de la simulation SPICE montrant la variation de (f_{ci} et R_x) par rapport à la fréquence, pour le dimensionnement optimal suggéré par PSO-2S.



(a) Solution donnant un f_{ci} maximum



(b) Solution donnant un R_x minimum

FIGURE 4.12: Cas ($I_0 = 300$ µA) : les résultats de la simulation SPICE montrant la variation de (f_{ci} et R_x) par rapport à la fréquence, pour le dimensionnement optimal suggéré par PSO-2S.

des circuits sont fortement dépendantes des dimensions des transistors. Cette application montre comment l'utilisation des métaheuristiques permet de trouver rapidement des solutions convenables, en termes de performance des circuits. Le problème est présenté comme un problème d'optimisation bi-objectif à minimiser. Ce problème a été traité à l'aide de *PSO-2S* et a donné des résultats compétitifs, par rapport aux autres algorithmes utilisés pour la comparaison.

Nous avons montré dans ce chapitre que les deux problèmes (i.e. la segmentation d'images et le dimensionnement de circuits électroniques) sont correctement résolus. En effet, les résultats obtenus pour les deux applications montrent l'intérêt de l'utilisation de *PSO-2S* sur ce type de problèmes.

Conclusion et perspectives

Les ingénieurs se heurtent quotidiennement à des problèmes technologiques de complexité grandissante, qui surgissent dans des domaines très divers, comme dans les transports, le traitement d'images, l'électronique, etc. La plupart de ces problèmes peuvent être formulés comme des problèmes d'optimisation. Les métaheuristiques forment un ensemble d'algorithmes utilisés en recherche opérationnelle pour résoudre ce type de problèmes. En effet, l'utilisation des métaheuristiques a fait l'objet d'un intérêt croissant du fait de leurs applications (notamment en traitement d'images et en électronique) qui ont connu un succès particulier ces dernières années grâce à l'augmentation de la puissance des ordinateurs.

Les travaux présentés dans cette thèse apportent des contributions dans différents domaines de recherche tels que la recherche opérationnelle et les applications des métaheuristiques en traitement d'images et en électronique.

Dans cette thèse, nous avons abordé le sujet de l'optimisation mono-objectif continue, puis nous avons exposé quelques-unes des métaheuristiques les plus connues dans la littérature. Parmi les métaheuristiques présentées, un intérêt particulier a été porté à la méthode d'Optimisation par Essaim Particulaire (OEP, ou PSO en anglais). PSO est une méthode d'optimisation stochastique qui est inspirée du comportement social d'animaux évoluant en essaim. Ce comportement social est modélisé par une équation mathématique qui revient à guider les particules durant le processus de déplacement. Le déplacement d'une particule est influencé par trois composantes : la composante d'inertie, la composante cognitive et la composante sociale. Chacune de ces composantes correspond à une partie de l'équation.

Dans PSO, les particules sont initialisées aléatoirement dans l'espace de recherche. Cependant, une telle initialisation peut entraîner un manque de diversité dans l'essaim dès le premier pas de temps, ce qui favorise une convergence prématurée vers un optimum local. La convergence prématurée est considérée comme un problème majeur de PSO. La première étape de ce travail de thèse a été d'élaborer un nouvel algorithme qui per-

met de remédier au problème de la convergence prématurée et d'améliorer les performances de l'algorithme PSO. Un nouvel algorithme d'optimisation, dénommé *PSO-2S*, a été proposé. Dans cet algorithme, trois idées principales ont été ajoutées à l'algorithme standard de PSO, sans introduire d'opérations complexes. La première idée consiste à utiliser deux types d'essaims : un essaim principal, noté $S1$, et S essaims auxiliaires, notés $S2_i$, où $1 \leq i \leq S$. La deuxième idée consiste à partitionner l'espace de recherche en S zones où les essaims auxiliaires sont initialisés. La dernière idée consiste à utiliser le concept de la répulsion électrostatique, qui permet de couvrir plus largement chaque zone de l'espace de recherche. Ces trois idées sont appliquées dans la première phase de l'algorithme *PSO-2S*, qui est la phase de construction de l'essaim principal, dont le but est d'explorer au mieux les différents optima locaux de la fonction objectif. Une recherche locale a été également ajoutée à *PSO-2S* afin d'améliorer la précision des solutions trouvées. A partir des analyses et des expérimentations faites sur de nombreuses fonctions de test, nous avons conclu que les idées utilisées dans *PSO-2S* ont pu limiter le phénomène de convergence prématurée de l'algorithme PSO. Elles ont également amélioré les performances de PSO d'une manière significative pour la plupart des fonctions de test.

Une des perspectives considérées comme intéressantes dans de nombreux travaux est l'intégration d'idées en provenance de diverses métaheuristiques, généralement sous la forme d'hybridations. En effet, tirer partie des avantages spécifiques de métaheuristiques différentes en les combinant peut conduire à une amélioration des performances et des solutions trouvées. Nous avons alors choisi d'hybrider *PSO-2S* avec l'algorithme DE (*Differential Evolution*), qui possède un opérateur de croisement facile à contrôler et efficace pour l'exploration, permettant la génération de nouvelles configurations par recombinaison de solutions existantes. Cette hybridation (*PSO-2S* avec DE) est implémentée dans un nouvel algorithme, dénommé *DEPSO-2S*. *DEPSO-2S* est une variante de *PSO-2S* qui utilise les trois idées principales sur lesquelles se base *PSO-2S* et qui sont présentées dans le paragraphe précédent. Par contre, dans *DEPSO-2S*, l'algorithme DE est utilisé pour construire l'essaim principal à la place de l'algorithme PSO. Ainsi, dans *DEPSO-2S*, une variante de DE modifiée à des fins d'adaptation à *PSO-2S* est utilisée. L'algorithme *DEPSO-2S* a été évalué sur les mêmes fonctions de test que *PSO-2S*. Il obtient également de bons résultats, l'hybridation des deux algorithmes ayant contribué à améliorer l'ancienne version de *PSO-2S*.

Ensuite, toujours dans le cadre de l'amélioration de l'algorithme PSO, nous nous

sommes intéressés à la composante sociale de l'équation modélisant le déplacement des particules, plus précisément au réseau de communication entre les particules (i.e. la *topologie* de voisinage), où les performances de PSO sont influencées par la topologie choisie. Nous avons proposé une nouvelle topologie dynamique, appelée *Dcluster*, qui est une combinaison de deux topologies (statique et dynamique). Ce travail a été évalué en utilisant différents problèmes (la plupart des fonctions définies dans CEC'05 et quelques problèmes réels très fréquents dans ce domaine). Les résultats obtenus par *Dcluster* ont été comparés à ceux obtenus via six autres topologies de la littérature de PSO. Les expériences ont montré qu'aucune des topologies étudiées ne surpasse complètement les autres, mais la topologie proposée obtient les meilleures performances pour la majorité des problèmes testés et a pu limiter le phénomène de convergence prématurée.

Notre algorithme *PSO-2S* a été appliqué dans deux domaines différents. Tout d'abord, nous avons montré que les métaheuristiques peuvent être une alternative rapide et performante pour des problèmes de segmentation d'images, telles que les images IRM cérébrales. Ensuite, *PSO-2S* a été utilisé pour résoudre un problème de dimensionnement de transistors en électronique analogique, où les performances des circuits sont très fortement dépendantes des dimensions des transistors. En effet, peu de méthodes automatiques existent pour résoudre ce type de problème. Nous avons montré que ce problème a été correctement résolu par *PSO-2S*. Ainsi, l'utilisation de méthodes d'optimisation pour ce problème est aussi une alternative crédible, tant en termes de performance que de précision.

En perspective, pour les algorithmes proposés (*PSO-2S*, *DEPSO-2S*) et la topologie *Dcluster*, des analyses expérimentales plus poussées restent à effectuer. Les variantes actuelles de *PSO-2S* obtiennent des résultats encourageants, mais de nombreuses pistes d'amélioration restent à explorer. Nous envisageons dans un premier temps d'améliorer les performances de *PSO-2S* en l'hybridant avec un algorithme de recherche locale plus performant que celui proposé dans ce travail, comme la méthode de Nelder & Mead. Pour chaque problème posé, il est important de trouver le bon réglage de paramètres, i.e. celui qui maximise les performances de l'algorithme. Cependant, trouver le bon réglage nécessite parfois beaucoup de temps, ce qui est fastidieux pour l'utilisateur. Donc, dans un second temps, nous souhaitons proposer une méthode permettant de rendre les paramètres de *PSO-2S* auto-adaptatifs, afin de faciliter son utilisation. D'autres pistes sont à explorer, telles que l'adaptation des algorithmes proposés aux plateformes de calcul parallèles, afin

de réduire le temps de calcul, ou bien l'élaboration d'une version multiobjectif de *PSO-2S*.

En perspective également, nous envisageons d'améliorer la topologie *Dcluster* que nous avons proposée. En effet, nous pourrions étudier différentes manières de générer les voisinages des particules, non seulement par une structure régulière comme nous l'avons vu dans ce travail, mais aussi par des structures auto-adaptatives, en respectant toujours les principes fondamentaux de la topologie proposée. En outre, nous pourrions pondérer les connexions entre les particules en ajoutant un poids pour chaque connexion, ce qui aura pour effet de pondérer l'influence de chaque particle sur celles auxquelles elle est connectée.

Enfin, l'application en segmentation d'images proposée au chapitre quatre pourrait être améliorée par l'utilisation d'autres critères plus complexes.

Annexe

A - Principales fonctions de test

Les principales fonctions de test sont classées en quatre groupes en fonction de leurs propriétés [Suga 05, Tu 04, Yao 99]. Ceux-ci sont définis comme suit :

- **Groupe A : Fonctions unimodales**

  1. Sphere function

$$f_1(x) = \sum_{i=1}^{d} x_i^2. \tag{4.4.1}$$

  2. Quadric function

$$f_2(x) = \sum_{i=1}^{d} \left(\sum_{j=1}^{i} x_j \right)^2. \tag{4.4.2}$$

- **Groupe B : Fonctions multimodales**

  3. Rosenbrock's function

$$f_3(x) = \sum_{i=1}^{d-1} \left(100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2 \right). \tag{4.4.3}$$

  4. Ackley's function

$$f_4(x) = -20\, exp\left(-0{,}2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2} \right) - exp\left(\frac{1}{d}\sum_{i=1}^{d} cos(2\pi x_i) \right) + 20 + e. \tag{4.4.4}$$

  5. Rastrigin's function

$$f_5(x) = \sum_{i=1}^{d} \left(x_i^2 - 10cos(2\pi x_i) + 10 \right). \tag{4.4.5}$$

  6. Weierstrass's function

$$f_6(x) = \sum_{i=1}^{d} \left(\sum_{k=0}^{k\ max} \left[a^k cos\left(2\pi b^k \left(x_i + 0{,}5 \right) \right) \right] \right) - D \sum_{k=0}^{k\ max} \left[a^k cos\left(2\pi b^k .0{,}5 \right) \right] \tag{4.4.6}$$

$$a = 0{,}5,\ b = 3,\ k\,max = 20.$$

7. Generalized penalized function

$$f_7(x) = 0{,}1 \left\{ \begin{array}{l} sin^2(3\pi x_1) \\ +\sum_{i=1}^{d-1}(x_i-1)^2\left[1+sin^2(3\pi x_{i+1})\right] \\ +(x_d-1)^2\left[1+sin^2(2\pi x_d)\right] \end{array} \right\} + \sum_{i=1}^{d} u(x_i,5,100,4) \quad (4.4.7)$$

avec :

$$u(x_i,a,k,m) = \left\{ \begin{array}{ll} k(x_i-a)^m, & x_i > a \\ 0, & -a \le x_i \le a \\ k(-x_i-a)^m, & x_i < -a \end{array} \right.$$

8. Griewank's function

$$f_8(x) = \sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1. \quad (4.4.8)$$

9. Tripod's function

$$f_9(x) = \left\{ \begin{array}{l} \frac{1-sign(x_2)}{2}(|x_1| + |x_2+50|) \\ +\frac{1+sign(x_2)}{2}\frac{1-sign(x_1)}{2}(1+|x_1+50|+|x_2-50|) \\ +\frac{1+sign(x_1)}{2}(2+|x_1-50|+|x_2-50|) \end{array} \right. \quad (4.4.9)$$

avec :

$$\left\{ \begin{array}{ll} sign(x) = -1 & si \quad x \le 0 \\ sign(x) = 1 & sinon. \end{array} \right.$$

– **Groupe C : Fonctions pivotées**

Nous utilisons la méthode de Salomon [Salo 96], qui génère une matrice orthogonale M pour faire tourner les fonctions. En effet, une nouvelle variable $y = M * x$ sera utilisée pour calculer la valeur de $fitness$, sachant que x est la variable originale. Le fait de faire tourner les fonctions ajoute une grande complexité sur les fonctions, qui deviennent très difficiles à résoudre par les algorithmes d'optimisation. Ces fonctions sont définies comme suit :

10. Rotated Quadric function

$$f_{10}(x) = \sum_{i=1}^{d} \left(\sum_{j=1}^{i} y_j \right)^2 . \tag{4.4.10}$$

11. Rotated Rosenbrock's function

$$f_{11}(x) = \sum_{i=1}^{d-1} \left(100(y_i^2 - y_{i+1})^2 + (y_i - 1)^2 \right) . \tag{4.4.11}$$

12. Rotated Ackley's function

$$f_{12}(x) = -20\, exp\left(-0{,}2\sqrt{\frac{1}{d}\sum_{i=1}^{d} y_i^2} \right) - exp\left(\frac{1}{d}\sum_{i=1}^{d} cos(2\pi y_i) \right) + 20 + e. \tag{4.4.12}$$

13. Rotated Rastrigin's function

$$f_{13}(x) = \sum_{i=1}^{d} \left(y_i^2 - 10cos(2\pi y_i) + 10 \right) . \tag{4.4.13}$$

14. Rotated Weierstrass's function

$$f_{14}(x) = \sum_{i=1}^{d} \left(\sum_{k=0}^{k_{max}} \left[a^k cos\left(2\pi b^k \left(y_i + 0{,}5 \right) \right) \right] \right) - D \sum_{k=0}^{k_{max}} \left[a^k cos\left(2\pi b^k .0{,}5 \right) \right] \tag{4.4.14}$$

$$a = 0{,}5,\ b = 3,\ k_{max} = 20.$$

15. Rotated Generalized penalized function

$$f_{15}(x) = 0.1 \left\{ \begin{array}{l} sin^2(3\pi y_1) \\ +\sum_{i=1}^{d-1} (y_i - 1)^2 \left[1 + sin^2(3\pi y_{i+1}) \right] \\ +(y_d - 1)^2 \left[1 + sin^2(2\pi y_d) \right] \end{array} \right\} + \sum_{i=1}^{d} u(y_i, 5, 100, 4) \tag{4.4.15}$$

avec :

$$u(y_i,a,k,m) = \begin{cases} k(y_i - a)^m & y_i > a \\ 0 & -a \leq y_i \leq a \\ k(-y_i - a)^m & y_i < -a \end{cases}$$

– **Groupe D : Fonctions décalées**

Pour décaler les fonctions, nous utilisons un vecteur de décalage $O = (o_1, o_2, \ldots, o_d)$, et nous ajoutons f_{bias_i} pour chaque fonction. En effet, ce décalage rend le problème plus difficile à résoudre. Les vecteurs de décalage utilisés sont repris de $CEC'2005$ [Suga 05].

Les fonctions décalées sont présentées ci-dessous :

16. Shifted Rosenbrock's function

$$f_{16}(x) = \sum_{i=1}^{d-1} \left(100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2\right) + f_{bias_2} \tag{4.4.16}$$

$$z_i = x_i - o_{2i}.$$

17. Shifted Ackley's function

$$f_{17}(x) = -20exp\left(-0{,}2\sqrt{\frac{1}{d}\sum_{i=1}^{d} z_i^2}\right) - exp\left(\frac{1}{d}\sum_{i=1}^{d} cos(2\pi z_i)\right) + 20 + e + f_{bias_3} \tag{4.4.17}$$

$$z_i = x_i - o_{3i}.$$

18. Shifted Rastrigin's function

$$f_{18}(x) = \sum_{i=1}^{d} \left(z_i^2 - 10cos(2\pi z_i) + 10\right) + f_{bias_4} \tag{4.4.18}$$

$$z_i = x_i - o_{4i}.$$

– *19. Shifted Griewank's function*

$$f_{19}(x) = \sum_{i=1}^{d} \frac{z_i^2}{4000} - \prod_{i=1}^{d} cos\left(\frac{z_i}{\sqrt{i}}\right) + 1 + f_{bias_5} \tag{4.4.19}$$

$$z_i = x_i - o_{5i}.$$

– *20. Shifted Sphere function*

$$f_{20}(x) = \sum_{i=1}^{d} x_i^2 + f_{bias_1} \tag{4.4.20}$$

$$z_i = x_i - o_{1i}.$$

B - Problèmes réels

F1 : problème « *Lennard-Jones atomic cluster* »

Le problème $Lennard - Jones\ (LJ)$ consiste, étant donné un agrégat de K atomes, représentant une molécule, à trouver les positions relatives de ces atomes dont l'énergie potentielle globale est minimale dans un espace euclidien à trois dimensions. Nous présentons le problème LJ comme suit : la molécule est constituée de K atomes, $x^i = (x^i_1, x^i_2, x^i{}_3)$ représente les coordonnées de l'atome i dans l'espace à trois dimensions, sachant que les K atomes sont situés aux positions $((x^1), \ldots, (x^K))$, où $x^i \in \mathbb{R}^3$ et $i = 1, 2, \ldots, K$. La formule de l'énergie potentielle de LJ pour une paire d'atomes (i, j) est donnée par :

$$v(r_{ij}) = \frac{1}{r_{ij}^{12}} - \frac{1}{r_{ij}^{6}} \quad \text{tels que} \quad 1 \leq i,j \leq K$$

où $r_{ij} = \left\| x^i - x^j \right\|$.

L'énergie potentielle globale est la somme des énergies résultant des interactions entre chaque couple d'atomes de la molécule. La configuration la plus stable de la molécule correspond à un minimum global d'énergie potentielle que l'on peut avoir en minimisant la fonction $V_k(x)$ dans l'équation suivante :

$$Min\ V_k(x) = \sum_{i<j} v\left(\left\| x_i - x_j \right\|\right) = \sum_{i=1}^{K-1} \sum_{j=i+1}^{K} \left(\frac{1}{\left\| x_i - x_j \right\|^{12}} - \frac{1}{\left\| x_i - x_j \right\|^{6}} \right). \tag{4.4.21}$$

Dans notre travail, nous utilisons 9 atomes dans un espace de recherche $[-2, 2]^{27}$. La valeur de l'optimum global dans ce cas est $f^* = -24{,}113360$.

F2 : problème « *Gear train design* »

$$Min\ f(x) = \left\{ \frac{1}{6{,}931} - \frac{T_d T_b}{T_a T_f} \right\}^2 = \left\{ \frac{1}{6{,}931} - \frac{x_1 x_2}{x_3 x_4} \right\}^2 \tag{4.4.22}$$

où :

- $12 \leq x_i \leq 60$ avec $i = 1, 2, 3, 4$ et x_i doit être entier ;
- T_a, T_b, T_d et T_f sont les nombres de dents des engrenages A, B, D et F, respectivement [Ali 10].

F3 : problème « *Frequency modulation sound parameter identification* »

Le problème est de déterminer les six paramètres a_1, w_1, a_2, w_2, a_3 et w_3 du modèle suivant :

$$y(t) = a_1 * sin(w_1 * t * \theta + a_2 * sin(w_2 * t * \theta + a_3 * sin(w_3 * t * \theta))),\ \theta = (2.\pi/100) \quad (4.4.23)$$

La fonction objectif est la somme des carrés des erreurs entre les données évaluées et le modèle de données. Elle est définie par :

$$f(a_1,w_1,a_2,w_2,a_3,w_3) = \sum_{t=0}^{100} (y(t) - y_0(t))^2$$

Le modèle de données est défini par l'équation suivante :

$$y_0(t) = 1{,}0 * sin(5{,}0 * t * \theta + 1{,}5 * sin(4{,}8 * t * \theta + 2{,}0 * sin(4{,}9 * t * \theta))) \quad (4.4.24)$$

où : $-6{,}4 \leq a_i, w_i \leq 6{,}35$ avec $i = 1, 2, 3$. [Ali 10]

F4 : problème « *Spread spectrum radar poly-phase code design* »

$$Min\ \ f(X) = max\{f_1(X), \ldots, f_{2m}(X)\} \quad (4.4.25)$$

$$X = \{(x_1, \ldots, x_n) \in R^n \mid 0 \leq x_j \leq 2\pi, j = 1{,}2, \ldots, n\} \text{ et } m = 2n - 1,$$

où :

$$f_{2i-1}(x) = \sum_{j=i}^{n} cos\left(\sum_{k=|2i-j-1|+1}^{j} x_k\right) \quad i = 1{,}2, \ldots, n\,;$$

$$f_{2i}(x) = 0{,}5 + \sum_{j=i+1}^{n} cos\left(\sum_{k=|2i-j|+1}^{j} x_k\right) \quad i = 1{,}2, \ldots, n-1\,;$$

$$f_{m+i}(X) = -f_i(X), i = 1, 2, \ldots, m.$$

F5 : problème « *Compression spring design* »

$$Min\ f(x) = \pi^2 \frac{x_2 x_3^2 (x_1 + 1)}{4} \quad (4.4.26)$$

où $x_1 \in \{1, \ldots, 70\}$ avec une granularité 1, $x_2 \in [0{,}6; 3{,}0]$ et $x_3 \in \{0{,}207; \ldots; 0{,}5\}$ avec une granularité 0,0001. La meilleure solution $x^* = (x_1, x_2, x_3)$ connue est $(7;\ 1{,}386599591;\ 0{,}292)$ avec sa valeur de *fitness* $f^* = 2{,}6254214578$. Ce problème a cinq contraintes que l'on peut trouver dans [Cler 06].

F6 : « *Perm function* »

$$Min\ f(x) = \sum_{k=1}^{5} \left[\sum_{i=1}^{5} \left(i^k + \beta \right) \left\{ (x_i/i)^k - 1 \right\} \right]^2 \tag{4.4.27}$$

où $x \in [-5, 5]$ et β est une constante fixée à 10 [Cler 12]. L'optimum global de cette fonction est $(1, 2, 3, 4, 5)$ et sa valeur de *fitness* f^* est égale à 0.

F7 : problème « *Mobile network design* »

Dans ce problème, la difficulté consiste à déterminer efficacement les emplacements des contrôleurs de stations de base (BSC), centres de commutation mobiles (MSC), ainsi que leurs liaisons de connexion pour un site donné de stations de base. Ce problème est complexe à décrire, pour plus d'informations le lecteur peut se référer à [El S 09]. L'espace de recherche est constitué de variables binaires (38 variables) et de variables continues (4 variables). Par conséquent, le problème a 42 dimensions.

F8 : problème « *Pressure Vessel design* »

$$Min\ f(x) = 0{,}6224 x_1 x_3 x_4 + 1{,}7781 x_2 x_3^2 + 3{,}1661 x_1^2 x_4 + 19{,}84 x_1^2 x_3 \tag{4.4.28}$$

sous réserve que :

$$g_1(x) = 0{,}0193 x_3 - x_1 \leq 0$$

$$g_2(x) = 0{,}00954 x_3 - x_2 \leq 0$$

$$g_3(x) = 750 \times 1728 - \pi x_3^2 (x_4 + \tfrac{4}{3} x_3) \leq 0$$

où $x_1 \in \{0{,}0625;\ \ldots;\ 12{,}5\}$ avec une granularité 0,0625, $x_2 \in \{0{,}625;\ \ldots;\ 12{,}5\}$ avec une granularité 0,0625, $x_3 \in]0,\ 240]$ et $x_4 \in]0,\ 240]$. La meilleure solution est $(1{,}125;\ 0{,}625;\ 58{,}2901554;\ 43{,}6926562)$ et $f^* = 7{,}197.72893$ [Kann 94].

F9 : problème « *Welded Beam design* »

$$Min\ f(x) = 1{,}10471 x_1^2 x_2 + 0{,}04811 x_3 x_4 (14{,}0 + x_2) \tag{4.4.29}$$

sous réserve que :

$$g_1(x) = \tau(x) - 13000 \leq 0, g_2(x) = \sigma(x) - 30000 \leq 0, g_3(x) = x_1 - x_4 \leq 0$$

$$g_4(x) = 6000 - P_c(x) \leq 0, g_5(x) = 0{,}125 - x_1 \leq 0, g_6(x) = \delta(x) - 0{,}25 \leq 0$$

$$g_7(x) = 0{,}10471x_1^2 + 0{,}04811x_3x_4(14{,}0 + x_2) - 5{,}0 \leq 0$$

où :

$$\tau(x) = \sqrt{(\tau')^2 + 2\tau'\tau''\frac{x_2}{2R} + (\tau'')^2}, \tau' = \frac{6000}{\sqrt{2}x_1x_2}, \tau'' = \frac{MR}{J}$$

$$M = 6000(14 + \frac{x_2}{2}), R = \sqrt{\frac{x_2^2}{4} + (\frac{x_1+x_3}{2})^2}, J = 2\left\{\sqrt{2}x_1x_2\left[\frac{x_2^2}{12} + (\frac{x_1+x_3}{2})^2\right]\right\}$$

$$P_c(x) = \frac{4{,}013(30\times10^6)\sqrt{\frac{x_3^2x_4^6}{36}}}{196}\left(1 - \frac{x_3\sqrt{\frac{30\times10^6}{4(12\times10^6)}}}{28}\right), \sigma(x) = \frac{504000}{x_4x_3^2}, \delta(x) = \frac{2{,}1952}{x_3^3x_4}$$

$x_1, x_4 \in [0{,}1;\ 2{,}0]\,, x_2, x_3 \in [0{,}1;\ 10{,}0]$, $x^* = (0{,}205730;\ 3{,}470489;\ 9{,}036624;\ 0{,}205729)$ et $f^* = 1{,}724852$ [Rags 76].

F10 : problème « *Speed Reducer design* »

$$\begin{aligned} Min\ f(x) = {} & 0{,}7854x_1x_2^2(3{,}3333x_3^2 + 14{,}9334x_3 - 43{,}0934) - 1{,}508x_1(x_6^2 + x_7^2) \\ & +7{,}4777(x_6^3 + x_7^3) + 0{,}7854(x_4x_6^2 + x_5x_7^2) \end{aligned} \tag{4.4.30}$$

sous réserve que :

$$g_1(x) = \frac{27}{x_1x_2^2x_3} \leq 0, g_2(x) = \frac{397{,}5}{x_1x_2^2x_3^2} \leq 0, g_3(x) = \frac{1{,}93x_4^3}{x_2x_3x_6^4} \leq 0, g_4(x) = \frac{1{,}93x_5^3}{x_2x_3x_7^4} \leq 0$$

$$g_5(x) = \frac{1{,}0}{110x_6^3}\sqrt{\left(\frac{745x_4}{x_2x_3}\right)^2 + 16{,}9\times10^6} - 1 \leq 0, g_6(x) = \frac{x_2x_3}{40} - 1 \leq 0, g_7(x) = \frac{5x_2}{x_1} - 1 \leq 0$$

$$g_8(x) = \frac{1{,}0}{85x_7^3}\sqrt{\left(\frac{745x_5}{x_2x_3}\right)^2 + 157{,}5\times10^6} - 1 \leq 0, g_9(x) = \frac{x_1}{12x_2} - 1 \leq 0$$

$$g_{10}(x) = \frac{1{,}5x_6+1{,}9}{x_4} - 1 \leq 0, g_{11}(x) = \frac{1{,}1x_7+1{,}9}{x_5} - 1 \leq 0$$

où $2{,}6 \leq x_1 \leq 3{,}6$; $0{,}7 \leq x_2 \leq 0{,}8$; $17 \leq x_3 \leq 28$; $7{,}3 \leq x_4 \leq 8{,}3$; $7{,}8 \leq x_5 \leq 8{,}3$; $2{,}9 \leq x_6 \leq 3{,}9$; $5{,}0 \leq x_7 \leq 5{,}5$, $x^* = (3{,}5;\ 0{,}7;\ 17;\ 7{,}3;\ 7{,}8;\ 3{,}350214;\ 5{,}286683)$ et $f^* = 2{,}996.348165$ [Goli 73].

F11 : problème de contraintes 1

$$Min\ f(x) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} {x_i}^2 - \sum_{i=1}^{13} x_i \tag{4.4.31}$$

sous réserve que :

$$g_1(x) = 2x_1 + 2x_2 + x_10 + x_11 - 10 \leq 0, g_2(x) = 2x_1 + 2x_3 + x10 + x12 - 10 \leq 0$$
$$g_3(x) = 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0, g_4(x) = -8x_1 + x_{10} \leq 0, g_5(x) = -8x_2 + x_{11} \leq 0$$
$$g_6(x) = -8x_3 + x_{12} \leq 0, g_7(x) = -2x_4 - x_5 + x_{10} \leq 0,\ g_8(x) = -2x_6 - x_7 + x_{11} \leq 0$$
$$g_9(x) = -2x_8 - x_9 + x_{12} \leq 0$$

où : $0 \leq x_i \leq 1$ $(i = 1, \ldots, 9, 13)$ et $0 \leq x_i \leq 100$ $(i = 10, 11, 12)$. L'optimum global est $(1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)$ et $f^* = -15$.

F12 : problème de contraintes 2

$$\begin{aligned} Min\ f(x) = {} & x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 \\ & +2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45 \end{aligned} \tag{4.4.32}$$

sous réserve que :

$$g_1(x) = -105 + 4x_1 + 5x_2 - 3x_7 + 9x_8 \leq 0$$

$$g_2(x) = 10x_1 - 8x_2 - 17x_7 + 2x_8 \leq 0$$

$$g_3(x) = -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \leq 0$$

$$g_4(x) = 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^2 - 7x_4 - 120 \leq 0$$

$$g_5(x) = 5x_1^2 + 8x_2 + (x3 - 6)^2 - 2x_4 - 40 \leq 0$$

$$g_6(x) = x_1^2 + 2(x_2 - 2)^2 - 2x_1x_2 + 14x_5 - 6x_6 \leq 0$$

$$g_7(x) = 0{,}5(x_1 - 8)^2 + 2(x_2 - 4)^2 + 3x_5^2 - x_6 - 30 \leq 0$$

$$g_8(x) = -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_{10} \leq 0$$

où :

- $-10 \leq x_i \leq 10$ pour tout $(i = 1, 2, \ldots, 10)$,
- $x^* = (2{,}171996;\ 2{,}363683;\ 8{,}773926;\ 5{,}095984;\ 0{,}9906548;\ 1{,}430574;\ 1{,}321644;\ 9{,}828726;\ 8{,}280092;\ 8{,}375927)$
- et $f(x^*) = 24{,}3062091$.

Références bibliographiques

[Alba 05] E. Alba. *Parallel Metaheuristics : A New Class of Algorithms*. John Wiley & sons, 2005.

[Ali 10] M. Ali, M. Pant, and V. P. Singh. "Two Modified Differential Evolution Algorithms and their Applications to Engineering Design Problems". *World Journal of Modeling and Simulation*, Vol. 6, No. 1, pp. 72–80, 2010.

[Ange 98] P. Angeline. "Evolutionary optimization versus particle swarm optimization : Philosophy and performance differences". pp. 601–610, 1998.

[Avri 76] M. Avriel. *Nonlinear Programming : Analysis and Methods*. Prentice-Hall, Englewood Cliffs, NJ, 1976.

[Azen 92] R. Azencott. *Simulated Annealing : Parallelization Techniques*. Wiley-Interscience, 1992.

[Basi 75] V. R. Basili and A. J. Turner. "Iterative enhancement : A practical technique for software development". *IEEE Transactions on Software Engineering*, Vol. 1, No. 4, pp. 390–396, 1975.

[Bask 04] S. Baskar and P. Suganthan. "A Novel Concurrent Particle Swarm Optimization". In : *In Proceedings IEEE of the Congress on Evolutionary Computation (CEC 2004)*, pp. 792–796, 2004.

[Bast 09] C. J. A. Bastos-Filho, D. F. Carvalho, E. M. N. Figueiredo, and P. B. C. de Miranda. "Dynamic Clan Particle Swarm Optimization". In : *Intelligent Systems Design and Applications, 2009. ISDA '09. Ninth International Conference on*, pp. 249–254, 2009.

[Ben 06] S. Ben Salem, M. Fakhfakh, D. S. Masmoudi, M. Loulou, P. Loumeau, and N. Masmoudi. "A high performances CMOS CCII and high frequency applications". *Analog Integr. Circuits Signal Process.*, Vol. 49, No. 1, pp. 71–78, 2006.

[Berg 04] F. Van den Bergh and A. P. Engelbrecht. "A cooperative approach to particle swarm optimization". *IEEE Transactions on Evolutionary Computation*, Vol. 8, No. 3, pp. 225–239, 2004.

[Bert 95] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

[Beye 01] H. G. Beyer. *The theory of evolution strategies*. Springer, first edition, 2001.

[Boya 98] J. A. Boyan and A. W. Moore. "Learning evaluation functions for global optimization and boolean satisfiability". In : *AAAI-98*, pp. 3–10, Madison, Wisconsin, 1998.

[Bran 02] J. Branke. *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic, 2002.

[Bran 12] J. Branke. "BrainWeb : Simulated Brain Database". http://brainweb.bic.mni.mcgill.ca/brainweb, 2012.

[Bres 06] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer. "Self-Adapting Control Parameters in Differential Evolution : A Comparative Study on Numerical Benchmark Problems". *Trans. Evol. Comp*, Vol. 10, No. 6, pp. 646–657, 2006.

[Bres 07] J. Brest, B. Boskovic, S. Greiner, V. Zumer, and M. S. Maucec. "Performance comparison of self-adaptive and adaptive differential evolution algorithms". *Soft Comput.*, Vol. 11, No. 7, pp. 617–629, 2007.

[Cern 85] V. Cerny. "Thermodynamical Approach to the Travelling Salesman Problem : An Efficient Simulation Algorithm". *Journal of Optimization Theory and Applications*, Vol. 45, pp. 41–51, 1985.

[Chak 06] U. K. Chakraborty, A. Konar, and S. Das. "Differential Evolution with Local Neighborhood". pp. 2042–2049, IEEE Press, Vancouver, BC, Canada, July 2006.

[Chat 06] A. Chatterjee and P. Siarry. "Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization". *Computers and Operations Research*, Vol. 33, pp. 859–871, 2006.

[Chat 10] A. Chatterjee, M. Fakhfakh, and P. Siarry. "Design of second-generation current conveyors employing bacterial foraging optimization". *Microelectron. J.*, Vol. 41, No. 10, pp. 616–626, 2010.

[Chen 10] D. Chen, C. Zhao, and H. Zhang. "An improved cooperative particle swarm optimization and its application". *Neural Computing and Applications*, Vol. 20, No. 2, pp. 171–182, 2010.

[Cler 02] M. Clerc and J. Kennedy. "The particle swarm : explosion, stability, and convergence in multi-dimensional complex space". *IEEE Transactions on Evolutionary Computation*, Vol. 6, pp. 58–73, 2002.

[Cler 03] M. Clerc. "TRIBES - Un exemple d'optimisation par essaims particulaires sans paramètres de contrôle". In : *Conférence OEP'03*, Paris, France, Octobre 2003.

[Cler 06] M. Clerc. *Particle Swarm Optimization*. ISTE (International Scientific and Technical Encyclopaedia), 2006.

[Cler 12] M. Clerc *et al.* "The *Particle Swarm Central* website". http://www.particleswarm.info, 2012.

[Coll 02] Y. Collette and P. Siarry. *Optimisation multiobjectif*. Eyrolles, 2002.

[Coll 98] D. L. Collins, A. P. Zijdenbos, V. Kollokian, J. G. Sled, N. J. Kabani, C. J. Holmes, and A. C. Evans. "Design and Construction of a Realistic Digital Brain Phantom". *IEEE Transactions on Medical Imaging*, Vol. 17, No. 3, pp. 463–468, 1998.

[Colo 91] A. Colorni, M. Dorigo, and V. Maniezzo. "Distributed Optimization by Ant Colonies". In : *Proceedings of the First European Conference on Artificial Life*, pp. 134–142, MIT Press, Paris, France, December 1991.

[Conn 96] A. R. Conn, K. C. Paula, A. H. Ruud, L. M. Gregory, and V. Chandu. "Optimization of Custom MOS Circuits by Transistor Sizing". pp. 174–180, 1996.

[Conw 98] J. H. Conway and N. J. A. Sloane. *Sphere Packings, Lattices and Groups*. Springer-Verlag, third Ed., 1998.

[Coor 08] Y. Cooren. *Perfectionnement d'un algorithme adaptatif d'Optimisation par Essaim Particulaire. Applications en génie médical et en électronique*. PhD thesis, Université Paris-Est Créteil, 2008.

[Corm 90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms, chapitre 16 : Greedy Algorithms*. MIT Press and McGraw-Hill, 1st Ed., 1990.

[Cour 94] J. P. Courat, G. Raynaud, I. Mrad, and P. Siarry. "Electronic component model minimization based on Log Simulated Annealing". pp. 790–795, 1994.

[Creu 83] M. Creutz. "Microcanonical Monte Carlo simulation". *Physical Review Letters*, Vol. 50, No. 19, pp. 1411–1415, 1983.

[Cuev 10] E. Cuevas, D. Zaldivar, and M. Pérez-Cisneros. "A novel multi-threshold segmentation approach based on differential evolution optimization". *Expert Systems with Applications*, Vol. 37, No. 7, pp. 5265–5271, 2010.

[Das 08] S. Das, A. Abraham, and A. Konar. "Particle swarm optimization and differential evolution algorithms : technical analysis, applications and hybridization perspectives". *Studies in Computational Intelligence (SCI)*, Vol. 116, No. , pp. 1–38, 2008.

[DeVo 96] R. A. DeVore and V. N. Temlyakov. "Some remarks on greedy algorithms". *Advances in Computational Mathematics*, Vol. 5, No. 1, pp. 173–187, 1996.

[Dori 05] M. Dorigo and C. Blum. "Ant colony optimization theory : a survey". *Theoretical Computer Science*, Vol. 344, No. 2-3, pp. 243–278, 2005.

[Dori 96] M. Dorigo, V. Maniezzo, and A. Colorni. "Ant System : optimization by a colony of cooperating agents". *IEEE Transactions on Systems, Man, and Cybernetics, Part B : Cybernetics*, Vol. 26, No. 1, pp. 29–41, 1996.

[Dro 03] J. Dréo, A. Pétrowski, P. Siarry, and E. Taillard. *Métaheuristiques pour l'Optimisation Difficile*. Eyrolles, 2003.

[Eber 00] R. Eberhart and Y. Shi. "Comparing inertia weights and constriction factors in particle swarm optimization". In : *Proceedings of the 6th IEEE Congress on Evolutionary Computation*, pp. 84–88, IEEE Press, 2000.

[Eber 01] R. C. Eberhart and Y. Shi. "Tracking and optimizing dynamic systems with particle swarms". In : *Proceedings of the 2001 IEEE Congress on Evolutionary Computation*, pp. 94–100, IEEE Press, Piscataway, NJ, 2001.

[Eber 96] R. Eberhart, P. Simpson, and R. Dobbins. *Computational Intelligence PC Tools*. AP Professional, 1996.

[El D 12a] A. El Dor, M. Clerc, and P. Siarry. "A multi-swarm PSO using charged particles in a partitioned search space for continuous optimization". *Computational Optimization and Applications*, Vol. 53, No. 1, pp. 271–295, 2012.

[El D 12b] A. El Dor, M. Clerc, and P. Siarry. "Hybridization of Differential Evolution and Particle Swarm Optimization in a New Algorithm : DEPSO-2S". In : *Proceedings of the 2012 International Conference on Swarm and Evolutionary Computation, ICAISC (SIDE-EC)*, pp. 57–65, Springer-Verlag, Zakopane, Poland, April 29-May 3, 2012.

[El S 09] A. A. El-Saleh, M. Ismail, R. Viknesh, C. C. Mark, and M. L. Chan. "Particle swarm optimization for mobile network design". *IEICE Electronics Express*, Vol. 6, No. 17, pp. 1219–1225, 2009.

[Fabr 98] A. Fabre, O. Saaid, W. F., and C. Boucheron. "High-frequency high-Q BiCMOS current-mode bandpass filter and mobile communication application". *IEEE Journal of Solid State Circuits*, Vol. 33, No. 4, pp. 614–625, 1998.

[Fan 04] S. Fan, Y. Liang, and E. Zahara. "Hybrid simplex search and particle swarm optimization for the global optimization of multimodal functions". *Engineering Optimization*, Vol. 36, pp. 401–418, 2004.

[Fish 22] R. A. Fisher. "On the interpretation of χ^2 from contingency tables, and the calculation of P". *Journal of the Royal Statistical Society*, Vol. 85, No. 1, pp. 87–94, 1922.

[Foge 62] L. J. Fogel. "Autonomous automata". *Industrial Research Magazine*, Vol. 4, No. 2, pp. 14–19, 1962.

[Foge 66] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley & sons, first edition, 1966.

[Fras 57] A. S. Fraser. "Simulation of genetic systems by automatic digital computers". *Australian Journal of Biological Science*, Vol. 10, pp. 484–491, 1957.

[Giel 91] G. Gielen and W. Sansen. *Symbolic Analysis for Automated Design of Analog Integrated Circuits*. Kluwer Academic Publishers, 1991.

[Glov 86] F. Glover. "Future paths for integer programming and links to artificial intelligence". *Computers and Operations Research*, Vol. 13, pp. 533–549, 1986.

[Glov 97] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[Gold 87] D. E. Goldberg and J. Richardson. "Genetic algorithms with sharing for multimodal function optimization". In : *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pp. 41–49, Lawrence Erlbaum Associates., Mahwah, NJ, USA, 1987.

[Goli 73] J. Golinski. "An Adaptive Optimization System Applied to Machine Synthesis". *Journal of Engineering for Industry, Transactions of the ASME*, Vol. 8, No. 4, pp. 419–436, 1973.

[Gonz 07] R. C. Gonzalez and R. E. Woods. *Digital image processing*. Prentice Hall, third Ed., 2007.

[Goss 89] S. Goss, S. Aron, J. L. Deneubourg, and J. M. Pasteels. "Self-organized shortcuts in the Argentine ant". *Naturwissenschaften*, Vol. 76, No. 12, pp. 579–581, 1989.

[Grae 01] H. E. Graeb, S. Zizala, J. Eckmueller, and K. Antreich. "The Sizing Rules Method for Analog Integrated Circuit Design.". pp. 343–349, 2001.

[Hast 70] W. K. Hastings. "Monte-Carlo sampling method using Markov chains and their applications". *Biometrika*, Vol. 57, 1970.

[Holl 75] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.

[Homa 94] A. Homaifar, C. X. Qi, and S. H. Lai. "Constrained Optimization Via Genetic Algorithms". *Simulation*, Vol. 62, No. 4, pp. 242–253, 1994.

[Hsie 09] S. T. Hsieh, T. Y. Sun, C. C. Liu, and S. J. Tsai. "Efficient population utilization strategy for particle swarm optimizer". pp. 444–456, IEEE Press, Piscataway, NJ, USA, 2009.

[Iqba 06] M. Iqbal and M. A. Montes de Oca. "An estimation of distribution particle swarm optimization algorithm". In : *Proceedings of the 5th international conference on Ant Colony Optimization and Swarm Intelligence*, pp. 72–83, Springer-Verlag, Berlin, Heidelberg, 2006.

[Jans 05] S. Janson and M. Middendorf. "A hierarchical particle swarm optimizer and its adaptive variant". *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, Vol. 35, No. 6, pp. 1272–1282, 2005.

[Kann 94] B. Kannan and S. Kramer. "An augmented Lagrange multiplier based method for mixed integer discrete continuous optimization and its applications to mechanical design". *Journal of Mechanical Design*, Vol. 116, No. 2, pp. 318–320, 1994.

[Kapu 85] J. N. Kapur, P. K. Sahoo, and A. C. K. Wong. "A new method for gray-level picture thresholding using the entropy of the histogram". *Computer Vision, Graphics and Image Processing*, Vol. 29, No. 3, pp. 273–285, 1985.

[Kenn 01] J. Kennedy, R. Eberhart, and Y. Shi. *Swarm Intelligence*. Morgan Kaufmann Academic Press, 2001.

[Kenn 02] J. Kennedy and R. Mendes. "Population Structure and Particle Swarm Performance". In : *Proceedings of the 2002 IEEE Congress on Evolutionary Computation, CEC'02*, pp. 1671–1676, Honolulu, HI, USA, 2002.

[Kenn 95] J. Kennedy and R. C. Eberhart. "Particle Swarm Optimization". In : *Proceedings of the IEEE International Conference on Neural Networks IV*, pp. 1942–1948, Perth, Australia, November 1995.

[Kenn 98] J. Kennedy. "The behavior of particles". In : *Proceedings of the 7th Conference on Evolutionary Computation*, pp. 581–589, LNCS, Springer, 1998.

[Kenn 99] J. Kennedy. "Small worlds and mega-minds : effects of neighborhood topology on particle swarm performance". In : *Proceedings of the 1999 Congress on Evolutionary Computation, CEC'99*, pp. 1931–1938, Washington, DC USA, 1999.

[Kirk 83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by Simulated Annealing". *Science*, Vol. 220, 4598, pp. 671–680, 1983.

[Kitt 86] J. Kittler and J. Illingworth. "Minimum error thresholding". *Pattern Recognition*, Vol. 19, No. 1, pp. 41–47, 1986.

[Koza 89] J. R. Koza. "Hierarchical genetic algorithms operating on populations of computer programs". pp. 768–774, Morgan Kaufmann, Detroit, Michigan, USA, 1989.

[Koza 90] J. R. Koza. "Genetic programming : A paradigm for genetically breeding populations of computer programs to solve problems". Tech. Rep., Technical Report STANCS-90-1314, Stanford University, Department of Computer Science, 1990.

[Krin 02] T. Krink, J. S. Vesterstrom, and J. Riget. "Particle swarm optimisation with spatial particle extension". In : *Proc. IEEE Congress on Evolutionary Computation*, pp. 1474–1479, Honolulu, Hawaii USA, May 2002.

[Lamp] J. Lampinen and I. Zelinka. "Mixed Integer-Discrete-Continuous Optimization By Differential Evolution - Part 1 : the optimization method". In : *Proceedings of MENDEL'99 - 5th International Mendel Conference on Soft Computing*.

[Lane 08] J. Lane, A. Engelbrecht, and J. Gain. "Particle swarm optimization with spatially meaningful neighbours". In : *Proceedings of the 2008 IEEE Swarm Intelligence Symposium. Piscataway, NJ : IEEE*, pp. 1–8, Press, 2008.

[Lepa 10] J. Lepagnot, A. Nakib, H. Oulhadj, and P. Siarry. "A new multiagent algorithm for dynamic continuous optimization". *International Journal of Applied Metaheuristic Computing*, Vol. 1, No. 1, pp. 16–38, 2010.

[Lian 06] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar. "Comprehensive Learning Particle Swarm Optimizer for Global Optimization of Multimodal Functions". *IEEE Transactions on Evolutionary Computation*, Vol. 10, No. 3, pp. 281–295, 2006.

[Liu 05] J. Liu and J. Lampinen. "A Fuzzy Adaptive Differential Evolution Algorithm". *Soft Computing*, Vol. 9, No. 6, pp. 448–462, 2005.

[Loul 02] M. Loulou, S. Ait Ali, M. Fakhfakh, and N. Masmoudi. "An optimized methodology to design CMOS operational amplifier". pp. 14–16, Beirut, Lebanon, 2002.

[Mann 47] H. B. Mann and D. R. Whitney. "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other". *The Annals of Mathematical Statistics*, Vol. 18, No. 1, pp. 50–60, 1947.

[Mede 94] F. Medeiro, R. Rodríguez-Macías, F. V. Fernández, R. Domínguez-Castro, J. L. Huertas, and A. Rodríguez-Vázquez. "Global design of analog cells using statistical optimization techniques". *Analog Integrated Circuits and Signal Processing*, Vol. 6, No. 3, pp. 179–195, 1994.

[Mend 03] R. Mendes, J. Kennedy, and J. Neves. "Watch thy neighbor or how the swarm can learn from its environment". In : *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, pp. 88–94, Indianapolis, Indiana, USA, 2003.

[Mend 04] R. Mendes, J. Kennedy, and J. Neves. "The Fully Informed Particle Swarm : Simpler, Maybe Better". *IEEE Trans. Evolutionary Computation*, Vol. 8, No. 3, pp. 204–210, 2004.

[Metr 53] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. "Equation of State Calculations by Fast Computing Machines". *The Journal of Chemical Physics*, Vol. 21, No. 6, pp. 1087–1092, 1953.

[Mira 02] V. Miranda and N. Fonseca. "New evolutionary particle swarm algorithm applied to voltage/VAR control". In : *Proceedings of the 14th Power Systems Computation Conference*, pp. 1–6, Sevilla, Spain, 2002.

[Naki 07] A. Nakib. *Conception de métaheuristiques d'optimisation pour la segmentation d'images. Application à des images biomédicales*. PhD thesis, Université Paris-Est Créteil, Décembre 2007.

[Neld 65] J. A. Nelder and R. Mead. "A Simplex Method for Function Minimization". *The Computer Journal*, Vol. 7, No. 4, pp. 308–313, 1965.

[Niu 07] B. Niu, Y. Zhu, X. He, and H. Wu. "MCPSO : A multi-swarm cooperative particle swarm optimizer". *Applied Mathematics and Computation*, Vol. 185, 2007.

[Noce 00] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 2000.

[Okam 95] Y. Okamoto and U. H. E. Hansmann. "Thermodynamics of helix-coil transitions studied by multicanonical algorithms". *J. Phys. Chem.*, No. 99, pp. 11276–11287, 1995.

[Otsu 79] N. A. Otsu. "A Threshold Selection Method from Gray Level Histograms". *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 9, No. 1, pp. 62–66, 1979.

[Ozca 99] E. Ozcan and C. Mohan. "Particle Swarm Optimization : surfing the waves". In : *Proceeding of the 1999 IEEE Congress on Evolutionary Computation*, pp. 1939–1944, IEEE Press, Washington, DC, USA, 1999.

[Pars 04] K. E. Parsopoulos and M. N. Vrahatis. "UPSO : A Unified Particle Swarm Optimization Scheme". pp. 868–873, VSP International Science Publishers, Zeist, Netherlands, 2004.

[Pass 02] K. M. Passino. "Biomimicry of Bacterial Foraging for distributed optimization and control". *IEEE Control Systems Magazine*, Vol. 22, No. 3, pp. 52–67, 2002.

[Peer 03] E. S. Peer, A. P. Engelbrecht, and F. Van Den Bergh. "Using neighborhoods with the guaranteed convergence PSO". In : *Proceedings of the IEEE Swarm Intelligence Symposium 2003 (SIS'03)*, pp. 235–242, IEEE Press, Indianapolis, Indiana, USA, 2003.

[Pera 03] T. Peram, K. Veeramachaneni, and C. K. Mohan. "Fitness-distance-ratio based particle swarm optimization". In : *Proc. IEEE Swarm Intelligence Symposium*, pp. 174–181, Indianapolis, Indiana, USA, 2003.

[Pric 05] K. Price, R. M. Storn, and J. A. Lampinen. *Differential Evolution : A Practical Approach to Global Optimization (Natural Computing Series)*. Springer, 2005.

[Qin 05] A. K. Qin and P. N. Suganthan. "Self-adaptive differential evolution algorithm for numerical optimization". In : *2005 IEEE Congress on Evolutionary Computation CEC2005*, pp. 1785–1791, IEEE Press, 2005.

[Rags 76] K. Ragsdell and D. Phillips. "Optimal Design of a Class of Welded Structures using Geometric Programming". *Journal of Engineering for Industry, Transactions of the ASME*, Vol. 98, No. 3, pp. 1021–1025, 1976.

[Rajp 07] S. Rajput and S. Jamuar. "Advanced Applications Of Current Conveyors : a tutorial". *Journal of Active and Passive Electronic devices*, Vol. 2, pp. 143–164, 2007.

[Rech 65] I. Rechenberg. "Cybernetic solution path of an experimental problem". Tech. Rep., Library translation 1122, Ministry of Aviation, Royal Air Force Establishment (UK), 1965.

[Rich 03] M. Richards and D. Ventura. "Dynamic Sociometry in Particle Swarm Optimization". In : *Joint Conference on Information Sciences*, pp. 1557–1560, Cary, North Carolina USA, 2003.

[Robi 02] J. Robinson, S. Sinton, and Y. Rahmat-Samii. "Particle Swarm, Genetic Algorithm, and Their Hybrids : Optimization of a Profiled Corrugated Horn Antenna". In : *Proceedings of the IEEE International Symposium on Antennas and Propogation*, pp. 314–317, San Antonio, Texas, États-Unis, 2002.

[Safa 09] E. Safavieh, A. Gheibi, M. Abolghasemi, and A. Mohades. "Particle swarm optimization with Voronoi neighborhood". In : *Proceedings of the 14th International CSI Computer Conference (CSICC2009)*, pp. 397–402, Tehran, Iran, 2009.

[Saho 88] P. K. Sahoo, S. Soltani, A. K. C. Wong, and Y. Chen. "A survey of thresholding techniques". *Computer Vision, Graphics, and Image Processing*, Vol. 41, No. 2, pp. 233–260, 1988.

[Salo 96] R. Salomon. "Reevaluating genetic algorithm performance under coordinate rotation of benchmark functions". *BioSystems*, Vol. 39, pp. 263–278, 1996.

[Schm 00] H. Schmid. "Approximating the universal active element". *IEEE Transactions on Circuits and Systems-II : Analog and Digital Signal Processing*, Vol. 47, No. 11, pp. 1160–1169, 2000.

[Schr 98] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1998.

[Sedr 70] A. Sedra and K. Smith. "A second generation current conveyor and its applications". *IEEE Transactions on Circuits Theory*, Vol. 17, pp. 132–134, 1970.

[Sedr 90] A. Sedra, G. Robert, and F. Gohln. "The current conveyor : History, progress and new results". In : *IEEE Proc. Part G*, pp. 78–87, 1990.

[Seev 00] E. Seevinck, E. Vittoz, M. Du Plessis, T. Joubert, and W. Beetge. "CMOS translinear circuits for minimum supply voltage". *IEEE Transactions on Circuits and Systems-II : Analog and Digital Signal Processing*, Vol. 47, No. 12, pp. 1560–1564, 2000.

[Sezg 04] M. Sezgin and B. Sankur. "Survey over image thresholding techniques and quantitative performance evaluation". *Journal of Electronic Imaging*, Vol. 13, No. 1, pp. 146–165, 2004.

[Shel 07] P. Shelokar, P. Siarry, V. Jayaraman, and B. Kulkarni. "Particle swarm and ant colony algorithms hybridized for improved continuous optimization". *Applied Mathematics and Computation*, Vol. 188, No. 1, pp. 129–142, 2007.

[Shi 99] Y. Shi and E. R. C. "Empirical study of particle swarm optimization". Vol. 3, pp. 1945–1950, 1999.

[Silv 96] F. Silveira, D. Flandre, and P. G. A. Jespers. "A GM/ID based methodology for the design of CMOS analog circuits and its application to the synthesis of a SOI micropower OTA". *IEEE Journal of Solid State Circuits*, Vol. 31, No. 9, 1996.

[Stor 97] R. Storn and K. Price. "Differential Evolution - A Simple and Efficient Heuristic for global Optimization over Continuous Spaces". *Journal of Global Optimization*, Vol. 11, No. 4, pp. 341–359, 1997.

[Suga 05] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. P. Chen, A. Auger, and S. Tiwari. "Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization". Tech. Rep., Nanyang Technological University, Singapore, 2005.

[Suga 99] P. N. Suganthan. "Particle swarm optimiser with neighbourhood operator". In : *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 99*, pp. 1958–1962, Washington, DC USA, 1999.

[Synd 90] W. Synder and G. Bilbro. "Optimal thresholding : A new approach". *Pattern Recognition Letters*, Vol. 11, No. 12, pp. 803–810, 1990.

[Talb 02] E. G. Talbi. "A Taxonomy of Hybrid Metaheuristics". *Journal of Heuristics*, Vol. 8, No. 5, pp. 541–564, 2002.

[Teo 06] J. Teo. "Exploring dynamic self-adaptive populations in differential evolution". *Soft Computing*, Vol. 10, No. 8, pp. 673–686, May 2006.

[Tlel 10] E. Tlelo-Cuautle, C. Sánchez-López, and D. Moro-Frías. "Symbolic analysis of (MO)(I)CCI(II)(III)-based analog circuits". *International Journal of Circuit Theory and Applications*, Vol. 38, No. 6, pp. 649–659, 2010.

[Toum 93] C. Toumazou, F. Lidgey, and D. Haigh. "Analog Integrated Circuits : The current mode approach". *IEEE Transactions on circuits and systems, series*, Vol. 2, 1993.

[Trel 03] I. C. Trelea. "The particle swarm optimization algorithm : convergence analysis and parameter selection". *Information Processing Letters*, Vol. 85, No. 6, pp. 317–325, 2003.

[Tu 04] Z. G. Tu and L. Yong. "A robust stochastic genetic algorithm (StGA) for global numerical optimization". *IEEE Trans. Evol. Comput.*, Vol. 8, No. 5, pp. 456–470, 2004.

[Van 01] F. Van Den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD thesis, Faculty of Natural and Agricultural Sciences, University of Pretoria, South Africa, 2001.

[Vent 05] G. Venter and J. Sobieszczanski-Sobieski. "A parallel particle swarm optimization algorithm accelerated by asynchronous evaluations". In : *6th World Congresses of Structural and Multidisciplinary Optimization*, Rio de Janeiro, Brazil, 30 May - 03 June 2005.

[Wang 08] Y. X. Wang and Q. L. Xiang. "Particle Swarms with dynamic ring topology". In : *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 419–423, IEEE, 2008.

[Watt 98] D. J. Watts and S. H. Strogatz. "Collective dynamics of 'small-world' networks". *Nature*, Vol. 393, No. 6684, pp. 440–442, June 1998.

[Watt 99] D. J. Watts. *Small worlds : The dynamics of networks between order and randomness*. Princeton University Press, Princeton, NJ, 1999.

[Wilc 45] F. Wilcoxon. "Individual Comparisons by Ranking Methods". *Biometrics Bulletin*, Vol. 1, No. 6, pp. 80–83, 1945.

[Yao 99] X. Yao, Y. Liu, and G. M. Lin. "Evolutionary programming made faster". *IEEE Trans. Evol. Comput.*, Vol. 3, No. 2, pp. 82–102, 1999.

[Yen 08] G. Yen and M. Daneshyari. "Diversity-Based Information Exchange among Multiple Swarms in Particle Swarm Optimization". *International Journal of Computational Intelligence and Applications*, Vol. 7, No. 1, pp. 57–75, 2008.

[Zhan 09] C. Zhang, J. Ning, S. Lu, D. Ouyang, and T. Ding. "A novel hybrid differential evolution and particle swarm optimization algorithm for unconstrained optimization". *Oper. Res. Lett.*, Vol. 37, No. 2, pp. 117–122, March 2009.

[Zhen 03] Y. Zheng, L. Ma, L. Zhang, and J. Qian. "On the convergence analysis and parameter selection in particle swarm optimization". In : *Proceedings of International Conference on Machine Learning and Cybernetics 2003*, pp. 1802–1807, 2003.

[Zhon 04] W. C. Zhong, J. Liu, Z. Xue, and L. C. Jiao. "A multiagent genetic algorithm for global numerical optimization". *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 34, No. 2, pp. 1128–1141, 2004.