Adaptation de la métaheuristique des colonies de fourmis pour l'optimisation difficile en variables continues. Application en génie biologique et médical.

Johann Dréo

Université Paris 12 – Val de Marne
UFR de Sciences

Thèse de doctorat en sciences

Spécialité : Génie Biologique et Médical

Option : Optimisation

Adaptation de la méthode des colonies de fourmis pour l'optimisation en variables continues. Application en génie biomédical

JOHANN DRÉO

Thèse préparée sous la direction de PATRICK SIARRY.

Soutenue le 13 décembre 2004, devant le jury composé de :

- DANIEL JOLLY, Professeur à l'Université d'Artois, 61ème section, *Rapporteur*
- GILLES VENTURINI, Professeur à l'Université de Tours, 27ème section, *Rapporteur*
- FLORENT CHAVAND, Professeur à l'Université d'Evry, 61ème section, *Examinateur*
- GUY THÉRAULAZ, Directeur de Recherche au CNRS, Toulouse, apparenté 27ème section, *Examinateur*
- ION CRISTIAN TRÉLÉA, Maître de Conférences, INA-Grignon, H.D.R. 61ème section, *Examinateur*
- PATRICK SIARRY, Professeur à l'Université de Paris 12, 61ème section, *Directeur de thèse*

Adaptation de la méthode des colonies de fourmis pour l'optimisation en variables continues. Application en génie biomédical

Johann Dréo

Résumé

Bien que les problèmes d'optimisation difficile en variables continues soient très courants en ingénierie, ils sont peu étudiés en recherche opérationnelle. La plupart des algorithmes d'optimisation sont en effet proposés dans le domaine combinatoire. Dès lors, l'adaptation de ces métaheuristiques combinatoires aux problèmes continus est profitable à un grand nombre de problèmes réels.

Parmi ces méthodes, les algorithmes de colonies de fourmis forment une classe des métaheuristiques récemment proposée pour les problèmes d'optimisation difficile. Ces algorithmes s'inspirent des comportements collectifs de dépôt et de suivi de piste observés dans les colonies de fourmis. Une colonie d'agents simples (les fourmis) communiquent indirectement via des modifications dynamiques de leur environnement (les pistes de phéromone) et construisent ainsi une solution à un problème, en s'appuyant sur leur expérience collective.

Deux approches sont possibles pour concevoir des métaheuristiques d'optimisation continue en suivant cette métaphore. La première consiste à créer un système multi-agent où la communication joue un rôle central, en tant que processus permettant l'émergence d'un comportement global cohérent du système. Nous proposons, dans cette optique, des algorithmes de "colonies de fourmis interagissantes" (baptisés *CIAC* et *HCIAC*). La deuxième approche décrit les algorithmes de colonies de fourmis comme des méthodes manipulant un échantillonnage d'une distribution de probabilité. Nous proposons ainsi une métaheuristique "à estimation de distribution" (nommée *CHEDA*).

De plus, la conception de métaheuristiques peut être guidée par le concept de programmation à mémoire adaptative, qui met notamment l'accent sur les processus de mémoire, de diversification et d'intensification. Nos algorithmes font l'objet d'une hybridation avec une recherche locale de Nelder-Mead, qui améliore l'intensification de la recherche.

Les algorithmes de colonies de fourmis du type "multi-agent" présentent des caractéristiques de flexibilité particulièrement intéressantes sur des problèmes combinatoires dynamiques. Nous avons donc adapté notre méthode hybride de colonie de fourmis interagissantes à des problèmes continus dynamiques (algorithme *DHCIAC*), pour lesquels nous proposons également un nouveau jeu de test cohérent.

Nos algorithmes sont enfin appliqués dans le cadre d'un problème biomédical concernant les pathologies du vieillissement oculaire. L'automatisation du suivi des lésions de l'oeil nécessite en effet une étape d'optimisation, lors du recalage d'images d'une séquence d'angiographie rétinienne.

Mots-clefs : Optimisation, optimisation difficile, optimisation continue, optimisation dynamique, métaheuristiques, algorithmes de colonies de fourmis, auto-organisation, programmation à mémoire adaptative, estimation de distribution, recalage d'image, angiographie rétinienne

Remerciements

Je remercie Armelle pour tout (et pour le reste).

Je veux remercier Patrick Siarry, pour m'avoir donné la possibilité de faire cette thèse, ainsi que pour son encadrement sans faille. Il a toute ma gratitude pour m'avoir laissé une grande liberté dans mes recherches et m'avoir consacré son temps sans compter (et ce n'est pas rien !).

Je voudrais exprimer ma gratitude à Daniel Jolly et à Gilles Venturini, pour avoir accepté d'être rapporteurs. Et également à Florent Chavand, Guy Théraulaz et Ion Cristian Tréléa, qui ont bien voulu faire partie de mon jury.

Je suis très reconnaissant au LERISS, notamment envers Jacques Lemoine pour m'avoir ouvert les portes de son laboratoire, ainsi que tous les membres, pour m'avoir d'emblée accueilli chaleureusement. Merci à Patricia Jamin et Annick Joncour, toujours là pour pallier ma malchance administrative.

Je remercie également mes collègues thésards, pour la bonne ambiance au laboratoire et leur amitié, notamment Jean-Claude Nunes, Steve Guyot, Kaouthar Hila, Jean-Michel Glaume, Jean-Christophe Chotis, Marie-Cécile Péron, Walid Tfaili et Lounis Salhi. Merci à Oumar Niang, qui partage ma vision de la recherche. Merci aussi à Béatrice Renou-Dias et Diane Bouvet, pour leur soutien moral de qualité.

Je tiens à remercier tout particulièrement Alain Pétrowski et Éric Taillard pour avoir accepté de co-écrire le livre avec Patrick Siarry et moi-même ; j'ai beaucoup appris grâce à eux. Merci aussi à Amitava Chatterjee pour la traduction du livre et ses idées de recherche. Merci à Gaël Desbois pour ses contributions.

Un grand merci également à tous ceux avec qui j'ai eu des discussions, toutes plus intéressantes les unes que les autres, notamment Yann Colette, Nicolas Monmarché, Alain Berro, Marc Sevaux, Krzysztof Socha et ceux que j'oublie. Un merci tout particulier à Jean-Sébastien Pierre et Jean-Louis Deneubourg, qui m'ont transmis le goût de la recherche.

Je n'oublie pas mes quelques stagiaires, qui m'ont fait progresser, merci Hakim Haroun, José Oliveira, Thibault Tertois, Pierre Truchetet et Francis Djebelo.

Des remerciements spéciaux à mes parents, Fabien Marty, Frédéric Vincent, Charles Basset, Romain Jetur et les autres, pour m'avoir forcé à faire autre chose de temps en temps.

Merci à celles et ceux qui auront lu tout ou partie de ce manuscrit et qui y auront trouvé quelque intérêt. Merci enfin à ceux que j'aurais oublié, ils savent que j'ai une piètre mémoire et ne m'en voudront pas trop, j'espère !

Table des matières

Introduction

Cette thèse part du constat simple que les colonies de fourmis résolvent des problèmes complexes, bien que l'intelligence d'une fourmi soit limitée. L'étude de leur comportement a donné naissance à plusieurs nouvelles méthodes de résolution de problèmes.

L'inspiration pour la création de nouvelles méthodes en ingénierie provient d'horizons très divers, des mathématiques les plus abstraites aux sciences sociales, en passant par la biologie. L'étude de phénomènes réels est d'ailleurs une source fertile d'inspiration en ingénierie informatique, où l'étude et la modélisation des systèmes complexes sont très présentes.

En recherche opérationnelle, et plus précisément dans le domaine de l'optimisation difficile, la majorité des méthodes sont inspirées par de telles études, et notamment par la biologie. Le fait que la biologie étudie souvent des systèmes présentant des comportements dits "intelligents" n'est pas étranger au fait qu'ils soient modélisés, puis transposés dans le cadre de problèmes "réels". On parle parfois d'*intelligence artificielle biomimétique* pour désigner de telles approches.

Parmi les domaines de la biologie fertiles en inspiration, l'éthologie (étude du comportement des animaux) a récemment donné lieu à plusieurs avancées significatives, dont la conception de systèmes de fourmis artificielles. Ces systèmes sont notamment étudiés en robotique, en classification ou encore en optimisation. On rencontre souvent le terme d'"intelligence en essaim" pour désigner ces systèmes, où l'intelligence du système entier est plus grande que celle de la simple somme de ses parties.

Dans le cadre de l'optimisation, cette approche a donné lieu à la création de nouvelles *métaheuristiques*. Les métaheuristiques forment une famille d'algorithmes d'optimisation visant à résoudre des problèmes d'*optimisation difficile*, pour lesquels on ne connaît pas de méthode classique plus efficace. Elles sont généralement utilisées comme des méthodes génériques pouvant optimiser une large gamme de problèmes différents, sans nécessiter de changements profonds dans l'algorithme employé. Les algorithmes de colonies de fourmis forment ainsi une classe de métaheuristique récemment proposée pour les problèmes d'optimisation difficile discrets.

Bien que les problèmes d'optimisation difficile en variables continues soient très courants en ingénierie, ils sont peu étudiés en recherche opérationnelle. La plupart des algorithmes d'optimisation sont, en effet, proposés dans le domaine combinatoire. Dès lors, l'adaptation de ces métaheuristiques combinatoires aux problèmes continus est profitable à un grand nombre de problèmes réels.

Cette thèse a été préparée au sein de l'université de Paris 12, dans le *Laboratoire d'Étude et de Recherche en Instrumentation, Signaux et Systèmes* (LERISS, E.A. 412). Ce laboratoire est principalement orienté vers l'imagerie et le traitement du signal en génie biologique et médical. Ce travail a été financé par une allocation de recherche du ministère de la recherche, puis par un demi poste d'attaché temporaire à l'enseignement et à la recherche (ATER), partagé entre le LERISS et l'IUT de Créteil-Vitry. Cette thèse a été encadrée par P. Siarry, professeur et responsable de l'équipe "optimisation", activité transversale au LERISS. Cette équipe est notamment spécialisée dans l'adaptation des métaheuristiques aux problèmes à variables continues. Les travaux de recherche précédents ont ainsi concerné l'optimisation multiobjectif ou encore l'adaptation au cas continu du recuit simulé, des algorithmes évolutionnaires, de la recherche avec tabous, ... Ont participé à mon travail de doctorat trois stagiaires en dernière année d'ingénieurs (José Oliveira, Thibault Tertois et Pierre Truchetet) et deux en DEA (Hakim Haroun et Francis Djebelo). Dans le cadre de l'équipe optimisation, une thèse est actuellement en cours sur un sujet connexe — concernant l'optimisation par essaims particulaires — (Lounis Salhi) et une autre vise plus particulièrement à poursuivre le présent travail de recherche sur les algorithmes de colonies de fourmis (Walid Tfaili).

Le but de la présente thèse est d'utiliser la métaphore des colonies de fourmis pour concevoir des métaheuristiques d'optimisation adaptées aux problèmes en variables continues comme on en rencontre souvent en ingénierie. Les objectifs que nous nous sommes fixés consistent à comprendre et isoler les mécanismes intéressants de cette métaphore, utiliser différentes approches de conception de métaheuristiques suivant cette optique et appliquer les algorithmes ainsi conçus à un problème d'optimisation du domaine biomédical.

Nous verrons, au cours du premier chapitre, comment les algorithmes de colonies de fourmis s'inspirent des comportements collectifs de dépôt et de suivi de piste observés dans les colonies de fourmis. Une colonie d'agents simples (les fourmis) communiquent indirectement via des modifications dynamiques de leur environnement (les pistes de phéromone) et construisent ainsi une solution à un problème, en s'appuyant sur leur expérience collective.

Deux approches sont possibles pour concevoir des métaheuristiques d'optimisation continue en suivant cette métaphore, comme nous le verrons dans le chapitre 2. La première consiste à utiliser les concepts d'auto-organisation et de programmation à mémoire

adaptative, pour créer un système multi-agent, où la communication joue un rôle central, en tant que processus permettant l'émergence d'un comportement global cohérent du système. La programmation à mémoire adaptative permet, en effet, de cerner les composants principaux d'une métaheuristique itérative, et donne ainsi des guides pour la conception de nouvelles méthodes. Les concepts de mémoire, de diversification et d'intensification sont donc, en un sens, des buts à atteindre pour construire une métaheuristique. Ces buts sont atteints via les concepts apportés par l'auto-organisation, qui décrit alors une voie de conception, où les interactions locales et les rétro-actions sont centrales. Cette approche part donc, en quelque sorte, du "bas" (le comportement des fourmis) vers le "haut" (le comportement de la métaheuristique).

La deuxième approche décrit les algorithmes de colonies de fourmis comme des méthodes manipulant un échantillonnage d'une distribution de probabilité. Ces méthodes se fondent notamment sur des principes communs avec ceux de la programmation à mémoire adaptative. Ainsi, les méthodes à colonies de fourmis peuvent être perçues comme des algorithmes probabilistes, qui manipulent un échantillonnage de distribution, en augmentant la probabilité d'explorer les meilleures solutions à chaque itération. Cette approche, à l'inverse de la précédente, part donc plutôt du "haut" (le comportement souhaité de l'algorithme) vers le "bas" (les processus de base).

Nous proposons, en suivant la première approche, des algorithmes de "colonies de fourmis interagissantes" (baptisés *CIAC*). Présentées dans le chapitre 3, ces méthodes mettent l'accent sur les concepts de canaux de communication et de comportement individuel des fourmis. Nos algorithmes sont principalement expérimentés sur des problèmes d'optimisation *statiques*, mais nous proposons une variante pour l'optimisation *dynamique* en variables continues (*DHCIAC*). Nous avons, à cette occasion, élaboré un nouveau jeu de fonctions de test, qui vise à parcourir une large gamme de caractéristiques de l'optimisation dynamique, dans un but de cohérence.

La seconde approche nous a permis de concevoir un algorithme à estimation de distribution, baptisé *CHEDA*, présenté dans le chapitre 4. Cette méthode utilise une distribution normale comme moyen de reproduction du comportement probabiliste des algorithmes de colonies de fourmis.

La programmation à mémoire adaptative met en avant les processus d'intensification, qui sont relativement bien illustrés par des algorithmes de recherche locale. Le grand nombre d'algorithmes d'optimisation déjà existants tendant à faire se recouper des méthodes venues d'horizons différents, il est ainsi souvent judicieux de s'appuyer sur les méthodes éprouvées pour améliorer une nouvelle métaheuristique. Aussi, nos algorithmes font-ils l'objet d'une hybridation avec une recherche locale. Cette hybridation a

été effectuée de façon à lier les deux algorithmes mis en jeu, autrement que par une simple intensification terminale (chapitres 3 et 4).

Le chapitre 5 présente comment, dans le cadre de l'application de nos algorithmes au domaine biomédical, nous apportons une nouvelle approche dans le processus de recalage d'images d'angiographies rétiniennes. L'automatisation du suivi des lésions de l'oeil nécessite en effet une étape d'optimisation ; elle vise à minimiser l'écart entre deux images successives, à l'aide d'une fonction de vraisemblance.

La conclusion récapitule enfin nos contributions et propose des perspectives sur les travaux effectués.

Chapitre 1

Algorithmes de colonies de fourmis en optimisation : état de l'art

1.1 Introduction

L'optimisation est un sujet central en recherche opérationnelle, un grand nombre de problèmes d'aide à la décision pouvant en effet être décrits sous la forme de problèmes d'optimisation. Les problèmes d'identification, l'apprentissage supervisé de réseaux de neurones ou encore la recherche du plus court chemin sont, par exemple, des problèmes d'optimisation.

Ce chapitre décrit tout d'abord le cadre de l'*optimisation difficile* et des *métaheuristiques* dans lequel nous nous plaçons dans ce travail, puis présente un état de l'art sur les métaheuristiques dites "de colonies de fourmis".

Les algorithmes de colonies de fourmis forment une classe des métaheuristiques récemment proposée pour les problèmes d'optimisation difficile. Ces algorithmes s'inspirent des comportements collectifs de dépôt et de suivi de piste observés dans les colonies de fourmis. Une colonie d'agents simples (les fourmis) communiquent indirectement via des modifications dynamiques de leur environnement (les pistes de phéromone) et construisent ainsi une solution à un problème, en s'appuyant sur leur expérience collective.

1.2 Métaheuristiques pour l'"optimisation difficile"

1.2.1 Optimisation difficile

1.2.1.1 Problème d'optimisation

Un problème d'optimisation au sens général est défini par un ensemble de solutions possibles S , dont la qualité peut être décrite par une fonction objectif f. On cherche

alors à trouver la solution s^* possédant la meilleure qualité $f(s^*)$ (par la suite, on peut chercher à minimiser ou à maximiser $f(s)$). Un problème d'optimisation peut présenter des contraintes d'égalité (ou d'inégalité) sur s , être dynamique si $f(s)$ change avec le temps ou encore multi-objectif si plusieurs fonctions objectifs doivent être optimisées.

Il existe des méthodes déterministes (dites "exactes") permettant de résoudre certains problèmes en un temps fini. Ces méthodes nécessitent généralement un certain nombre de caractéristiques de la fonction objectif, comme la stricte convexité, la continuité ou encore la dérivabilité. On peut citer comme exemple de méthode la programmation linéaire, quadratique ou dynamique, la méthode du gradient, la méthode de Newton, etc.

1.2.1.2 "Optimisation difficile"

Certains problèmes d'optimisation demeurent cependant hors de portée des méthodes exactes. Un certain nombre de caractéristiques peuvent en effet être problématiques, comme l'absence de convexité stricte (multimodalité), l'existence de discontinuités, une fonction non dérivable, présence de bruit, etc.

Dans de tels cas, le problème d'optimisation est dit "difficile" , car aucune méthode exacte n'est capable de le résoudre exactement en un temps "raisonnable", on devra alors faire appel à des heuristiques permettant une optimisation *approchée*.

L'optimisation difficile peut se découper en deux types de problèmes : les problèmes *discrets* et les problèmes *continus*. Le premier cas rassemble les problèmes de type NP-complets, tels que le problème du voyageur de commerce. Un problème "NP" est dit complet s'il est possible de le décrire à l'aide d'un algorithme polynomial sous la forme d'un sous-ensemble d'instances. Concrètement, il est "facile" de décrire une solution à un tel problème, mais le nombre de solutions nécessaires à la résolution croît de manière exponentielle avec la taille de l'instance. Jusqu'à présent, la conjecture postulant que les problèmes NP-complets ne sont pas solubles en un temps polynomial n'a été ni démontrée, ni révoquée. Aucun algorithme polynomial de résolution n'a cependant été trouvé pour de tels problèmes. L'utilisation d'algorithmes d'optimisation permettant de trouver une solution approchée en un temps raisonnable est donc courante.

Dans la seconde catégorie, les variables du problème d'optimisation sont continues. C'est le cas par exemple des problèmes d'identifications, où l'on cherche à minimiser l'erreur entre le modèle d'un système et des observations expérimentales. Ce type de problème est moins formalisé que le précédent, mais un certains 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 pratique, certains problèmes sont mixtes et présente à la fois des variables discrètes et des variables continues.

1.2.2 Algorithmes d'optimisation approchée

1.2.2.1 Heuristiques

Une heuristique d'optimisation est une méthode approchée se voulant simple, rapide et adaptée à un problème donné. Sa capacité à optimiser un problème avec un minimum d'informations est contrebalancée par le fait qu'elle n'offre aucune garantie quant à l'optimalité de la meilleure solution trouvée.

Du point de vue de la recherche opérationnelle, ce défaut n'est pas toujours un problème, tout spécialement quand seule une approximation de la solution optimale est recherchée.

1.2.2.2 Métaheuristiques

Parmi les heuristiques, certaines sont adaptables à un grand nombre de problèmes différents sans changements majeurs dans l'algorithme, on parle alors de *méta*-heuristiques. La plupart des heuristiques et des métaheuristiques utilisent des processus aléatoires comme moyens de récolter de l'information et de faire face à des problèmes comme l'explosion combinatoire. En plus de cette base stochastique, les métaheuristiques sont généralement itératives, c'est-à-dire qu'un même schéma de recherche est appliqué plusieurs fois au cours de l'optimisation, et directes, c'est-à-dire qu'elles n'utilisent pas l'information du gradient de la fonction objectif. Elles tirent en particulier leur intérêt de leur capacité à éviter les optima locaux, soit en acceptant une dégradation de la fonction objectif au cours de leur progression, soit en utilisant une population de points comme méthode de recherche (se démarquant ainsi des heuristiques de *descente locale*).

Souvent inspirées d'analogies avec la réalité (physique, biologie, éthologie, ...), elles sont généralement conçues au départ pour des problèmes discrets, mais peuvent faire l'objet d'adaptations pour des problèmes continus.

Les métaheuristiques, du fait de leur capacité à être utilisées sur un grand nombre de problèmes différents, se prêtent facilement à des extensions. Pour illustrer cette caractéristique, citons notamment :

— l'optimisation multiobjectif (dites aussi multicritère) [Collette and Siarry, 2002], où il faut optimiser plusieurs objectifs contradictoires. La recherche vise alors non pas à trouver un optimum global, mais un ensemble d'optima "au sens de Pareto" formant la "surface de compromis" du problème.

— l'optimisation multimodale, où l'on cherche un ensemble des meilleurs optima globaux et/ou locaux.

— l'optimisation de problèmes bruités, où il existe une incertitude sur le calcul de la fonction objectif. Incertitude dont il faut alors tenir comtpes dans la recherche de l'optimum.
— l'optimisation dynamique, où la fonction objectif varie dans le temps. Il faut alors approcher au mieux l'optimum à *chaque* pas de temps.
— la parallélisation, où l'on cherche à accélérer la vitesse de l'optimisation en répartissant la charge de calcul sur des unités fonctionnant de concert. Le problème revient alors à adapter les métaheuristiques pour qu'elle soient distribuées.
— l'hybridation, qui vise à tirer parti des avantages respectifs de métaheuristiques différentes en les combinants [Talbi, 2002].

Enfin, la grande vitalité de ce domaine de recherche ne doit pas faire oublier qu'un des intérêts majeurs des métaheuristiques est leur facilité d'utilisation dans des problèmes concrets. L'utilisateur est généralement demandeur de méthodes *efficaces* permettant d'atteindre un optimum avec une *précision* acceptable dans un *temps* raisonnable. Un des enjeux de la conception des métaheuristiques est donc de faciliter le *choix* d'une méthode et de simplifier son *réglage* pour l'adapter à un problème donné.

Du fait du foisonnement de la recherche, un grand nombre de classes de métaheuristiques existent, certaines seront présentées plus en détail dans le chapitre 2, comme le recuit simulé, les algorithmes évolutionnaires ou la recherche avec tabous. Le présent chapitre faisant plus particulièrement l'objet des métaheuristiques dites "de colonies de fourmis".

1.3 Algorithmes de colonies de fourmis en optimisation

Le premier algorithme de ce type (le "Ant System"[1]) a été conçu pour le problème du voyageur de commerce, mais n'a pas permis de produire des résultats compétitifs. Cependant, l'intérêt pour la métaphore était lancé et de nombreux algorithmes s'en inspirant ont depuis été proposés — dans divers domaines —, certains atteignant des résultats très convaincants.

1.3.1 Optimisation naturelle : pistes de phéromone

Les algorithmes de colonies de fourmis sont nés à la suite d'une constatation : les insectes sociaux en général, et les fourmis en particulier, résolvent naturellement des problèmes relativement complexes. Les biologistes ont étudié comment les fourmis ar-

[1] traduisible simplement par "Système de Fourmis"

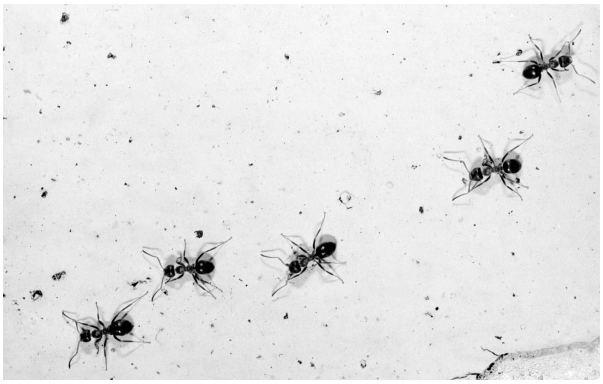


FIG. 1.1 – *Des fourmis suivant une piste de phéromone (photographie de E. D. Taillard, tirée de [Dréo et al., 2003]).*

rivent à résoudre collectivement des problèmes trop complexes pour un seul individu, notamment les problèmes de choix lors de l'exploitation de sources de nourriture.

Les fourmis ont la particularité d'employer pour communiquer des substances volatiles appelées *phéromones*. Elles sont attirées par ces substances, qu'elles perçoivent grâce à des récepteurs situés dans leurs antennes. Ces substances sont nombreuses et varient selon les espèces. Les fourmis peuvent déposer des phéromones au sol, grâce à une glande située dans leur abdomen, et former ainsi des pistes odorantes, qui pourront être suivies par leurs congénères (figure 1.1).

Les fourmis utilisent les pistes de phéromone pour marquer leur trajet, par exemple entre le nid et une source de nourriture. Une colonie est ainsi capable de choisir (sous certaines conditions) le plus court chemin vers une source à exploiter [Goss et al., 1989, Beckers et al., 1992], sans que les individus aient une vision *globale* du trajet.

En effet, comme l'illustre la figure 1.2, les fourmis le plus rapidement arrivées au nid, après avoir visité la source de nourriture, sont celles qui empruntent les deux branches les plus courtes. Ainsi, la *quantité* de phéromone présente sur le plus court trajet est légèrement plus importante que celle présente sur le chemin le plus long. Or, une piste présentant une plus grande concentration en phéromone est plus attirante pour les fourmis, elle a une *probabilité* plus grande d'être empruntée. La piste courte va alors être plus renforcée que la longue, et, à terme, sera choisie par la grande majorité des fourmis.

On constate qu'ici le choix s'opère par un mécanisme d'*amplification* d'une fluctuation initiale. Cependant, il est possible qu'en cas d'une plus grande quantité de phéromone déposée sur les grandes branches, au début de l'expérience, la colonie choisisse le plus long parcours.

D'autres expériences [Beckers et al., 1992], avec une autre espèce de fourmis, ont montré que si les fourmis sont capables d'effectuer des demi-tours sur la base d'un trop grand

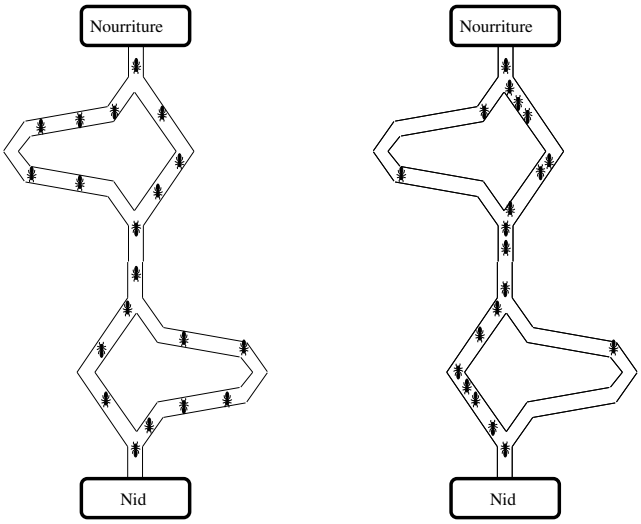


FIG. 1.2 – *Expérience de sélection des branches les plus courtes par une colonie de fourmis : (a) au début de l'expérience, (b) à la fin de l'expérience.*

écart par rapport à la direction de la source de nourriture, alors la colonie est plus flexible et le risque d'être piégé sur le chemin long est plus faible.

Il est difficile de connaître avec précision les propriétés physico-chimiques des pistes de phéromone, qui varient en fonction des espèces et d'un grand nombre de paramètres. Cependant, les métaheuristiques d'optimisation de colonies de fourmis s'appuient en grande partie sur le phénomène d'*évaporation* des pistes de phéromone. Or, on constate dans la nature que les pistes s'évaporent plus lentement que ne le prévoient les modèles. Les fourmis réelles disposent en effet "d'heuristiques" leur apportant un peu plus d'informations sur le problème (par exemple une information sur la direction). Il faut garder à l'esprit que l'intérêt immédiat de la colonie (trouver le plus court chemin vers une source de nourriture) peut être en concurrence avec l'intérêt *adaptatif* de tels comportements. Si l'on prend en compte l'ensemble des contraintes que subit une colonie de fourmis (prédation, compétition avec d'autres colonies, etc.), un choix rapide et stable peut être meilleur, et un changement de site exploité peut entraîner des coûts trop forts pour permettre la sélection naturelle d'une telle option.

1.3.2 Optimisation par colonies de fourmis et problème du voyageur de commerce

Le problème du voyageur de commerce ("Travelling Salesman Problem", *TSP*) a fait l'objet de la première implémentation d'un algorithme de colonies de fourmis : le "Ant System" (*AS*) [Colorni et al., 1992]. Le passage de la métaphore à l'algorithme est ici relativement facile et le problème du voyageur de commerce est bien connu et étudié. Il est intéressant d'approfondir le principe de ce premier algorithme pour bien comprendre le mode de fonctionnement des algorithmes de colonies de fourmis. Il y a deux façons d'aborder ces algorithmes. La première, la plus évidente au premier abord, est celle qui a historiquement mené au "Ant System" original ; nous avons choisi de la décrire dans cette section. La seconde est une description plus formelle des mécanismes communs aux algorithmes de colonies de fourmis, elle sera décrite dans la section 1.3.4.

Le problème du voyageur de commerce consiste à trouver le trajet le plus court (désigné par "tournée" ou plus loin par "tour") reliant n villes données, chaque ville ne devant être visitée qu'une seule fois. Le problème est plus généralement défini comme un graphe complètement connecté (N, A), où les villes sont les noeuds N et les trajets entre ces villes, les arêtes A.

1.3.2.1 Algorithme de base

Dans l'algorithme *AS*, à chaque itération t ($1 \leq t \leq t_{max}$), chaque fourmi k ($k = 1, \ldots, m$) parcourt le graphe et construit un trajet complet de $n = |N|$ étapes (on note $|N|$ le cardinal de l'ensemble N). Pour chaque fourmi, le trajet entre une ville i et une ville j dépend de :

1. la liste des villes déjà visitées, qui définit les mouvements possibles à chaque pas, quand la fourmi k est sur la ville i : J_i^k ;
2. l'inverse de la distance entre les villes : $\eta_{ij} = \frac{1}{d_{ij}}$, appelée *visibilité*. Cette information "statique" est utilisée pour diriger le choix des fourmis vers des villes proches, et éviter les villes trop lointaines ;
3. la quantité de phéromone déposée sur l'arête reliant les deux villes, appelée l'*intensité de la piste*. Ce paramètre définit l'attractivité d'une partie du trajet global et change à chaque passage d'une fourmi. C'est, en quelque sorte, une mémoire globale du système, qui évolue par apprentissage.

La règle de déplacement (appelée "règle aléatoire de transition proportionnelle" par les auteurs [Bonabeau et al., 1999]) est la suivante :

$$p_{ij}^{k}(t) = \begin{cases} \frac{(\tau_{ij}(t))^{\alpha} \cdot (\eta_{ij})^{\beta}}{\sum_{l \in J_i^k} (\tau_{il}(t))^{\alpha} \cdot (\eta_{ij})^{\beta}} & \text{si } j \in J_i^k \\ 0 & \text{si } j \notin J_i^k \end{cases} \tag{1.1}$$

où α et β sont deux paramètres contrôlant l'importance relative de l'*intensité* de la piste, $\tau_{ij}(t)$, et de la *visibilité*, η_{ij}. Avec $\alpha = 0$, seule la visibilité de la ville est prise en compte ; la ville la plus proche est donc choisie à chaque pas. Au contraire, avec $\beta = 0$, seules les pistes de phéromone jouent. Pour éviter une sélection trop rapide d'un trajet, un compromis entre ces deux paramètres, jouant sur les comportements de *diversification* et d'*intensification* (voir section 1.3.4.3 de ce chapitre), est nécessaire.

Après un tour complet, chaque fourmi laisse une certaine quantité de phéromone $\Delta\tau_{ij}^{k}(t)$ sur l'ensemble de son parcours, quantité qui dépend de la *qualité* de la solution trouvée :

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \frac{Q}{L^k(t)} & \text{si } (i,j) \in T^k(t) \\ 0 & \text{si } (i,j) \notin T^k(t) \end{cases} \tag{1.2}$$

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

L'algorithme ne serait pas complet sans le processus d'*évaporation* des pistes de phéromone. En effet, pour éviter d'être piégé dans des solutions sous-optimales, il est nécessaire de permettre au système "d'oublier" les mauvaises solutions. On contrebalance donc l'additivité des pistes par une décroissance constante des valeurs des arêtes à chaque itération. La règle de mise à jour des pistes est donc :

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) \tag{1.3}$$

où m est le nombre de fourmis et ρ le taux d'évaporation. La quantité initiale de phéromone sur les arêtes est une distribution uniforme d'une petite quantité $\tau_0 \geq 0$.

La figure 1.3 présente un exemple simplifié de problème du voyageur de commerce optimisé par un algorithme AS dont le pseudo-code est présenté sur l'algorithme 1.1.

1.3.2.2 Variantes

***Ant System* & élitisme** Une première variante du "Système de Fourmis" a été proposée dans [Dorigo et al., 1996] : elle est caractérisée par l'introduction de fourmis "élitistes". Dans cette version, la meilleure fourmi (celle qui a effectué le trajet le plus court) dépose une quantité de phéromone plus grande, dans l'optique d'accroître la probabilité des autres fourmis d'explorer la solution la plus prometteuse.

Algorithme 1.1 Algorithme de colonies de fourmis de base : le "Ant System".

Pour $t = 1, \ldots, t_{max}$

  Pour chaque fourmi $k = 1, \ldots, m$

    Choisir une ville au hasard

    Pour chaque ville non visitée i

      Choisir une ville j, dans la liste J_i^k des villes restantes, selon la formule 1.1

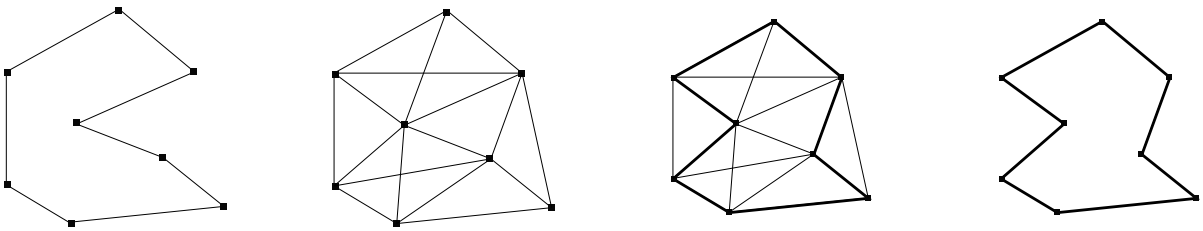
    Fin Pour

    Déposer une piste $\Delta\tau_{ij}^k(t)$ sur le trajet $T^k(t)$ conformément à l'équation 1.2

  Fin Pour

  Évaporer les pistes selon la formule 1.3

Fin Pour



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

FIG. 1.3 – *Le problème du voyageur de commerce optimisé par l'algorithme AS, les points représentent les villes et l'épaisseur des arêtes la quantité de phéromone déposée. (a) exemple de trajet construit par une fourmi, (b) au début du calcul, tous les chemins sont explorés, (c) le chemin le plus court est plus renforcé que les autres, (d) l'évaporation permet d'éliminer les moins bonnes solutions.*

Ant-Q Dans cette variante de AS, la règle de mise à jour locale est inspirée du "Q-learning[2]" [Gambardella and Dorigo, 1995]. Cependant, aucune amélioration par rapport à l'algorithme AS n'a pu être démontrée. Cet algorithme n'est d'ailleurs, de l'aveu même des auteurs, qu'une pré-version du "Ant Colony System".

Ant Colony System L'algorithme "Ant Colony System" (ACS) a été introduit pour améliorer les performances du premier algorithme sur des problèmes de grandes tailles [Dorigo and Gambardella, 1997b, Dorigo and Gambardella, 1997a]. ACS est fondé sur des modifications du AS :

[2]un algorithme d'apprentissage par renforcement

1. *ACS* introduit une règle de transition dépendant d'un paramètre q_0 ($0 \leq q_0 \leq 1$), qui définit une balance *diversification/intensification*. Une fourmi k sur une ville i choisira une ville j par la règle :

$$j = \begin{cases} argmax_{u \in J_i^k} \left[(\tau_{iu}(t)) \cdot (\eta_{iJ})^\beta \right] & \text{si } q \leq q_0 \\ J & \text{si } q > q_0 \end{cases}$$

où q est une variable aléatoire uniformément distribuée sur $[0, 1]$ et $J \in J_i^k$ une ville sélectionnée aléatoirement selon la probabilité :

$$p_{iJ}^k(t) = \frac{(\tau_{iJ}(t)) \cdot (\eta_{iJ})^\beta}{\sum_{l \in J_i^k} (\tau_{il}(t)) \cdot (\eta_{il})^\beta} \tag{1.4}$$

En fonction du paramètre q_0, il y a donc deux comportements possibles : si $q > q_0$ le choix se fait de la même façon que pour l'algorithme *AS*, et le système tend à effectuer une *diversification* ; si $q \leq q_0$, le système tend au contraire vers une *intensification*. En effet, pour $q \leq q_0$, l'algorithme exploite davantage l'information récoltée par le système, il ne peut pas choisir un trajet non exploré.

2. La gestion des pistes est séparée en deux niveaux : une mise à jour locale et une mise à jour globale. Chaque fourmi dépose une piste lors de la mise à jour locale :

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \rho \cdot \tau_0$$

où τ_0 est la valeur initiale de la piste. À chaque passage, les arêtes visitées voient leur quantité de phéromone diminuer, ce qui favorise la diversification par la prise en compte des trajets non explorés. À chaque itération, la mise à jour globale s'effectue comme ceci :

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

où les arêtes (i, j) appartiennent au meilleur tour T^+de longueur L^+ et où $\Delta\tau_{ij}(t) = \frac{1}{L^+}$. Ici, seule la meilleure piste est donc mise à jour, ce qui participe à une intensification par sélection de la meilleure solution.

3. Le système utilise une liste de candidats. Cette liste stocke pour chaque ville les v plus proches voisines, classées par distances croissantes. Une fourmi ne prendra en compte une arête vers une ville en dehors de la liste que si celle-ci a déjà été explorée. Concrètement, si toutes les arêtes ont déjà été visitées dans la liste de candidats, le choix se fera en fonction de la règle 1.4, sinon c'est la plus proche des villes non visitées qui sera choisie.

ACS & 3-opt Cette variante est une hybridation entre le *ACS* et une recherche locale de type 3-opt [Dorigo and Gambardella, 1997b]. Ici, la recherche locale est lancée pour améliorer les solutions trouvées par les fourmis (et donc les ramener à l'optimum local le plus proche).

Max-Min Ant System Cette variante (notée *MMAS*) est fondée sur l'algorithme *AS* et présente quelques différences notables [Stützle and Hoos, 1997, Stützle and Hoos, 2000] :

1. Seule la meilleure fourmi met à jour une piste de phéromone.
2. Les valeurs des pistes sont bornées par τ_{min} et τ_{max}.
3. Les pistes sont initialisées à la valeur maximum τ_{max}.
4. La mise à jour des pistes se fait de façon proportionnelle, les pistes les plus fortes étant moins renforcées que les plus faibles.
5. Une ré-initialisation des pistes peut être effectuée.

Les meilleurs résultats sont obtenus en mettant à jour la meilleure solution avec une *fréquence* de plus en plus forte au cours de l'exécution de l'algorithme.

1.3.2.3 Choix des paramètres

Pour l'algorithme *AS*, les auteurs préconisent que, bien que la valeur de Q ait peu d'influence sur le résultat final, cette valeur soit du même ordre de grandeur qu'une estimation de la longueur du meilleur trajet trouvé. D'autre part, la ville de départ de chaque fourmi est typiquement choisie par un tirage aléatoire uniforme, aucune influence significative du placement de départ n'ayant pu être démontrée.

En ce qui concerne l'algorithme *ACS*, les auteurs conseillent d'utiliser $\tau_0 = (n \cdot L_{nn})^{-1}$, où n est le nombre de villes et L_{nn} la longueur d'un tour trouvé par la méthode du plus proche voisin.

Le nombre de fourmis m est un paramètre important ; les auteurs suggèrent d'utiliser autant de fourmis que de villes (i.e. $m = n$) pour de bonnes performances sur le problème du voyageur de commerce. Il est possible de n'utiliser qu'une seule fourmi, mais l'effet d'amplification des longueurs différentes est alors perdu, de même que le parallélisme naturel de l'algorithme, ce qui peut s'avérer néfaste pour certains problèmes. En règle générale, les algorithmes de colonies de fourmis semblent assez peu sensibles à un réglage fin du nombre de fourmis.

1.3.3 Autres problèmes combinatoires

Les algorithmes de colonies de fourmis sont très étudiés depuis quelques années et il serait trop long de faire ici une liste exhaustive de toutes les applications et variantes

qui ont été produites, même en se restreignant au domaine de l'optimisation. Dans les deux principaux champs d'application (problèmes NP-difficiles et problèmes dynamiques), certains algorithmes ont cependant donné de très bons résultats. On peut notamment retenir des performances particulièrement intéressantes dans le cas de l'affectation quadratique [Stützle and Hoos, 2000], de problèmes de planification [Merkle et al., 2000], de l'ordonnancement séquentiel [Gambardella and Dorigo, 2000], du routage de véhicule [Gambardella et al., 1999], ou du routage sur réseau [Di Caro and Dorigo, 1998b] (voir aussi pour cette application la section 1.3.5.2 de ce chapitre). Il existe une littérature importante sur toutes sortes de problèmes : voyageur de commerce, coloriage de graphes, affectation de fréquence, affectation généralisée, sac à dos multi-dimensionnel, satisfaction de contraintes, etc.

1.3.4 Formalisation et propriétés d'un algorithme de colonie de fourmis

Une description élégante a été proposée [Dorigo and Stützle, 2003], qui s'applique aux problèmes (combinatoires) où une construction partielle de la solution est possible. Ce cas, bien que restrictif, permet de dégager les apports originaux de ces métaheuristiques (dénommées *ACO,* pour "Ant Colony Optimization", par les auteurs). Nous en avons traduit ci-dessous un extrait :

> *Une métaheuristique de colonie de fourmis est un processus stochastique construisant une solution, en ajoutant des composants aux solutions partielles. Ce processus prend en compte (i) une heuristique sur l'instance du problème (ii) des pistes de phéromone changeant dynamiquement pour refléter l'expérience acquise par les agents.*

Une formalisation plus précise existe [Dorigo and Stützle, 2003]. Elle passe par une *représentation* du problème, un *comportement* de base des fourmis et une *organisation* générale de la métaheuristique. Plusieurs concepts sont également à mettre en valeur pour comprendre les principes de ces algorithmes, notamment la définition des pistes de phéromone en tant que *mémoire adaptative*, la nécessité d'un réglage *intensification/diversification* et enfin l'utilisation d'une *recherche locale*. Nous traitons ci-après ces différents sujets.

1.3.4.1 Formalisation

Représentation du problème Le problème est représenté par un *jeu de solutions*, une *fonction objectif* assignant une valeur à chaque solution et un *jeu de contraintes*. L'objectif est de trouver l'optimum global de la fonction objectif satisfaisant les contraintes. Les différents états du problème sont caractérisés comme une séquence de composants. On

(a) (b)

FIG. 1.4 – *Dans un algorithme de colonie de fourmis, les pistes de phéromone peuvent être associées aux composants (a) ou aux connexions (b) du graphe représentant le problème à résoudre.*

peut noter que, dans certains cas, un coût peut être associé à des états autres que des solutions.

Dans cette représentation, les fourmis construisent des solutions en se déplaçant sur un graphe $G = (C, L)$, où les noeuds sont les composants de C et où l'ensemble L connecte les composants de C. Les contraintes du problème sont implémentées directement dans les règles de déplacement des fourmis (soit en empêchant les mouvements qui violent les contraintes, soit en pénalisant de telles solutions).

Comportement des fourmis Les fourmis artificielles peuvent être caractérisées comme une procédure de construction stochastique *construisant* des solutions sur le graphe $G = (C, L)$. En général, les fourmis tentent d'élaborer des solutions faisables mais, si nécessaire, elles peuvent produire des solutions infaisables. Les composants et les connexions peuvent être associés à des pistes de phéromone τ (mettant en place une mémoire adaptative décrivant l'état du système) et à une valeur heuristique η (représentant une information a priori sur le problème, ou venant d'une source autre que celle des fourmis ; c'est bien souvent le coût de l'état en cours). Les pistes de phéromone et la valeur de l'heuristique peuvent être associées soit aux composants, soit aux connexions (figure 1.4).

Chaque fourmi dispose d'une mémoire utilisée pour stocker le trajet effectué, d'un état initial et de conditions d'arrêt. Les fourmis se déplacent d'après une règle de décision probabiliste fonction des *pistes* de phéromone locales, de l'*état* de la fourmi et des *contraintes* du problème. Lors de l'ajout d'un composant à la solution en cours, les fourmis peuvent mettre à jour la piste associée au composant ou à la connexion correspondante. Une fois la solution construite, elles peuvent mettre à jour la piste de phéromone des composants ou des connexions utilisées. Enfin, une fourmi dispose au minimum de la capacité de construire une solution du problème.

Organisation de la métaheuristique En plus des règles régissant le comportement des fourmis, un autre processus majeur a cours : l'*évaporation* des pistes de phéromone. En effet, à chaque itération, la valeur des pistes de phéromone est *diminuée*. Le but de cette diminution est d'éviter une convergence trop rapide et le piégeage de l'algorithme dans des minimums locaux, par une forme d'oubli favorisant l'exploration de nouvelles régions.

Selon les auteurs du formalisme *ACO*, il est possible d'implémenter d'autres processus nécessitant un contrôle *centralisé* (et donc ne pouvant être directement pris en charge par des fourmis), sous la forme de processus annexes. Ce n'est, à notre sens, que peu souhaitable ; en effet, on perd alors la caractéristique décentralisée du système. De plus, l'implémentation de processus *annexes* entre difficilement dans une formalisation rigoureuse.

1.3.4.2 Phéromones et mémoire

L'utilisation de la *stigmergie* est cruciale pour les algorithmes de colonies de fourmis. Le choix de la méthode d'implémentation des pistes de phéromone est donc important pour obtenir les meilleurs résultats. Ce choix est en grande partie lié aux possibilités de *représentation* de l'espace de recherche, chaque représentation pouvant apporter une façon différente d'implémenter les pistes. Par exemple, pour le problème du voyageur de commerce, une implémentation efficace consiste à utiliser une piste τ_{ij} entre deux villes i et j comme une représentation de l'*intérêt* de visiter la ville j après la ville i. Une autre représentation possible, moins efficace en pratique, consiste à considérer τ_{ij} comme une représentation de l'intérêt de visiter i en tant que $j$ème ville.

En effet, les pistes de phéromone décrivent à chaque pas l'état de la recherche de la solution par le système, les agents modifient la façon dont le problème va être *représenté* et perçu par les autres agents. Cette information est partagée par le biais des modifications de l'*environnement* des fourmis, grâce à une forme de communication indirecte : la stigmergie. L'information est donc stockée un certain temps dans le système, ce qui a amené certains auteurs à considérer ce processus comme une forme de *mémoire adaptative* [Taillard, 1998, Taillard et al., 1998], où la dynamique de stockage et de partage de l'information va être cruciale pour le système.

1.3.4.3 Intensification/diversification

Le problème de l'emploi relatif de processus de *diversification* et d'*intensification* est un problème extrêmement courant dans la conception et l'utilisation de métaheuristiques. Par intensification, on entend l'*exploitation* de l'information accumulée par le système à un moment donné. La diversification est au contraire l'*exploration* de régions de l'espace

de recherche imparfaitement prises en compte. Bien souvent, il va s'agir de choisir où et quand "injecter de l'aléatoire" dans le système (*diversification*) et/ou améliorer une solution (*intensification*).

Dans les algorithmes de type *ACO*, comme dans la plupart des cas, il existe plusieurs façons de gérer l'emploi de ces deux facettes des métaheuristiques d'optimisation. La plus évidente passe par le réglage via les deux paramètres α et β, qui déterminent l'influence relative des pistes de phéromone et de l'information heuristique. Plus la valeur de α sera élevée, plus l'*intensification* sera importante, car plus les pistes auront une influence sur le choix des fourmis. À l'inverse, plus α sera faible, plus la *diversification* sera forte, car les fourmis éviteront les pistes. Le paramètre β agit de façon similaire. On doit donc gérer conjointement les deux paramètres, pour régler ces aspects.

On peut également introduire des modifications de la gestion des pistes de phéromone. Par exemple, l'emploi de stratégies *élitistes* (les meilleures solutions contribuent plus aux pistes, voir section 1.3.2.2 : l'algorithme *AS* avec élitisme) favorise l'intensification, alors qu'une ré-initialisation de l'ensemble des pistes favorisera l'exploration (section 1.3.2.2, algorithme *MMAS*).

Ce choix diversification/intensification peut s'effectuer de manière statique avant le lancement de l'algorithme, en utilisant une connaissance a priori du problème, ou de manière dynamique, en laissant le système décider du meilleur réglage. Deux approches sont possibles : un réglage par les paramètres ou l'introduction de nouveaux processus. Dans ces algorithmes fondés en grande partie sur l'utilisation de l'auto-organisation, ces deux approches peuvent être équivalentes, un changement de paramètre pouvant induire un comportement complètement différent du système, au niveau global.

1.3.4.4 Recherche locale et heuristiques

Les métaheuristiques de colonies de fourmis sont souvent plus efficaces quand elles sont *hybridées* avec des algorithmes de recherche locale. Ceux-ci optimisent les solutions trouvées par les fourmis, avant que celles-ci ne soient utilisées pour la mise à jour des pistes de phéromone. Du point de vue de la recherche locale, utiliser des algorithmes de colonies de fourmis pour engendrer une solution initiale est un avantage indéniable. Ce qui différencie une métaheuristique de type *ACO intéressante* d'un algorithme réellement *efficace* est bien souvent l'hybridation avec une recherche locale.

Une autre possibilité pour améliorer les performances est d'injecter une information heuristique plus pertinente. Cet ajout a généralement un coût élevé en terme de calculs supplémentaires.

Il faut noter que ces deux approches sont similaires de par l'emploi qu'elles font des informations de coût pour améliorer une solution. La recherche locale le fait de façon plus

directe que l'heuristique, cependant que cette dernière est peut-être plus naturelle pour utiliser des informations a priori sur le problème.

1.3.4.5 Parallélisme

La structure même des métaheuristiques de colonies de fourmis comporte un parallélisme *intrinsèque*. D'une manière générale, les solutions de bonne qualité émergent du résultat des *interactions* indirectes ayant cours dans le système, pas d'un codage explicite d'échanges. En effet, chaque fourmi ne prend en compte que des informations locales de son environnement (les pistes de phéromone) ; il est donc facile de paralléliser un tel algorithme. Il est intéressant de noter que les différents processus en cours dans la métaheuristique (i.e. le comportement des fourmis, l'évaporation et les processus annexes) peuvent également être implémentés de manière indépendante, l'utilisateur étant libre de décider de la manière dont ils vont interagir.

1.3.4.6 Convergence

Les métaheuristiques peuvent être vues comme des modifications d'un algorithme de base : une recherche aléatoire. Cet algorithme possède l'intéressante propriété de *garantir* que la solution optimale sera trouvée tôt ou tard, on parle alors de convergence. Cependant, puisque cet algorithme de base est *biaisé*, la garantie de convergence n'existe plus.

Si, dans certains cas, on peut facilement être certain de la convergence d'un algorithme de colonies de fourmis (*MMAS* par exemple, voir section 1.3.2.2), le problème reste entier en ce qui concerne la convergence d'un algorithme *ACO* quelconque. Cependant, il existe une variante dont la convergence a été prouvée [Gutjahr, 2000, Gutjahr, 2002] : le "Graph-Based Ant System" (*GBAS*). La différence entre *GBAS* et l'algorithme *AS* se situe au niveau de la mise à jour des pistes de phéromone, qui n'est permise que si une meilleure solution est trouvée. Pour certaines valeurs de paramètres, et étant donné $\epsilon > 0$ une "faible" valeur, l'algorithme trouvera la solution optimale avec une probabilité $P_t \geq 1 - \epsilon$, après un temps $t \geq t_0$ (où t_0 est fonction de ϵ).

1.3.5 Extensions

Devant le succès rencontré par les algorithmes de colonies de fourmis, de nombreuses pistes autres que celle de l'optimisation combinatoire commencent à être explorées : par exemple, l'utilisation de ces algorithmes dans des problèmes *continus* et/ou *dynamiques*, ou encore la mise en relation de ce type d'algorithmes dans un cadre d'*intelligence en essaim* et avec d'autres métaheuristiques.

1.3.5.1 Optimisation continue

Problèmes d'adaptation Les métaheuristiques sont bien souvent élaborées pour des problèmes combinatoires, mais il existe une classe de problèmes souvent rencontrée en ingénierie, où la fonction objectif est à variables *continues* et pour lesquels les métaheuristiques peuvent être d'un grand secours (fonction non dérivable, multiples minimums locaux, grand nombre de variables, non-convexité, etc.). Plusieurs tentatives pour adapter les métaheuristiques de colonies de fourmis au domaine continu sont apparues.

Outre les problèmes classiques d'adaptation d'une métaheuristique, les algorithmes de colonies de fourmis posent un certain nombre de problèmes spécifiques. Ainsi, le principal problème vient si l'on se place dans le formalisme *ACO* avec une construction de la solution composant par composant. En effet, un problème continu peut — selon la perspective choisie — présenter une infinité de composants, le problème de la construction est difficilement soluble dans ce cas. La plupart des algorithmes s'inspirent donc des caractéristiques d'auto-organisation et de mémoire externe des colonies de fourmis, laissant de côté la construction itérative de la solution ; cependant, depuis peu, le caractère probabiliste du formalisme *ACO* commence à être employé.

En dehors de nos propres travaux, qui seront exposés en détail dans les chapitres suivants, nous avons recensé dans la littérature plusieurs algorithmes de colonies de fourmis pour l'optimisation continue : *CACO*, un algorithme hybride non baptisé, *API*, *ACO* pour l'optimisation continue et *CACS*.

L'algorithme *CACO* Le premier de ces algorithmes, tout naturellement nommé *CACO* ("Continuous Ant Colony Algorithm") [Bilchev and Parmee, 1995][Wodrich and Bilchev, 1997] [Mathur et al., 2000], utilise deux approches : un algorithme de type *évolutionnaire* sélectionne et croise des régions d'intérêt, que des *fourmis* explorent et évaluent. Une fourmi sélectionne une région avec une probabilité proportionnelle à la concentration en phéromone de cette région, de la même manière que — dans le "Ant System" —, une fourmi sélectionnerait une piste allant d'une ville à une autre :

$$p_i(t) = \frac{\tau_i^{\alpha}(t) \cdot \eta_i^{\beta}(t)}{\sum_{j=1}^{N} \tau_j^{\alpha}(t) \cdot \eta_j^{\beta}(t)}$$

où N est le nombre de régions et $\eta_i^{\beta}(t)$ est utilisé pour inclure une heuristique spécifique au problème. Les fourmis partent alors du centre de la région et se déplacent selon une direction choisie aléatoirement, tant qu'une amélioration de la fonction objectif est trouvée. Le pas de déplacement utilisé par la fourmi entre chaque évaluation est donné par :

$$\delta r(t, R) = R \cdot \left(1 - u^{(1-\frac{t}{T})^c}\right)$$

où R est le diamètre de la région explorée, $u \in [0, 1]$ un nombre aléatoire, T le nombre total d'itérations de l'algorithme et c un paramètre de refroidissement (permettant de réduire le pas à chaque itération). Si la fourmi a trouvé une meilleure solution, la région est déplacée de façon à ce que son centre coïncide avec cette solution, et la fourmi augmente la quantité de phéromone de la région proportionnellement à l'amélioration trouvée (appelée ici "fitness"). L'évaporation des "pistes" se fait classiquement en fonction d'un coefficient ρ.

Des modifications ont été apportées par Wodrich et al. [Wodrich and Bilchev, 1997] pour améliorer les performances de l'algorithme original. Ainsi, en plus des fourmis "locales" de *CACO*, des fourmis "globales" vont explorer l'espace de recherche (figure 1.5) pour éventuellement remplacer les régions peu intéressantes par de nouvelles régions non explorées. Les régions sont également affectées d'un âge, qui augmente si aucune amélioration n'est découverte. De plus, le paramètre t dans le pas de recherche des fourmis $\delta r(t, R)$ est défini par l'âge de la région explorée.

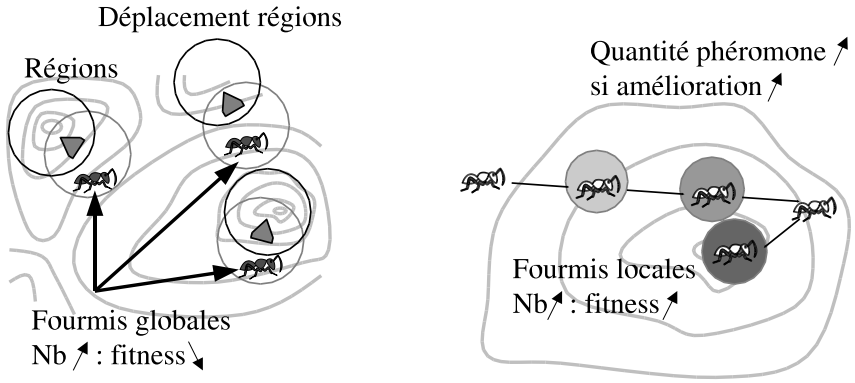


FIG. 1.5 – *L'algorithme* CACO *: les fourmis globales (a) participent au déplacement des régions que les fourmis locales (b) évaluent.*

Une refonte de l'algorithme [Mathur et al., 2000] a été opérée par d'autres auteurs en vue de relier plus finement *CACO* avec le paradigme des colonies de fourmis et d'abandonner la liaison avec l'algorithme évolutionnaire. On parle ainsi par exemple de *diffusion* pour définir la création de nouvelles régions.

Une méthode hybride Une approche semblable — utilisant à la fois une approche de colonies de fourmis et d'algorithme évolutionnaire — a été proposée par Ling et al. [Ling et al., 2002], mais peu de résultats sont disponibles au moment où nous écrivons cette thèse. L'idée principale de cette méthode est de considérer les écarts entre deux individus sur chaque dimension comme autant de parties d'un chemin où les phéromones sont déposées, l'évolution des individus étant prise en charge par des opérateurs de mutation et de croisement. D'un certain point de vue, cette méthode tente donc de reproduire le mécanisme de construction de la solution, composant par composant.

La méthode procède précisément selon l'algorithme 1.2. Chaque fourmi x_i de la population contenant m individus est considérée comme un vecteur à n dimensions. Chaque élément $x_{i,e}$ de ce vecteur peut donc être considéré comme un candidat à l'élément $x^*_{i,e}$ de la solution optimale. L'idée est d'utiliser le chemin entre les éléments $x_{i,e}$ et $x_{j,e}$ — noté (i, j) — pour déposer une piste de phéromone dont la concentration est notée $\tau_{ij}(t)$ au pas de temps t.

Les auteurs ont proposé une version "adaptative", où les probabilités de mutation et de croisement sont variables. Malheureusement, cet algorithme n'a pas encore été complètement testé.

L'algorithme *API* Dans tous les algorithmes évoqués jusqu'ici, le terme "colonies de fourmis" s'entend de par l'utilisation de la stigmergie comme processus d'échange d'information.

Il existe cependant un algorithme adapté au cas continu [Monmarché et al., 2000b] qui s'inspire du comportement de fourmis primitives (ce qui ne veut pas dire *inadaptées*) de l'espèce Pachycondyla apicalis, et qui ne fait *pas* usage de la communication indirecte par pistes de phéromone : l'algorithme *API*.

Dans cette méthode, on commence par positionner un nid aléatoirement sur l'espace de recherche, puis des fourmis sont distribuées aléatoirement dans l'espace. Ces fourmis vont alors explorer localement leur "site de chasse" en évaluant plusieurs points dans un périmètre donné (voir figure 1.6). Chaque fourmi mémorise le meilleur point trouvé. Si, lors de l'exploration de son site de chasse, elle trouve un meilleur point, alors elle reviendra sur ce site, sinon, après un certain nombre d'explorations, elle choisira un autre site. Une fois les explorations des sites de chasse terminées, des fourmis tirées au hasard comparent deux à deux (comme peuvent le faire les fourmis réelles durant le comportement de "tandem-running[3]") leurs meilleurs résultats, puis mémorisent le meilleur des deux sites de chasse. Le nid est finalement réinitialisé sur le meilleur point trouvé après un temps donné, la

[3]qu'on pourrait traduire par "course en couple"

Algorithme 1.2 Un algorithme de colonies de fourmis hybride pour le cas continu.

1. À chaque itération, sélectionner pour chaque fourmi une valeur initiale dans le groupe de valeurs candidates avec la probabilité :

$$p_{ij}^{k}(t) = \frac{\tau_{ij}(t)}{\sum \tau_{ir}(t)}$$

2. Utiliser des opérateurs de mutation et de croisement sur les m valeurs, afin d'obtenir m nouvelles valeurs ;
3. Ajouter ces nouvelles valeurs au groupe de valeurs candidates pour le composant $x_{i,e}$;
4. Les utiliser pour former m solutions de la nouvelle génération ;
5. Calculer la "fitness" de ces solutions ;
6. Quand m fourmis ont parcouru toutes les arêtes, mettre à jour les pistes de phéromone des valeurs candidates de chaque composant par :

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ir}(t) + \sum \tau_{ij}^{k}$$

7. Si la $k^{\text{ème}}$ fourmi choisit la $j^{\text{ème}}$ valeur candidate du groupe de composants, alors $\Delta\tau_{ij}^{k}(t+1) = Wf_k$, sinon $\Delta\tau_{ij}^{k} = 0$, en désignant par W une constante et par f_k la "fitness" de la solution trouvée par la $k^{\text{ème}}$ fourmi ;
8. Effacer les m valeurs ayant les plus basses intensités de phéromone dans chaque groupe de candidats.

mémoire des sites des fourmis est remise à zéro, et l'algorithme effectue une nouvelle itération.

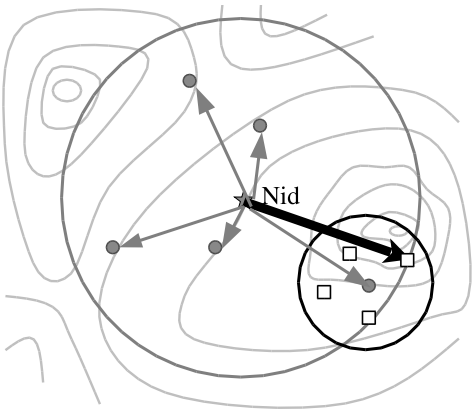


FIG. 1.6 – *L'algorithme API : une méthode à démarrage multiple inspirée par une espèce de fourmi primitive. Les fourmis (cercles pleins) explorent des sites de chasse (petits carrés) dans un périmètre (grand cercle) autour du nid. Le nid est déplacé sur le meilleur point au moment de la ré-initialisation (flèche en trait gras).*

***ACO* pour l'optimisation continue** Cet algorithme [Socha, 2004] tente de maintenir la construction itérative des solutions dans le cas de variables continues, en adoptant un point de vue différent des précédents. En effet, il prend le parti de considérer que les composants de toutes les solutions sont formés par les différentes variables optimisées. De plus, plutôt que de considérer l'algorithme du point de vue de la fourmi, il prend le parti de se placer au niveau de la colonie, les fourmis n'étant plus que des points à évaluer.

Dans cette méthode, dénommée simplement "ACO pour l'optimisation continue", on tire aléatoirement une population de fourmis dans une distribution de probabilité à chaque itération. De cet ensemble de points ne sont conservés que les meilleurs points, qui servent alors à construire une "meilleure" distribution de probabilité.

La distribution de probabilité utilisée est un "amalgame pondéré de noyaux normaux", soit un ensemble de distributions normales combinées :

$$P(x) = \sum_{j=1}^{k} w_j \cdot \left(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \right)$$

en désignant par k le nombre de noyaux utilisés, μ et σ^2 la moyenne et la variance d'un noyau et w_i la pondération.

Chaque distribution n'est utilisée que sur une variable, sans dépendance avec les autres. La modification des distributions est nommée "mise à jour de phéromone" et consiste à

Algorithme 1.3 Algorithmes du type colonies de fourmis, utilisant l'estimation de distribution pour l'optimisation en variables continues.

Construire la distribution de probabilité initiale :$\tau_i^0 = P_i^0(x_i), i \in \{1 \dots n\}$

Tant que (critère d'arrêt non atteint) :

Pour chaque fourmi, de $a = 1$ à m :

Pour chaque variable, de $i = 1$ à n :

Choisir aléatoirement une valeur x_i selon la distribution $P_i(x_i)$

Ajouter à la solution en construction :

$S^a = \left\{s_1^a, \dots, s_{i-1}^a\right\} \cup \{x_i\}$

Fin

Fin

Mémoriser les k meilleures solutions trouvées : $S^* = \left\{s_1^*, \dots, s_k^*\right\}$

Reconstruire la distribution de probabilité selon les meilleures solutions : $\tau = P(S^*)$

Fin

renforcer ou à diminuer l'influence des noyaux correspondant aux solutions. Le principe de la méthode est présenté sur l'algorithme 1.3.

L'algorithme *CACS* Cette méthode appelée "Continuous Ant Colony System" [Pourtakdoust and Nobahari, 2004] est très proche de la précédente, bien qu'ayant été présentée simultanément. En effet, dans *CACS* comme dans *ACO* pour l'optimisation continue, le coeur de l'algorithme consiste à faire évoluer une distribution de probabilité.

Le principe de la méthode est le même que celui présenté sur l'algorithme 1.3. Dans *CACS*, la distribution utilisée est dite "normale", mais la formule de la distribution diffère légèrement de la formule classique (équation 1.5) et la variance utilisée est en fait un nouvel indice de dispersion (voir équation 1.6).

$$P(x) = e^{-\frac{(x-x_{min})^2}{2\sigma^2}} \tag{1.5}$$

en désignant par x_{min} le mode de la distribution et σ^2 l'indice de dispersion suivant :

$$\sigma^2 = \frac{\sum_{j=1}^{m} \frac{1}{f_j - f_{min}} \left(x_j - x_{min}\right)^2}{\sum_{j=1}^{m} \frac{1}{f_j - f_{min}}} \tag{1.6}$$

en désignant par m le nombre de fourmis, f_j la valeur de la fonction associée à la fourmi j, et f_{min} la meilleure valeur trouvée.

Dans *CACS*, la seule distribution utilisée est centrée sur le mode de la distribution de l'itération précédente, et non sur la moyenne.

En résumé dans le domaine continu On a pu voir que deux types d'algorithmes sur cinq étaient en fait plus ou moins hybridés avec un algorithme de type évolutionnaire, et qu'un troisième n'utilisait pas la métaphore "classique" des colonies de fourmis. Les deux derniers algorithmes utilisent une approche différente des précédents, plus centrée sur l'aspect probabiliste que comportemental des algorithmes à colonies de fourmis. D'une manière générale, la recherche en est encore à ses débuts et les algorithmes produits n'ont pas atteint leur pleine maturité, et ne sont donc pas encore vraiment compétitifs par rapport à d'autres classes de métaheuristiques plus élaborées sur les problèmes continus. La présentation de notre contribution dans le domaine fait l'objet des chapitres 3 et 4.

1.3.5.2 Problèmes dynamiques

Un problème est dit dynamique s'il *varie* dans le temps, i.e. si la solution optimale ne possède pas les mêmes caractéristiques durant le temps de l'optimisation. Ces problèmes soulèvent des difficultés spécifiques, du fait qu'il faut approcher au mieux la meilleure solution à *chaque* pas de temps.

La première application des algorithmes de colonies de fourmis à des problèmes dynamiques concernait l'optimisation du routage sur des réseaux de type téléphonique [Schoonderwoerd et al., 1996]. Cependant l'algorithme proposé n'ayant pas fait l'objet d'études et de comparaisons approfondies, il est difficile d'en tirer des enseignements. Une autre application a également été exposée par White et al. [White et al., 1998, Bieszczad and White, 1999] sur des problèmes similaires. Une application a été décrite pour des problèmes de routage sur des réseaux de type Internet (voir figure 1.7) : l'algorithme AntNet [Di Caro and Dorigo, 1997]. Cette métaheuristique a fait l'objet de plusieurs études (voir notamment [Di Caro and Dorigo, 1998b]) et semble avoir prouvé son efficacité sur plusieurs problèmes tests.

Chacun de ces algorithmes utilise — pour mettre à jour des tables de routage probabilistes — des fourmis pour explorer le réseau, de façon à ce que l'information pertinente soit la fréquence de passage des fourmis sur chaque noeud. D'une manière générale, l'aspect distribué et flexible des algorithmes de colonies de fourmis semble très bien adapté à des problèmes dynamiques. Notre contribution en la matière sera présentée en détail dans le chapitre 3.

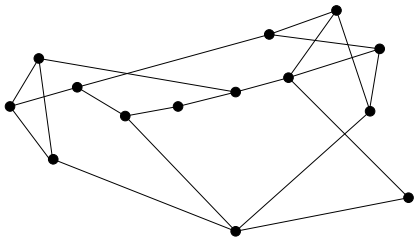


FIG. 1.7 – *Exemple de réseau utilisé pour tester l'algorithme AntNet :* NFSNET *(chaque arête représente une liaison orientée).*

1.4 Métaheuristiques et éthologie

Les métaheuristiques sont la plupart du temps issues de *métaphores* provenant de la nature, et notamment de la biologie [Dréo and Siarry, 2003a]. Les algorithmes de colonies de fourmis sont inspirés par le comportement d'insectes sociaux, mais ce ne sont pas les seuls algorithmes à être issus de l'étude du comportement animal (*éthologie*). En effet, l'optimisation par essaim particulaire ("Particle Swarm Optimization" [Eberhart et al., 2001], voir section 2.3.1) est issue d'une analogie avec les comportements collectifs de déplacements d'animaux, tels qu'on peut les observer dans les bancs de poissons ou les vols d'oiseaux ; il existe également des algorithmes inspirés des comportements des abeilles [Choo, 2000, Panta, 2002]. On trouve encore des algorithmes s'inspirant de certains aspects du comportement des insectes sociaux, bien que ne faisant pas usage des caractéristiques *classiques* des algorithmes de colonies de fourmis (voir par exemple [De Wolf et al., 2002, Nouyan, 2002]).

Dans le chapitre suivant, nous verrons comment plusieurs concepts théoriques — notamment d'inspiration biologique — peuvent également aider à la conception de nouvelles métaheuristiques.

Chapitre 2

Concepts théoriques exploités dans notre travail

2.1 Auto-organisation

2.1.1 Introduction

La biologie est la source d'inspiration de nombreuses métaheuristiques. Ainsi, les théories de l'évolution ont inspiré les algorithmes évolutionnaires [Goldberg, 1994], les phénomènes de suivi de piste chez les fourmis ont conduit à l'élaboration des algorithmes de colonies de fourmis [Bonabeau et al., 1999], l'étude de l'organisation de groupes d'animaux a donné naissance aux méthodes d'optimisation par essaim particulaire [Eberhart et al., 2001]. Il existe, en outre, d'autres algorithmes, moins connus que ceux que nous venons de citer, qui découlent de la biologie : en particulier, des algorithmes inspirés du fonctionnement du système immunitaire [De Castro and Von Zuben, 1999], des algorithmes pour l'allocation dynamique de tâches [Cicirello and Smith, 2001], s'appuyant sur des modèles d'organisation du travail chez les fourmis, des algorithmes de classification suggérés par les essaims d'insectes [Aupetit et al., 2003].

Une contribution importante de la biologie dans ce domaine vient de la *théorie de l'auto-organisation* [Camazine et al., 2000, p.8], qui permet d'analyser les propriétés de plusieurs métaheuristiques issues des métaphores biologiques. Cette théorie (notamment étudiée en dehors de la biologie [Nicolis and Prigogine, 1977]) décrit les conditions d'apparition de phénomènes complexes à partir de systèmes distribués, dont les agents font l'objet d'interactions simples, mais nombreuses. La théorie met en avant des concepts tels que la communication, les rétroactions, l'amplification des fluctuations et l'émergence. L'intelligence en essaim est ainsi née sur deux fronts : via une approche "systèmes auto-

organisés" (ayant donné lieu aux algorithmes de colonies de fourmis) et via une approche "systèmes socio-cognitifs" (ayant conduit à l'optimisation par essaim particulaire).

Nous proposons de mettre la théorie de l'auto-organisation en relation avec le concept de *programmation à mémoire adaptative* [Taillard, 1998], qui tente de décrire les points clefs des métaheuristiques modernes, en insistant notamment sur le rôle de la mémoire et des mécanismes d'intensification et de diversification [Dréo and Siarry, 2003b] [Dréo and Siarry, 2003c].

Plus généralement, nous pensons que la théorie de l'auto-organisation combinée à la programmation à mémoire adaptative donne des clefs pour concevoir les composants de base de métaheuristiques relevant de l'intelligence en essaim.

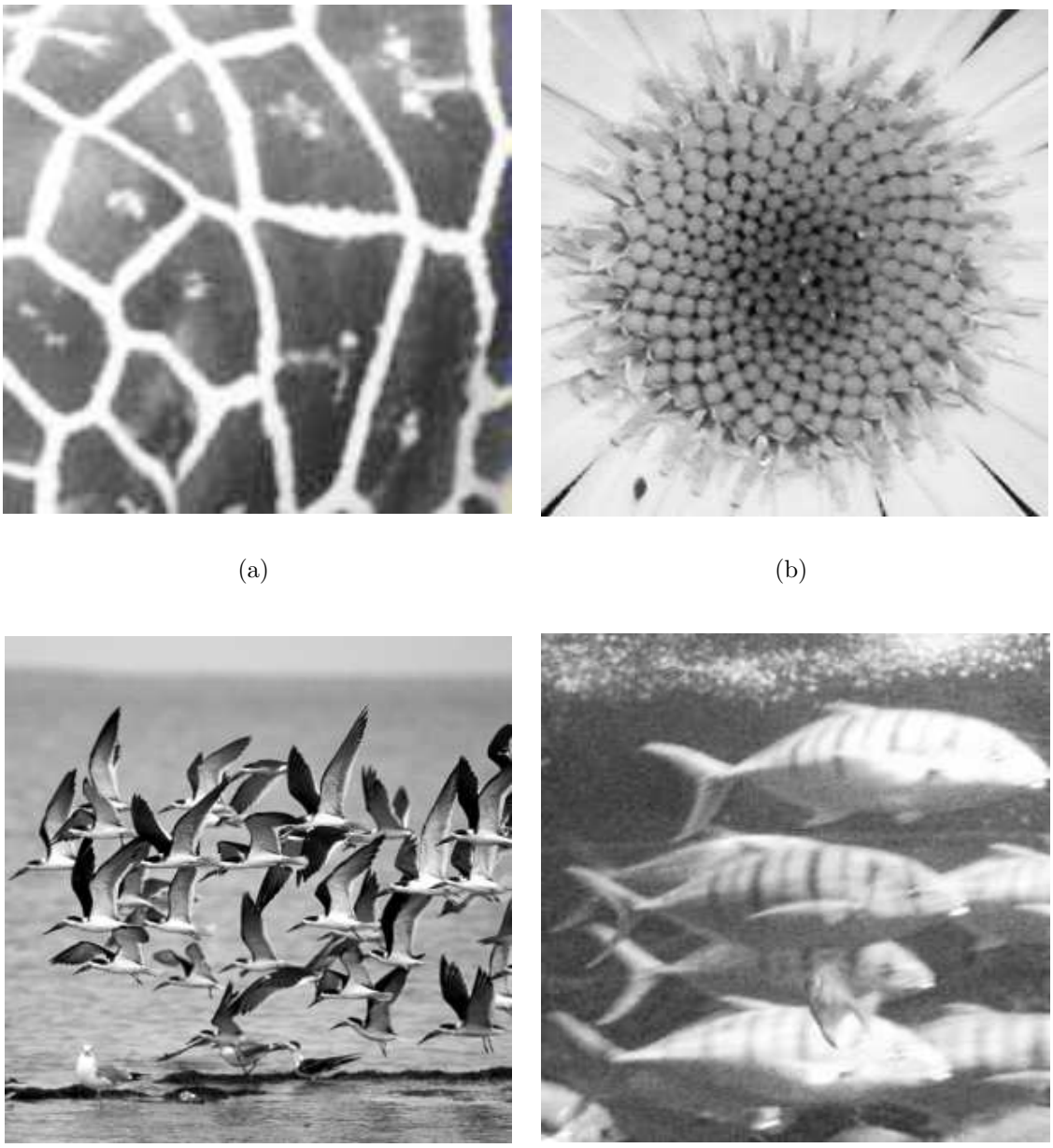
2.1.2 Auto-organisation

L'*auto-organisation* est un phénomène décrit dans plusieurs disciplines, notamment en physique [Prigogine and Glandsdorf, 1971, Nicolis and Prigogine, 1977] et en biologie . Étant donné que les métaheuristiques présentées dans ce chapitre sont inspirées de phénomènes biologiques, nous avons choisi de considérer une définition dans ce cadre. Une définition claire a été proposée [Camazine et al., 2000, p.8], que nous avons traduite comme suit :

> *L'auto-organisation caractérise un processus au cours duquel une* structure émerge *au niveau global uniquement d'un grand nombre d'interactions entre les composants de niveau local du système. De plus, les règles spécifiant les interactions entre composants du système sont suivies en utilisant uniquement des informations locales, sans référence au modèle global.*

Deux termes sont à préciser pour une bonne compréhension, "structure" et "émerger". Le mot *structure* est une traduction approximative du mot anglais "pattern", qui déborde la notion de modèle, et peut signifier aussi gestalt, configuration générale, forme, schéma, type [Meinhardt, 1982]. D'une manière générale, il s'applique à un "arrangement organisé d'objets dans l'espace ou le temps". La figure 2.1 présente des exemples de *structures* observables dans la nature. Une propriété *émergente* d'un système est quant à elle une caractéristique qui apparaît "à l'improviste" (sans avoir été *explicitement* déterminée), de par les interactions entre les composants de ce système. Nous parlerons donc d'émergence pour souligner le caractère non-déterminé d'une propriété, sans faire référence au fait que cette propriété soit organisée en *structures,* ni qu'elle soit située à un niveau différent des interactions.

La question cruciale est donc de comprendre comment les composants d'un système interagissent entre eux pour produire une *structure* complexe (au sens relatif du terme,



(a) (b)

(c) (d)

FIG. 2.1 – *Exemples de* structures *observables dans des systèmes biologiques. (a) motifs de la robe d'une girafe réticulée (U.S. Fish and Wildlife Service, Gary M. Stolz), (b) double spirale de Fibonacci au coeur d'une pâquerette, (c) groupe d'oiseaux en vol, (d) poissons assemblés en banc.*

i.e. *plus* complexe que les composants eux-mêmes). Un certain nombre de phénomènes nécessaires ont été identifiés : ce sont les processus de *rétroaction* et la gestion des *flux d'informations*.

Les *rétroactions positives* sont des processus dont le résultat renforce l'action, par exemple par amplification, facilitation, auto-catalyse, etc. Les rétroactions positives sont capables d'amplifier les *fluctuations* du système, permettant la mise à jour d'informations peu apparentes. De tels processus peuvent facilement entraîner une "explosion" du système, s'ils ne sont pas maintenus sous contrôle par des *rétroactions négatives*, qui jouent ainsi le rôle de stabilisateurs du système. Lorsqu'ils sont couplés, de tels processus de rétroaction sont de puissants générateurs de modèles [Camazine et al., 2000].

Dans le cadre de la biologie du comportement, il est aisé de comprendre que les interactions entre les composants d'un système vont très souvent mettre en jeu des processus de *communication*, de transfert d'informations entre individus. D'une manière générale, les individus peuvent communiquer, soit par le biais de signaux, c'est-à-dire en utilisant un moyen spécifique pour porter une information, soit par le biais d"'indices", où l'information est portée accidentellement [Seeley, 1989]. De même, l'information peut provenir directement d'autres individus, ou bien passer par le biais de l'état d'un travail en cours. Cette deuxième possibilité d'échange des informations, par le biais de modifications de l'environnement, se nomme la *stigmergie* [Grassé, 1959, Theraulaz and Bonabeau, 1995].

D'une manière générale, tous ces processus sont plus ou moins inter-connectés, permettant à un système constitué d'un grand nombre d'individus agissant de concert de résoudre des problèmes trop complexes pour un individu unique.

Certaines caractéristiques des systèmes auto-organisés sont particulièrement intéressantes, en particulier leur *dynamisme*, ou encore leur capacité à produire des modèles *stables*. Dans le cadre de l'étude du comportement des insectes sociaux, certains concepts liés au principe de l'auto-organisation méritent d'être soulignés : la *décentralisation* intrinsèque de ces systèmes, leur organisation en *hétérarchie dense* et l'utilisation récurrente de la *stigmergie*. En effet, ces concepts sont parfois utilisés comme autant d'angles de vue différents sur un même problème et recouvrent une partie des principes de l'auto-organisation.

2.1.2.1 Stigmergie

La stigmergie est un des concepts à la base de la création des métaheuristiques de colonies de fourmis. Elle est précisément définie comme une "forme de communication passant par le biais de modifications de l'environnement", mais on peut rencontrer le terme "interactions sociales indirectes" pour décrire le même phénomène. Un exemple d'utilisation de la stigmergie est décrit dans la section 1.3.1. La spécificité de la stigmergie

est que les individus échangent des informations par le biais du travail en cours, de l'*état* d'avancement de la tâche globale à accomplir.

2.1.2.2 Contrôle décentralisé

Dans un système auto-organisé, il n'y a pas de prise de décision à un niveau donné, suivie d'ordres et d'actions pré-déterminées. En effet, dans un système décentralisé, chaque individu dispose d'une vision *locale* de son environnement, et ne connaît donc pas le problème dans son ensemble. La littérature des systèmes multi-agents (voir [Ferber, 1997] pour une première approche) emploie souvent ce terme ou celui "d'intelligence artificielle distribuée" [Jennings, 1996], bien que, d'une manière générale, l'étude des systèmes multi-agents tende à utiliser des modèles de comportement plus complexes, fondés notamment sur les sciences de la cognition. Les avantages d'un contrôle décentralisé sont notamment la *robustesse* et la *flexibilité* [Bonabeau et al., 1999] : systèmes robustes, car capables de continuer à fonctionner en cas de panne d'une de leurs composantes ; flexibles, car efficaces sur des problèmes dynamiques.

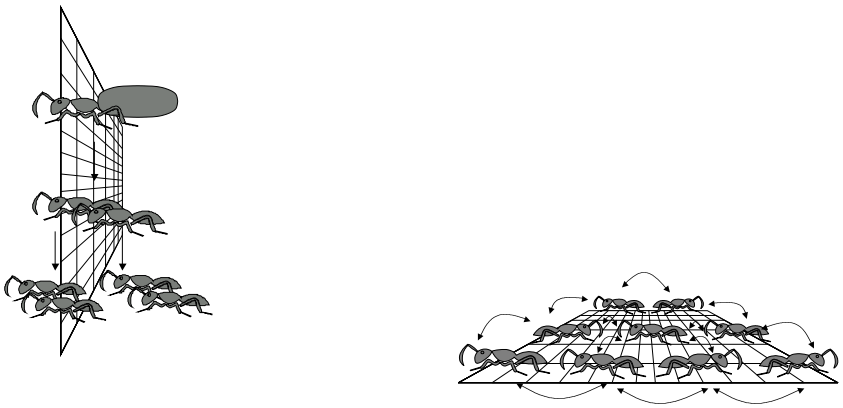
2.1.2.3 Hétérarchie dense

L'hétérarchie dense est un concept issu directement de la biologie, utilisé pour décrire l'organisation des insectes sociaux, et plus particulièrement des colonies de fourmis. La définition donnée dans [Wilson and Hölldobler, 1988] peut être traduite comme suit :

> *[Une] colonie de fourmis est une variante particulière de hiérarchie qui peut avantageusement être appelée une* hétérarchie*. Cela signifie que les propriétés des niveaux globaux agissent sur les niveaux locaux, mais que l'activité induite dans les unités locales influence en retour les niveaux globaux.*

L'hétérarchie est dite *dense* dans le sens où un tel système forme un réseau hautement connecté, où chaque individu peut échanger des informations avec n'importe quel autre. Ce concept est en quelque sorte opposé à celui de *hiérarchie* où, dans une vision populaire mais erronée, la reine gouvernerait ses sujets en faisant passer des ordres dans une structure *verticale*, alors que, dans une *hétérarchie*, la structure est plutôt *horizontale* (figure 2.2).

On constate que ce concept recoupe celui de contrôle décentralisé, mais aussi celui de stigmergie, en ce sens que l'hétérarchie décrit la *manière* dont le flux d'information parcourt le système. Cependant, dans une hétérarchie dense, tout type de communication doit être pris en compte, tant la stigmergie que les échanges directs entre individus.



(a) (b)

FIG. 2.2 – *Hiérarchie (a) et hétérarchie dense (b) : deux concepts opposés.*

2.2 Programmation à mémoire adaptative (*PMA*)

Le concept de programmation à mémoire adaptative (*PMA*) [Taillard, 1998, Taillard et al., 1998] est né de l'observation que les métaheuristiques récentes tendaient à devenir proches. Ainsi, du point de vue de la programmation à mémoire adaptative, certaines métaheuristiques partagent maintenant une démarche commune, présentée dans l'algorithme 2.1.

Algorithme 2.1 Démarche employée par un programme à mémoire adaptative.

1. Mémorisation d'un jeu de solutions ou une structure de données rassemblant les particularités des solutions produites par la recherche,
2. construction d'une solution provisoire sur la base des données mémorisées,
3. amélioration de la solution par un algorithme de recherche locale,
4. mémorisation de la nouvelle solution ou de la structure de données associée.

La notion de programmation à mémoire adaptative insiste sur trois concepts fondamentaux : la *mémoire*, l'*intensification* et la *diversification*. Dans la littérature des algorithmes évolutionnaires, ces deux dernières notions sont souvent respectivement désignées par les termes *exploitation* et *exploration*, ayant un sens similaire. Nous avons choisi ici d'utiliser les termes de la définition originale de la programmation à mémoire adaptative, employés plus généralement dans la littérature de la recherche avec tabous ou du recuit simulé.

La mémoire représente ici l'information récoltée par l'algorithme, sur laquelle il va s'appuyer pour effectuer sa recherche. La mémoire est présente sous de nombreuses formes possibles, de la plus simple (une population de points) à des structures plus complexes (les pistes de phéromone des algorithmes de colonies de fourmis). La mémoire est une

modélisation des solutions, elle est un moyen de décrire la fonction objectif. Cette modélisation s'opère par une distribution sur l'espace de recherche, biaisée vers les meilleures solutions, qui évolue vers les optima du problème. Cette mémoire peut être définie comme globale (par rapport au problème dans son ensemble) ou inter-individuelle (d'une solution relativement à une autre).

L'intensification consiste en l'utilisation des informations disponibles pour améliorer la pertinence de celles-ci. Du point de vue des métaheuristiques, il s'agit généralement tout simplement de recherche locale. Les algorithmes de recherche locale sont maintenant souvent employés en association avec d'autres métaheuristiques plus complexes, donnant lieu à des algorithmes *hybrides* [Talbi, 2002]. On rencontre ainsi souvent l'algorithme du "simplexe" de Nelder-Mead [Nelder and Mead, 1965], mais des métaheuristiques plus complexes, comme la recherche avec tabous, sont parfois employées.

La diversification est la recherche de nouvelles informations, afin d'augmenter la connaissance du problème. Elle fait souvent appel à des méthodes stochastiques, et il est pour le moment difficile de dégager des idées générales, tant la diversité d'approches de cette composante des métaheuristiques est grande.

En pratique, les trois composantes de la *PMA* sont liées, et il est parfois difficile de les repérer distinctement dans les métaheuristiques proposées. De fait, les métaheuristiques tentent d'équilibrer la balance entre diversification et intensification, et bien souvent les améliorations d'une métaheuristique existante consistent à faire pencher la balance dans un sens ou dans l'autre.

Le concept de programmation à mémoire adaptative se veut une forme de généralisation du mode de fonctionnement des métaheuristiques. Certaines métaheuristiques ont un mode de fonctionnement qui semble d'emblée très proche de la *PMA*, c'est le cas par exemple de la méthode *GRASP* ("Greedy Randomized Adaptive Search Procedure") [Resende, 2000] ou encore des algorithmes à estimation de distribution (*EDA*) [Larrañaga and Lozano, 2002]. Cependant, la *PMA* propose une approche généraliste, sans entrer dans les détails de l'implémentation, qui induisent souvent des a priori sur la façon d'aborder tel ou tel problème. C'est le cas par exemple des *EDA*, où la phase de diversification utilise un tirage aléatoire dans une loi donnée, il y a donc un a priori fort sur le type de loi utilisée, qui dépend en pratique du problème abordé. La programmation à mémoire adaptative tente d'unifier différentes métaheuristiques.

2.3 Bases communes et exemples de métaheuristiques fondées sur ces principes

L'auto-organisation nous renseigne sur la structure à employer pour concevoir des métaheuristiques flexibles et capables de s'adapter à un problème donné. En effet, la grande qualité de tels systèmes est de construire des comportements complexes à partir de règles simples, en se fondant sur une architecture fortement *décentralisée* (on rencontre également le terme de *parallèle* ou *distribuée*), maintenant reconnue et utilisée pour sa flexibilité et son efficacité [Rudolph, 1992, Pardalos et al., 1996].

La programmation à mémoire adaptative nous renseigne quant à elle sur les méthodes employées par des métaheuristiques efficaces. Les auteurs insistent d'ailleurs sur les qualités de la *PMA* : parallélisme et flexibilité [Taillard et al., 1998]. Nous proposons d'établir un pont entre ces deux théories, qui ont été établies séparément et dans des contextes éloignés.

Les points communs entre ces deux théories sont en effet nombreux et, du point de vue de la conception des métaheuristiques, il existe une relation simple entre elles : la *PMA* décrit le "but" à atteindre, et la théorie de l'auto-organisation un "moyen" pour atteindre ce but. Ainsi, une métaheuristique efficace devrait, selon la programmation à mémoire adaptative, mettre en place des mécanismes de mémoire, d'intensification et de diversification, reste la question des moyens à utiliser pour mettre en place ces mécanismes. L'auto-organisation propose un modèle de réalisation : un algorithme à base de population définissant des interactions simples au niveau local, permettant l'émergence d'un comportement complexe au niveau global.

Cette section présente quelques métaheuristiques inspirées de la biologie et utilisant des phénomènes d'auto-organisation. Nous avons délibérément choisi de restreindre la liste des métaheuristiques présentées à des classes d'algorithmes parmi les plus connues, afin d'éviter une énumération peu pertinente et forcément incomplète. De plus, certaines classes de métaheuristiques se recoupent entre elles, la classification proposée ici est celle la plus communément admise. Nous laissons de côté les algorithmes de colonies de fourmis, déjà décrits plus haut.

2.3.1 Optimisation par essaim particulaire

L'optimisation par essaim particulaire ("Particle Swarm Optimization", *PSO*) [Kennedy and Eberhart, 1995, Eberhart et al., 2001] est issue d'une analogie avec les comportements collectifs de déplacements d'animaux (la métaphore a de plus été largement agrémentée de socio-psychologie). En effet, chez certains groupes d'animaux, comme les bancs de poissons, on peut observer des dynamiques de déplacements relativement com-

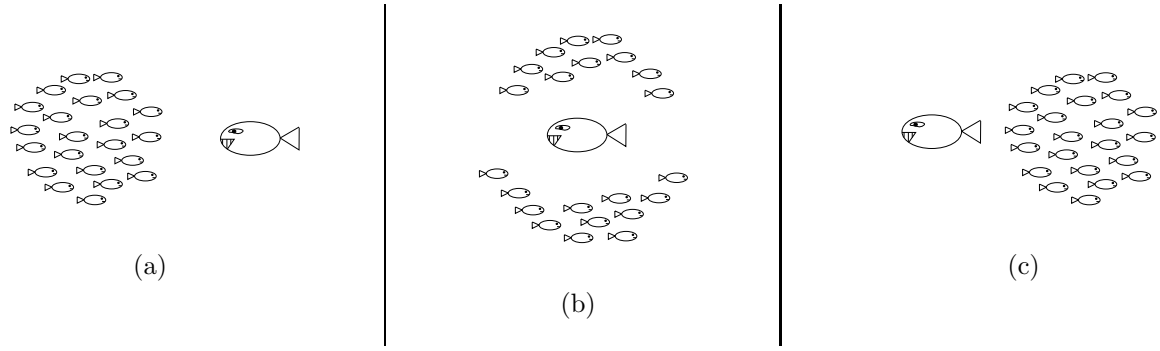


FIG. 2.3 – *Schéma de l'évitement d'un prédateur par un banc de poissons. (a) le banc forme un seul groupe, (b) les individus évitent le prédateur en formant une structure en "fontaine", (c) le banc se reforme.*

plexes, alors que les individus eux-mêmes n'ont accès qu'à des informations limitées, comme la position et la vitesse de leurs plus proches voisins. On peut par exemple observer qu'un banc de poissons est capable d'éviter un prédateur : d'abord en se divisant en deux groupes, puis en reformant le banc originel (voir figure 2.3), tout en maintenant la cohésion du banc.

Ces comportements collectifs s'inscrivent tout à fait dans la théorie de l'auto-organisation. Pour résumer, chaque individu utilise l'information locale à laquelle il peut accéder sur le déplacement de ses plus proches voisins pour décider de son propre déplacement. Des règles très simples, comme "rester relativement proche des autres individus", "aller dans la même direction", "à la même vitesse" suffisent à maintenir la cohésion du groupe tout entier, et à permettre des comportements collectifs complexes et adaptés.

Les auteurs de la méthode d'optimisation par essaim particulaire se sont inspirés de ces comportements en mettant en perspective la théorie de la socio-psychologie sur le traitement de l'information et les prises de décisions dans des groupes sociaux. Fait exceptionnel et remarquable, cette métaheuristique a été conçue d'emblée dans le cas continu, et c'est toujours dans ce domaine que se situent la majorité des applications à ce jour. La méthode en elle-même met en jeu de larges groupes de *particules* sous forme de vecteurs se déplaçant sur l'espace de recherche. Chaque particule i est caractérisée par sa *position* $\vec{x}_i$ et un vecteur de changement de position (appelé *vélocité*) $\vec{v}_i$. À chaque itération, la particule se déplace : $\vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t-1)$. Le coeur de la méthode consiste à choisir comment définir $\vec{v}_i$. La socio-psychologie suggère que des individus se déplaçant (dans une carte socio-cognitive) sont influencés par leur comportement passé et par celui de leurs voisins (voisins dans le réseau social et non nécessairement dans l'espace). On tient donc compte, dans la mise à jour, de la position des particules, de la direction de leur mouvement (leur vitesse), la meilleure position précédente $\vec{p_i}$ et la meilleure position

$\vec{p_g}$ parmi leurs "voisins" :

$$\vec{x}_i(t) = f\left(\vec{x}_i(t-1), \vec{v}_i(t-1), \vec{p}_i, \vec{p}_g\right)$$

Le changement de position s'effectue comme suit :

$$\begin{cases} \vec{v}_i(t) = \vec{v}_i(t-1) + \varphi_1\left(\vec{p}_i - \vec{x}_i(t-1)\right) + \varphi_2\left(\vec{p}_g - \vec{x}_i(t-1)\right) \\ \vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t-1) \end{cases}$$

où les paramètres φ_n sont des variables aléatoires tirées dans $U_{[0,\varphi_{max}]}$ et qui ont pour rôle de pondérer les rôles relatifs de l'*expérience individuelle* (φ_1) et de la *communication sociale* (φ_2). C'est la notion de "compromis psycho-social", mise en avant par les auteurs. Le tirage aléatoire uniforme est justifié si l'on ne considère aucun a priori sur l'importance de l'une ou l'autre source d'informations. Pour éviter que le système n'"explose" en cas d'amplification trop grande d'oscillations, un paramètre V_{max} permet de limiter la vitesse (sur chaque dimension). Le pseudo-code pour la version la plus générale de l'algorithme — la version continue — est présenté sur l'algorithme 2.2.

Des améliorations peuvent être apportées à cet algorithme de base, notamment du point de vue du contrôle de la divergence. Dans la version de base, c'est le paramètre V_{max} qui va empêcher le système d'"exploser" par amplification des rétroactions positives. L'utilisation de "coefficients de constriction" permet de mieux contrôler ce comportement [Clerc and Kennedy, 2002]. Il s'agit ici d'ajouter, aux différents membres des équations de vélocité, des coefficients pouvant être manipulés pour contrôler la dynamique intensification/diversification du système. Dans la version "plus simple constriction", un seul coefficient multiplie les deux membres de l'équation du calcul de la vélocité. Cette méthode de constriction permet de provoquer une convergence de l'algorithme (l'amplitude des mouvements des individus diminue jusqu'à s'annuler). L'algorithme réalise un compromis efficace entre intensification et diversification ; la seule restriction apparaît si les points p_i et p_g sont éloignés, auquel cas les particules continueront d'osciller entre ces deux points, sans converger. Une particularité intéressante est que, si un nouvel optimum est découvert alors que l'algorithme a convergé (donc, après une phase d'intensification), les particules iront explorer la région du nouveau point (phase de diversification).

Dans le même ordre d'idées, une version met en place un *poids d'inertie* ("Inertia Weight"), en multipliant chaque membre de l'équation de vélocité par un coefficient différent [Shi and Eberhart, 1998]. Pour résumer, le poids d'inertie décroît en fonction du temps, ce qui provoque une convergence contrôlable par ce paramètre. La dynamique générale reste la même que dans la version avec coefficient de constriction, à l'exception

Algorithme 2.2 Optimisation par essaim particulaire (en variables continues).

$n =$**nombre** d'individus

$D =$**dimensions** du problème

Tant que critère d'arrêt :

Pour $i = 1$ à n :

Si $F(\vec{x}_i) > F(\vec{p}_i)$ alors :

Pour $d = 1, \ldots, D$:

$p_{id} = k_{id}$ // p_{id} *est donc le meilleur individu trouvé*

fin d

fin si

$g = i$

Pour $j =$index des voisins :

Si $F(\vec{p}_j) > F(\vec{p}_g)$ alors :

$g = j$ // g *est le meilleur individu du voisinage*

fin si

fin j

Pour $d = 1, \ldots, D$:

$v_{id}(t) = v_{id}(t-1) + \varphi_1 \left(p_{id} - x_{id}\left(t-1\right)\right) + \varphi_2 \left(p_{gd} - x_{id}\left(t-1\right)\right)$

$v_{id} \in (-V_{max'} + V_{max})$

$x_{id}(t) = x_{id}(t-1) + v_{id}(t)$

fin d

fin i

fin

près de l'impossibilité de repartir dans une dynamique de diversification, si un nouveau meilleur point est trouvé.

Un autre paramètre important est la notion de voisinage, appliquée aux particules. Il semble être communément admis qu'un voisinage *social* (un individu x_2 ayant par exemple pour voisins les individus x_1 et x_3, quelles que soient les localisations spatiales de x_1, x_2, x_3) donne de meilleurs résultats qu'un voisinage *spatial* (fonction de la proximité des individus dans l'espace de recherche, par exemple).

Des variantes ont également vu le jour, modifiant la notion de *meilleure position précédente* d'une particule par celle de meilleure position du centre de gravité de groupes d'individus sélectionnés dans la population [Kennedy, 2000]. L'influence de la distribution initiale des particules a également été étudiée [Shi and Eberhart, 1999].

Ici, les rétroactions positives sont mises en place au niveau de l'attirance des particules les unes pour les autres. Les limitations de déplacement de chaque particule entre deux itérations forment les rétroactions négatives. La mémoire est structurée au niveau local, entre particules voisines ; à chaque itération, chaque particule n'évolue qu'en fonction de ses proches voisins, et non pas selon l'état global de la population à l'itération précédente.

On trouvera un état de l'art complet sur l'optimisation par essaim particulaire et les concepts qui lui sont associés dans [Eberhart et al., 2001] ainsi qu'une synthèse en français dans [Clerc, 2002] et une présentation très détaillée dans [Clerc, 2004].

2.3.2 Algorithmes évolutionnaires

Cette section est un extrait du livre [Dréo et al., 2003], section notamment rédigée par A. Pétrowski.

Les algorithmes évolutionnaires (AE) sont des techniques de recherche inspirées par l'évolution biologique des espèces, apparues à la fin des années 1950 [Fraser, 1957]. Parmi plusieurs approches [Holland, 1962, Fogel et al., 1966, Rechenberg, 1965], les algorithmes génétiques (AG) en constituent certainement l'exemple le plus connu, à la suite de la parution en 1989 du célèbre livre de D. E. Goldberg [Goldberg, 1989] : *Genetic Algorithms in Search, Optimization and Machine Learning* (voir en [Goldberg, 1994] la traduction française).

Le principe d'un algorithme évolutionnaire se décrit simplement (voir figure 2.4). Un ensemble de N points dans un espace de recherche, choisis a priori au hasard, constituent la *population* initiale ; chaque individu x de la population possède une certaine performance, qui mesure son degré d'*adaptation* à l'objectif visé : dans le cas de la minimisation d'une fonction objectif f, x est d'autant plus performant que $f(x)$ est plus petit. Un AE consiste à faire évoluer progressivement, par *générations* successives, la composition de la population, en maintenant sa taille constante. Au cours des générations, l'objectif

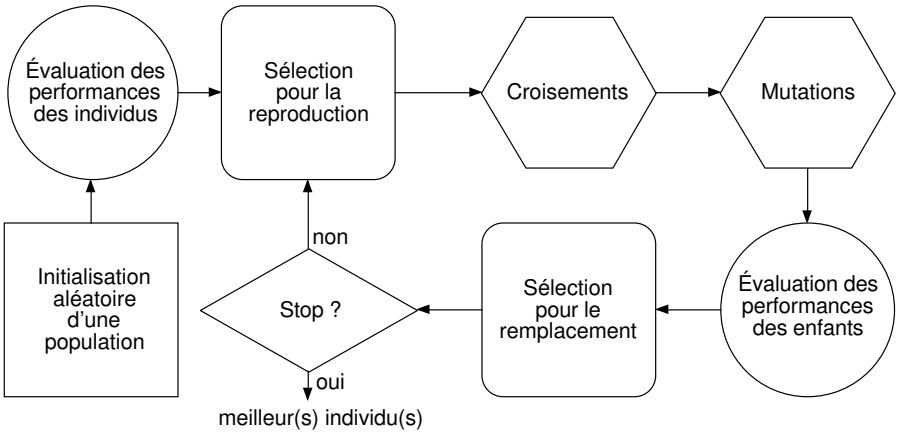


FIG. 2.4 – *Principe d'un algorithme évolutionnaire standard (figure de A. Pétrowski, extraite du livre [Dréo et al., 2003]).*

est d'améliorer globalement la performance des individus ; on s'efforce d'obtenir un tel résultat en mimant les deux principaux mécanismes qui régissent l'évolution des êtres vivants, selon la théorie de C. Darwin :

— la *sélection*, qui favorise la reproduction et la survie des individus les plus performants,

— et la *reproduction*, qui permet le brassage, la recombinaison et les variations des caractères héréditaires des parents, pour former des descendants aux potentialités nouvelles.

En pratique, une représentation doit être choisie pour les individus d'une population. Classiquement, un individu pourra être une liste d'entiers pour des problèmes combinatoires, un vecteur de nombres réels pour des problèmes numériques dans des espaces continus, une chaîne de nombres binaires pour des problèmes booléens, ou pourra même, au besoin, combiner ces représentations dans des structures complexes. Le passage d'une génération à la suivante se déroule en quatre phases : une phase de sélection, une phase de reproduction (ou de variation), une phase d'évaluation des performances et une phase de remplacement. La phase de sélection désigne les individus qui participent à la reproduction. Ils sont choisis, éventuellement à plusieurs reprises, a priori d'autant plus souvent qu'ils sont performants. Les individus sélectionnés sont ensuite disponibles pour la phase de reproduction. Celle-ci consiste à appliquer des opérateurs de variation sur des copies des individus sélectionnés, pour en engendrer de nouveaux ; les opérateurs les plus utilisés sont le *croisement* (ou *recombinaison*), qui produit un ou deux descendants à partir de deux parents, et la *mutation*, qui produit un nouvel individu à partir d'un seul individu. La structure des opérateurs de variation dépend étroitement de la représentation choisie pour les individus. Les performances des nouveaux individus sont ensuite mesurées, du-

rant la phase d'évaluation, à partir des objectifs fixés. Enfin, la phase de remplacement consiste à choisir les membres de la nouvelle génération : on peut, par exemple, remplacer les individus les moins performants de la population par les meilleurs individus produits, en nombre égal. L'algorithme est interrompu après un certain nombre de générations, selon un critère d'arrêt à préciser.

Dans cette famille de métaheuristiques [Baeck et al., 2000a, Baeck et al., 2000b], les rétroactions sont parfois difficiles à cerner, tant les variantes sont nombreuses. D'une façon générale, les rétroactions positives sont implémentées sous la forme d'opérateurs de type sélection, alors que les rétroactions négatives sont typiquement mises en place par des opérateurs de mutation. La mémoire est située au niveau local, l'évolution de chaque individu d'une itération à l'autre étant liée à l'évolution des individus voisins.

Le livre [Goldberg, 1995] constitue une bonne introduction aux algorithmes évolutionnaires.

2.3.3 Systèmes immunitaires artificiels

Le terme "système immunitaire artificiel" ("Artificial Immune System", *AIS*) s'applique à une vaste gamme de systèmes différents, notamment aux métaheuristiques d'optimisation inspirées du fonctionnement du système immunitaire des vertébrés. Un grand nombre de systèmes ont été conçus dans plusieurs domaines différents, tels que la robotique, la détection d'anomalies ou l'optimisation (voir [De Castro and Von Zuben, 2000] pour un survol de différentes applications).

Le système immunitaire est responsable de la protection de l'organisme contre les "agressions" d'organismes extérieurs. La métaphore dont sont issus les algorithmes *AIS* met l'accent sur les aspects d'*apprentissage* et de *mémoire* du système immunitaire dit *adaptatif* (par opposition au système dit *inné*), notamment via la discrimination entre le *soi* et le *non-soi*. En effet, les cellules vivantes disposent sur leurs membranes de molécules spécifiques dites "antigènes". Chaque organisme dispose ainsi d'une identité unique, déterminée par l'ensemble des antigènes présents sur ses cellules. Les *lymphocytes* (un type de globule blanc) sont des cellules du système immunitaire qui possèdent des *récepteurs* capables de se lier spécifiquement à un antigène unique, permettant ainsi de reconnaître une cellule étrangère à l'organisme. Un lymphocyte ayant ainsi reconnu une cellule du non-soi va être stimulé à proliférer (en produisant des clones de lui-même) et à se différencier en cellule permettant de garder en mémoire l'antigène, ou en cellule permettant de combattre les agressions. Dans le premier cas, il sera capable de réagir plus rapidement à une nouvelle exposition à l'antigène : c'est le principe même de l'efficacité des vaccins. Dans le second cas, le combat contre les agressions est possible grâce à la production d'anticorps. La figure 2.5 résume ces principales étapes. Il faut également no-

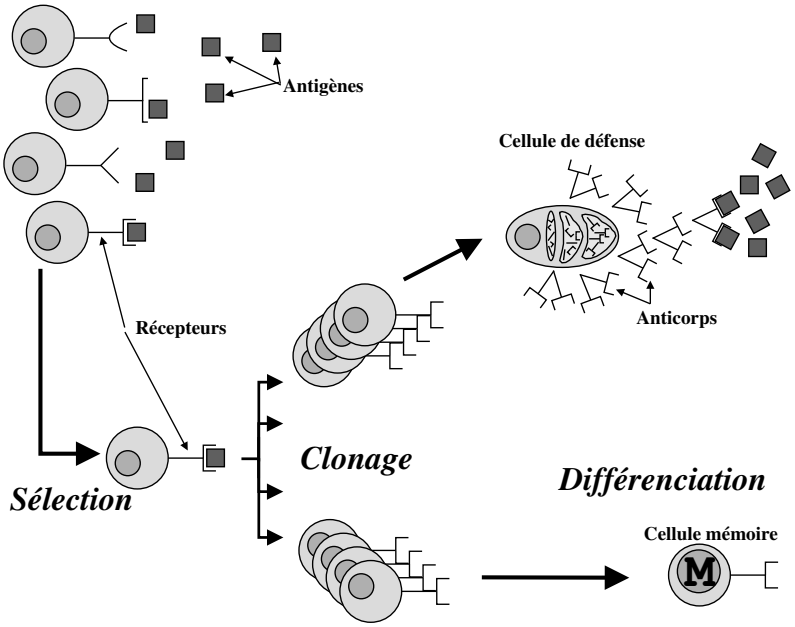


FIG. 2.5 – *La sélection par clonage : des lymphocytes, présentant des récepteurs spécifiques d'un antigène, se différencient en cellule mémoire ou en cellule participant à la défense active de l'organisme par le biais d'anticorps.*

ter que la diversité des récepteurs dans l'ensemble de la population des lymphocytes est, quant à elle, produite par un mécanisme d'*hyper-mutation* des cellules clonées.

Les principales idées utilisées pour la conception de la métaheuristique sont les sélections opérées sur les lymphocytes, accompagnées par les rétroactions positives permettant la multiplication et la mémoire du système. En effet, ces caractéristiques sont capitales pour maintenir les caractéristiques auto-organisées du système.

L'approche utilisée dans les algorithmes *AIS* est très proche de celle des algorithmes évolutionnaires, mais a également été comparée à celle des réseaux de neurones. On peut, dans le cadre de l'optimisation difficile, considérer les *AIS* comme une forme d'algorithme évolutionnaire présentant des opérateurs particuliers. Pour opérer la sélection, on se fonde par exemple sur une mesure d'affinité (i.e. entre le récepteur d'un lymphocyte et un antigène), la mutation s'opère quant à elle via un opérateur d'hyper-mutation directement issu de la métaphore. Au final, l'algorithme obtenu est très proche d'un algorithme génétique (voir algorithme 2.3).

On retiendra par exemple des algorithmes intéressants pour l'optimisation dynamique [Gaspar and Collard, 1999]. Une description des fondements théoriques et de nombreuses applications des systèmes immunitaires artificiels peuvent être trouvées dans [De Castro and Von Zuben, 1999], [De Castro and Von Zuben, 2000] et dans [Dasgupta and Attoh-Okine, 1997], mais aussi dans un livre de référence [Dasgupta, 1999].

Algorithme 2.3 Un exemple simple d'algorithme de type *système immunitaire artificiel.*

1. Engendrer un ensemble de solutions P, composé d'un ensemble de cellules mémoires P_M ajoutées à la population présente P_r : $P = P_M + P_r$;
2. Déterminer les n meilleures cellules P_n parmi la population P, en se fondant sur la mesure de l'affinité ;
3. Cloner les n individus pour former une population C. Le nombre de clones produits pour chaque cellule est fonction de l'affinité ;
4. Effectuer une hyper-mutation des clones, engendrer ainsi une population C^*. La mutation est proportionnelle à l'affinité ;
5. Sélectionner les individus de C^* pour former la population mémoire P_M ;
6. Remplacer les plus mauvais individus dans P pour former P_r ;
7. Si un critère d'arrêt n'est pas atteint, retourner en 1.

2.3.4 Algorithmes inspirés des insectes sociaux non apparentés aux algorithmes de colonies de fourmis

Il existe des métaheuristiques inspirées du comportement des insectes sociaux qui ne sont pas explicitement liées aux — plus connus — algorithmes de colonies de fourmis. En effet, les comportements de ces espèces sont complexes et riches ; et les caractéristiques auto-organisées que présentent ces groupes d'insectes sont une source d'inspiration intéressante.

Un bon exemple est celui d'un algorithme inspiré des modèles d'organisation du travail chez les fourmis [Campos et al., 2000, Cicirello and Smith, 2001, Nouyan, 2002]. Le partage des tâches chez certaines espèces fait apparaître des individus qui accomplissent des tâches spécifiques, ce qui permet d'éviter les coûts (en temps et en énergie par exemple) liés aux ré-attributions de tâches. Cependant, la spécialisation des individus n'est pas rigide, ce qui pourrait être préjudiciable à la colonie, mais elle s'adapte en fonction des nombreux stimulus internes et externes perçus par les individus [Wilson, 1984].

Des modèles de comportement ont été proposés pour expliquer ce phénomène [Bonabeau et al., 1996, Bonabeau et al., 1998, Theraulaz et al., 1998]. Ces modèles mettent en jeu, pour chaque type de tâche, des seuils de réponse représentant le niveau de spécialisation de l'individu. Ces seuils sont soit fixés [Bonabeau et al., 1998], soit mis à jour en fonction de l'accomplissement des tâches par les individus [Theraulaz et al., 1998].

Ces modèles ont inspiré des algorithmes pour l'allocation dynamique de tâches, où chaque machine se voit associée à un individu disposant d'un jeu de seuils de réponse Θ_a, où $\Theta_{a,j}$ représente le seuil de l'agent a pour la tâche j. La tâche j envoie aux agents un stimulus S_j représentant le temps d'attente de la tâche. L'agent a aura une probabilité

d'effectuer la tâche j de :

$$P\left(\Theta_{a,j}, S_j\right) = \frac{S_j^2}{S_j^2 + \Theta_{a,j}^2}$$

L'algorithme dispose ensuite de règles de mise à jour des seuils et de règles de décision, au cas où deux agents tenteraient d'effectuer la même tâche (voir [Cicirello and Smith, 2001] pour plus de détails). Des améliorations à cet algorithme de base [Nouyan, 2002] ont permis d'augmenter sa rapidité et son efficacité sur des problèmes d'allocation dynamique de tâches.

Ici, les rétroactions positives sont liées à la spécialisation des individus, alors que les rétroactions négatives sont mises en place sous la forme des changements de tâches. La mémoire est locale, au niveau des seuils des différents individus.

2.3.5 Conclusion

Les deux théories présentées permettent de mieux comprendre le fonctionnement des métaheuristiques existantes et d'orienter la conception de nouvelles métaheuristiques. Les concepts importants à retenir sont l'utilisation par les métaheuristiques modernes de la mémoire, de l'intensification et de la diversification. L'aspect distribué et les qualités de flexibilité de ces algorithmes sont également des caractéristiques notables. Cependant il faut souligner la difficulté de conception d'un système auto-organisé, ce qui explique que l'inspiration vienne de la biologie, où de tels systèmes sont relativement courants.

Les difficultés principales sont les suivantes :

— concevoir une mémoire échantillonnant correctement le problème et dont il est aisé d'extraire l'information pertinente pour orienter la recherche,
— équilibrer la balance entre des techniques d'intensification et de diversification,
— maintenir la flexibilité de l'algorithme, de façon à ce qu'il s'adapte au problème, ou qu'il résolve des problèmes dynamiques,

Les perspectives ouvertes par les points de vue des théories de la programmation à mémoire adaptative et de l'auto-organisation permettront peut-être la conception de nouvelles métaheuristiques.

2.4 Échantillonnage de distribution

2.4.1 Introduction

Les métaheuristiques sont souvent classées selon deux ensembles : les algorithmes à base de solution courante unique et les méthodes à population. Dans le premier cas, la métaheuristique manipule un point, et décide à chaque itération quel sera le point suivant.

On classe par exemple la recherche avec tabous et le recuit simulé dans cet ensemble. Dans le second cas, la métaheuristique manipule une population de points, un nouvel ensemble de points est choisi à chaque itération. Beaucoup d'algorithmes peuvent entrer dans cet ensemble, comme les algorithmes évolutionnaires ou les algorithmes de colonies de fourmis.

Bien que ces classes soient perméables (un algorithme pouvant se trouver dans les deux classes, selon le point de vue où l'on se place), elles permettent de mettre en valeur une caractéristique importante : les métaheuristiques à population manipulent un ensemble de points, elles font évoluer un *échantillonnage* de la fonction objectif.

2.4.2 Fonction objectif et distribution de probabilité

Dans la majorité des cas, l'échantillonnage est probabiliste, les meilleures solutions ayant plus de chances d'être sélectionnées. En effet, dans le cas idéal, la fonction objectif représente la distribution de probabilité qu'il faudrait échantillonner : l'optimum devant présenter la plus grande probabilité d'être tiré.

Cependant, dans un problème d'optimisation, le but n'est pas d'échantillonner la fonction objectif, mais de trouver le mode (l'optimum) de cette distribution. Ainsi, l'échantillonnage doit se concentrer sur les régions d'intérêt, en convergeant progressivement vers l'optimum. Du point de vue de l'échantillonnage, cette convergence s'effectue par une baisse progressive de la dispersion dans ces zones.

2.4.3 Échantillonnage et *PMA*

Ici encore, on peut rapprocher ce point de vue de la programmation à mémoire adaptative : l'échantillonnage de la fonction objectif est appelé diversification et la baisse de dispersion, intensification. La mémoire est présente dans le processus "d'apprentissage" des paramètres de la distribution manipulée.

Le point crucial dans cette perspective est en fait la description de la distribution de probabilité à utiliser. La plupart des algorithmes à population actuels n'ont aucun a priori sur la distribution et utilisent des techniques empiriques pour effectuer plus ou moins simultanément diversification et intensification. Cependant, certaines méthodes postulent que la fonction objectif peut être raisonnablement approchée par une distribution donnée, et vont donc utiliser cette distribution pour effectuer le tirage aléatoire de la population (diversification), et la réduction de dispersion (intensification). On peut parler de méthodes implicites dans le premier cas, et explicites dans le second [Baluja, 1997].

Comme le suggère la PMA, les étapes de diversification permettent de rassembler de l'information sur le problème, ces informations sont mémorisées, puis triées par l'intensification, après quoi les informations en mémoire sont utilisées pour effectuer une

Algorithme 2.4 Méthode de Metropolis.

Initialiser un point de départ x_0 et une *température* T

Pour $i = 1$ à n :

Jusqu'à x_i accepté

Si $f(x_i) \leq f(x_{i-1})$: accepter x_i

Si $f(x_i) > f(x_{i-1})$: accepter x_i avec la probabilité $e^{-\frac{f(x_i)-f(x_{i-1})}{T}}$

Fin

Fin

nouvelle étape de diversification. Cette succession d'étapes apparaît clairement dès lors que l'on considère les métaheuristiques à population comme des algorithmes manipulant un échantillonnnage.

2.4.4 Exemples de métaheuristiques vues sous l'angle de l'échantillonnage de distribution

Nous verrons dans cette section comment quelques métaheuristiques emploient de fait l'échantillonnage de distribution pour résoudre un problème d'optimisation.

2.4.4.1 Recuit simulé

Le recuit simulé est fondé sur une analogie entre un processus physique (le recuit) et le problème de l'optimisation. Le recuit simulé en tant que métaheuristique [Kirkpatrick et al., 1983, Cerny, 1985] s'appuie en effet sur des travaux visant à simuler l'évolution d'un solide vers son état d'énergie minimale [Metropolis et al., 1953, Hastings, 1970].

La description classique du recuit simulé le présente comme un algorithme probabiliste, où un point évolue sur l'espace de recherche. En effet, la méthode repose sur l'algorithme de Metropolis, présenté sur la figure 2.4, qui décrit un processus Markovien [Aarts and Van Laarhoven, 1985, Krauth, 1998]. Le recuit simulé (dans sa version la plus courante, dite "homogène") appelle à chaque itération cette méthode.

Cependant, il est possible de voir le recuit simulé comme un algorithme à population [Collette, 2004]. En effet, l'algorithme de Metropolis est une méthode d'échantillonnage d'une distribution de probabilité : il échantillonne directement la fonction objectif par le biais d'une distribution de Bolzmann paramétrique (de paramètre T). Un des paramètres cruciaux est donc la décroissance de la température ; de nombreuses lois de décroissance différentes ont été proposées [Triki et al., 2004]. Il existe par ailleurs des

versions du recuit simulé qui mettent en avant la manipulation d'une population de points [Hukushima and Nemoto, 1996, Wendt and König, 1997, Liang and Wong, 2000, Iba, 2003].

Ici, la méthode de Metropolis (ou toute autre méthode d'échantillonnage [Creutz, 1983, Okamoto and Hansmann, 1995]) tient lieu de diversification, associée à la décroissance de température, qui contrôle l'intensification.

2.4.4.2 Algorithmes évolutionnaires

Les algorithmes évolutionnaires forment peut être la classe d'algorithmes à population la plus connue, ils ont été décrits plus haut.

Les algorithmes évolutionnaires classiques sont implicites. On peut cependant observer que les opérateurs de croisement et de mutation visent à produire un ensemble de nouveaux points (diversification) dans les limites de la population précédente (mémoire), points dont on va ensuite réduire le nombre (intensification), et ainsi de suite.

La difficulté à manipuler les opérateurs de diversification a entraîné la création de méthodes explicites (voir notamment [Syswerda, 1993], [Harik et al., 1998, Harik, 1999], [Baluja and Davies, 1998], qui démontrent plus clairement le rapport des algorithmes évolutionnaires avec la notion d'échantillonnage de distribution. La sous-section suivante décrit succinctement l'une de ces méthodes.

La méthode *PBIL* L'algorithme *PBIL* ("Population-Based Incremental Learning") est à l'origine inspiré de l'apprentissage compétitif et est conçu pour des problèmes binaires. Il fait l'objet d'une large littérature, on se référera notamment à [Baluja, 1994, Baluja and Caruana, 1995, Baluja, 1995], ainsi qu'à [Sebag and Ducoulombier, 1998] pour le domaine continu.

La méthode *PBIL*, décrite sur l'algorithme 2.5, utilise un coefficient d'apprentissage (noté ici α) qui contrôle l'amplitude des changements. La version continue utilise une distribution gaussienne à base d'un produit de densités univariantes et indépendantes, ainsi que plusieurs adaptations pour l'évolution du vecteur de variance.

L'algorithme enchaîne de façon itérative des étapes de diversification, de mémorisation et d'intensification sur un vecteur de probabilité (pour la version combinatoire) : nous sommes bien en présence d'un échantillonnage probabiliste.

2.4.4.3 Algorithmes à estimation de distribution

Les algorithmes à estimation de distribution ("Estimation of Distribution Algorithms", *EDA*) ont été conçus à l'origine comme une variante des algorithmes évolutionnaires [Mühlenbein and Paaβ, 1996]. Cependant, dans les méthodes de type *EDA*, il n'y a pas

Algorithme 2.5 La métaheuristique PBIL.

Initialiser un vecteur de probabilité $p_0(x)$

Jusqu'à critère d'arrêt :

Construire m individus $x_1^l, \ldots, x_m^l$ en utilisant $p_l(x)$

Évaluer et trier $x_1^l, \ldots, x_m^l$

Mémoriser les k meilleurs individus $x_{1:k}^l, \ldots, x_{m:k}^l$

Reconstruire le vecteur de probabilité $p_{l+1}(x) = (p_{l+1}(x_1), \ldots, p_{l+1}(x_k))$

Pour $i = 1$ à n :

$p_{l+1}(x_i) = (1-\alpha)\, p_l(x_i) + \alpha \frac{1}{k} \sum_{j=1}^{k} x_{i,j:k}^l$

Fin

Fin

d'opérateurs de croisement ou de mutation. En effet, la population des nouveaux individus est tirée au hasard, selon une distribution estimée depuis des informations issues de la population précédente. Dans les algorithmes évolutionnaires, la relation entre les différentes variables est implicite alors que, dans les algorithmes *EDA*, le coeur de la méthode consiste justement à estimer ces relations, à travers l'estimation de la distribution de probabilité associée à chaque individu sélectionné.

La meilleure manière de comprendre les *EDA* est d'étudier un exemple le plus simple possible. Ici, prenons comme problème la fonction cherchant à maximiser le nombre de 1 sur trois dimensions : on cherche donc à maximiser $h(x) = \sum_{i=1}^{3} x_i$ avec $x_i = \{0, 1\}$ (problème "OneMax").

La première étape consiste à engendrer la population initiale, on tire donc au hasard M individus, selon la distribution de probabilité : $p_0(x) = \prod_{i=1}^{3} p_0(x_i)$, où la probabilité que chaque élément x_i soit égal à 1 vaut $p_0(x_i)$. En d'autres termes, la distribution de probabilité, à partir de laquelle le tirage est fait, est factorisée comme un produit des trois distributions de probabilité marginales univariantes (ici, puisque les variables sont binaires, des distributions de Bernouilli de paramètre 0.5). La population ainsi échantillonnée est nommée D_0. Prenons pour l'exemple une population de six individus (illustrée figure 2.6).

i	x_1	x_2	x_3	$h(x)$
1	1	1	0	2
2	0	1	1	2
3	1	0	0	1
4	1	0	1	2
5	0	0	1	1
6	0	1	0	1

FIG. 2.6 – *L'algorithme EDA optimisant le problème OneMax : la population initiale* D_0.

La seconde étape consiste à sélectionner des individus parmi cette population ; on construit ainsi de façon probabiliste une deuxième population $D_0^{S_e}$, tirée parmi les meilleurs individus de D_0. La méthode de sélection est libre. Ici, on peut, pour l'exemple, sélectionner les trois meilleurs individus (figure 2.7).

i	x_1	x_2	x_3	$h(x)$
1	1	1	0	2
2	0	1	1	2
4	1	0	1	2
$p(x)$	0.3	0.3	0.3	

FIG. 2.7 – *L'algorithme EDA optimisant le problème OneMax : les individus sélectionnés* $D_0^{S_e}$.

La troisième étape consiste à estimer les paramètres de la distribution de probabilité représentée par ces individus sélectionnés. Dans cet exemple, on considère que les variables sont indépendantes ; trois paramètres permettent donc de caractériser la distribution. On va donc estimer chaque paramètre $p(x_i \mid D_0^{S_e})$ par sa fréquence relative dans $D_0^{S_e}$. On a donc : $p_1(x) = p_1(x_1, x_2, x_3) = \prod_{i=1}^{3} p(x_i \mid D_0^{S_e})$. En échantillonnant cette distribution de probabilité $p_1(x)$, on peut obtenir une nouvelle population D_1 (figure 2.8).

i	x_1	x_2	x_3	$h(x)$
1	1	1	1	3
2	0	1	1	2
3	1	0	0	1
4	1	0	1	2
5	0	1	1	2
6	1	1	0	2
$p(x)$	0.3	0.3	0.3	

FIG. 2.8 – *L'algorithme EDA optimisant le problème OneMax : la nouvelle population* D_1.

On continue ainsi jusqu'à ce que l'on atteigne un critère d'arrêt prédéterminé... L'algorithme général d'une méthode à estimation de distribution est présenté sur l'algorithme 2.6.

Algorithme 2.6 Algorithme à estimation de distribution.

$D_0 \leftarrow$ Engendrer M individus aléatoirement

$i = 0$

Tant que critère d'arrêt :

$i = i + 1$

$D_{i-1}^{S_e} \leftarrow$ Sélectionner $N \leq M$ individus dans D_{i-1} grâce à la méthode de sélection

$p_i(x) = p\left(x \mid D_{i-1}^{S_e}\right) \leftarrow$ Estimer la probabilité de distribution d'un individu d'être parmi les individus sélectionnés

$D_i \leftarrow$ Échantillonner M individus depuis $p_i(x)$

Fin

La principale difficulté, lorsqu'on utilise un algorithme à estimation de distribution, est d'estimer la distribution de probabilité. En pratique, il faut approcher les paramètres de la distribution, conformément à un modèle choisi. De nombreuses approximations ont été proposées pour des problèmes d'optimisation continue, aussi bien que combinatoire. On peut classer les différents algorithmes proposés selon la complexité du modèle utilisé pour évaluer les dépendances entre variables. On définit ainsi trois catégories :

1. Modèles *sans* dépendance : la distribution de probabilité est factorisée à partir de distributions indépendantes univariantes pour chaque dimension. Ce choix a le défaut d'être peu probable dans les cas de l'optimisation difficile, où une forte dépendance des variables est généralement la règle ;
2. Modèles à dépendances *bivariantes* : la distribution de probabilité est factorisée à partir de distributions bivariantes. Dans ce cas, l'apprentissage de la distribution peut être étendu jusqu'à la notion de *structure* ;
3. Modèles à dépendances *multiples* : la factorisation de la distribution de probabilité est obtenue à partir de statistiques d'ordre *supérieur* à deux.

On peut noter que, dans le cas de problèmes continus, le modèle de distribution est souvent fondé sur une base de distribution normale.

Quelques variantes d'importance ont été proposées : l'utilisation de "data clustering" pour l'optimisation multimodale et des variantes parallèles pour des problèmes combinatoires. Des théorèmes de convergence ont également été formulés, notamment avec l'aide de modélisations par chaînes de Markov ou par systèmes dynamiques.

Il serait fastidieux d'énumérer ici les algorithmes proposés dans chaque cas, on se reportera pour cela au très complet livre de référence dans le domaine [Larrañaga and Lozano, 2002].

Dans la grande majorité des EDA, l'accent est porté sur le fait que la diversification utilise des distributions de probabilité explicites, l'intensification n'étant, quant à elle, effectuée que par un opérateur de sélection.

2.4.4.4 Algorithmes de colonies de fourmis

Du point de vue de l'estimation de distribution, le point clef concerne le choix des composants de la solution. En effet, du fait des pistes de phéromone, chaque fourmi choisit de façon *probabiliste* un composant parmi d'autres. Ainsi, chaque solution est associée à une certaine probabilité de choix, probabilité résultant de l'ensemble des probabilités des composants, mises à jour de façon constructive.

Très récemment, des approches facilement comparables à des problèmes d'échantillonnage ont été employées dans la conception d'une métaheuristique du type colonies de fourmis pour des problèmes continus [Socha, 2004, Pourtakdoust and Nobahari, 2004]. Dans les deux cas, le principe de l'algorithme est comparable, il a été précédemment présenté sur l'algorithme 1.3 de la section 1.3.5.1. Dans ces méthodes, la distribution de probabilité mémorisée sous la forme de "phéromones" τ est celle qui est échantillonnée et dont l'évolution va guider la recherche. Dans les deux cas, il s'agit en théorie d'une somme de distributions normales univariées.

Si l'on observe les algorithmes de colonies de fourmis avec une approche "haut niveau", il y a bien échantillonnage et manipulation d'une distribution de probabilité [Dréo and Siarry, 2004a], qui va tendre à évoluer vers un optimum. Dans les modèles continus, la diversification est faite par échantillonnage d'une distribution à base normale, et l'intensification se fait par sélection des meilleurs individus, appelée ici "évaporation".

2.4.4.5 Cadres généraux

Suite aux origines indépendantes de métaheuristiques se révélant finalement très proches, plusieurs tentatives de structuration dans le sens de l'échantillonnage de distribution ont vu le jour.

Monmarché *et al.* proposent par exemple un modèle appelé *PSM* ("Probabilistic Search Metaheuristic" [Monmarché et al., 1999, Monmarché et al., 2000a]), en se fondant sur la comparaison des algorithmes *PBIL* ([Baluja, 1994, Baluja and Caruana, 1995], décrits dans le paragraphe 2.4.4.2), *BSC* [Syswerda, 1993] et du *Ant System* ([Colorni et al., 1992], décrit dans la section 1.3.2.1). Le principe général d'une méthode *PSM* est présenté sur l'algorithme 2.7. On peut constater la parenté de cette approche avec les algorithmes à estimation de distribution, mais l'approche *PSM* se restreint à l'utilisation de vecteurs de probabilité, en précisant toutefois que la règle de mise à jour de ces vecteurs est cruciale.

Algorithme 2.7 Le cadre de la méthode *PSM*.

Initialiser un vecteur de probabilité $p_0(x)$

Jusqu'à critère d'arrêt :

Construire m individus $x_1^l, \ldots, x_m^l$ en utilisant $p_l(x)$

Évaluer $f\left(x_1^l\right), \ldots, f\left(x_m^l\right)$

Reconstruire le vecteur de probabilité $p_{l+1}(x)$ en tenant compte de $x_1^l, \ldots, x_m^l$ et de $f\left(x_1^l\right), \ldots, f\left(x_m^l\right)$

Fin

Algorithme 2.8 L'approche *IDEA*.

Initialiser une population P_0 de n points

Tant_que critère d'arrêt non atteint :

Mémoriser le plus mauvais point θ

Chercher une distribution $D_i(X)$ adéquate à partir de la population P_{i-1}

Construire une population O_i de m points selon $D_i(X)$, avec $\forall O_i^j \in O_i : f\left(O_i^j\right) < f(\theta)$

Créer une population P_i à partir d'une partie de P_{i-1} et d'une partie de O_i

Évaluer P_i

Fin

Les *EDA* ont été présentés dès le départ comme des algorithmes évolutionnaires où la diversification serait explicite [Mühlenbein and Paaβ, 1996]. Ce sont sans doute les algorithmes les plus proches d'un cadre général. Une généralisation de ces méthodes aux cas continus et discrets a été proposée sous le terme de *IDEA* ("Iterated Density Evolutionary Algorithms", [Bosman and Thierens, 1999, Bosman and Thierens, 2000a] [Bosman and Thierens, 2000b]), elle se dote notamment de bases mathématiques, dont les auteurs de *PSM* regrettaient l'absence dans ce genre d'approche [Monmarché et al., 1999]. Le principe de l'approche *IDEA* est présenté sur l'algorithme 2.8.

IDEA utilise une diversification plus générale que *PSM*, en ne se limitant pas à un vecteur de probabilité comme modèle, mais en précisant que la recherche de la meilleure distribution de probabilité fait partie intégrante de l'algorithme. Cependant, la baisse de dispersion s'effectue en sélectionnant les meilleurs individus, aucune précision sur l'utilisation de principes d'intensification différents n'est donnée.

2.4.5 Conclusion

Les métaheuristiques à population peuvent être vues comme des algorithmes manipulant un échantillonnage de la fonction objectif via des techniques de diversification et d'intensification.

On peut les répartir en trois catégories de ce point de vue :

— utilisation d'une description implicite de la distribution utilisée pour effectuer l'échantillonnage (méthodes évolutionnaires classiques, algorithmes de colonies de fourmis, plus généralement métaheuristiques à population dites "classiques"),

— utilisation d'une description explicite (méthodes évolutionnaires explicites et plus généralement algorithmes à estimation de distribution),

— utilisation directe de la fonction objectif (recuit simulé).

Les méthodes implicites ont l'avantage de pouvoir s'affranchir du choix d'une description de l'échantillonnage à utiliser, mais aussi le défaut d'être difficiles à manipuler et à comprendre. Les méthodes explicites permettent de maîtriser complètement les processus de diversification et d'intensification de façon indépendante, mais sont liées au choix d'une distribution de probabilité. Les méthodes directes permettent d'utiliser la distribution de probabilité idéale (la fonction objectif) mais rendent l'intensification délicate, car indépendante de la structure du problème.

Le chapitre suivant présente les métaheuristiques que nous avons conçues en nous appuyant sur les notions que nous venons de présenter.

Chapitre 3

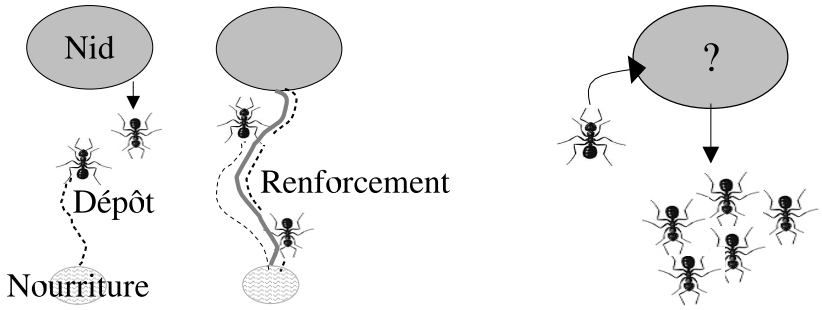
Élaboration d'algorithmes de colonies de fourmis en variables continues

3.1 Élaboration d'un algorithme exploitant la communication directe entre individus (*CIAC*)

3.1.1 Introduction

Les algorithmes de colonies de fourmis utilisent généralement un trait particulier du comportement des fourmis réelles : le dépôt de piste. En effet, les colonies de fourmis sont souvent vues comme des systèmes distribués capables de résoudre des problèmes par le biais de la stigmergie, une forme de communication indirecte passant par la modification de l'environnement, dont le comportement de dépôt de piste se veut l'archétype. Mais ce comportement fait également partie du processus plus général de "recrutement", défini par les biologistes comme "une forme de rassemblement dans lequel les membres d'une société sont dirigés vers un endroit où un travail est requis" [Hölldobler and Wilson, 1990, p. 642].

De notre point de vue, la métaphore des colonies de fourmis peut être définie comme un système utilisant la stigmergie, et plus généralement un processus de recrutement. Conformément à cette idée, le dépôt de piste ne serait pas la seule façon d'appréhender l'utilisation des algorithmes de type "colonies de fourmis" pour l'optimisation. Notre stage de recherche portant sur la modélisation de certains comportements des fourmis [Dréo, 2001] a montré qu'il était possible d'enclencher une séquence de recrutement, sans prendre en compte les pistes de phéromone. Dans ce modèle, les processus stigmergiques sont délibérément ignorés, au profit des relations inter-individuelles. Le modèle tente ainsi



(a) Par piste (b) Mobilisation

FIG. 3.1 – *Deux types de recrutement.*

de reproduire le flux de fourmis sortant du nid, après l'entrée d'une éclaireuse ayant découvert une nouvelle source de nourriture. Afin de différencier ce processus du recrutement "stigmergique", nous l'appelons *mobilisation* (Fig. 3.1b). La mobilisation est aussi une forme de recrutement : une éclaireuse rentre au nid après avoir localisé une source de nourriture et provoque ainsi la sortie de plusieurs fourmis. Nous avons montré que, grâce à une modélisation simple fondée sur le comportement de trophallaxie (échange de liquide nourricier entre deux individus), une colonie de fourmis pouvait résoudre certains problèmes, comme adapter l'amplitude et la vitesse du flux de sortie à l'état de l'éclaireuse. Ainsi, les principaux mécanismes mis en jeu dans la réponse du modèle sont la distribution des états des fourmis et le type de propagation de l'information de mobilisation à travers le système. Un tel modèle suggère que l'importance des mécanismes de communication inter-individuels pourrait être sous-estimée et qu'inclure ces mécanismes dans un algorithme de colonies de fourmis devrait améliorer les performances, en accélérant la diffusion de l'information.

Nous nous sommes intéressés à ces différentes façons d'appréhender la même idée et nous avons tenté de les réunir dans un même formalisme. Nous nous sommes donc tout naturellement intéressés à la notion d'hétérarchie dense (développée plus haut), qui est une description originale du mode de fonctionnement d'une colonie de fourmis, insistant sur la notion de *communication*. Nous proposons, à partir de cette idée, une formalisation simple permettant d'utiliser ces différentes notions, pour concevoir une méthode d'optimisation. Un *algorithme hétérarchique* met ainsi l'accent sur le flux d'informations passant à travers une population d'agents, informations qui sont échangées grâce à des *canaux de communication* et permettent à une certaine forme de description de la fonction objectif d'émerger du système. L'algorithme que nous avons conçu sur ce modèle est nommé *CIAC*, de l'anglais "Continuous Interacting Ant Colony" et fait l'objet de la

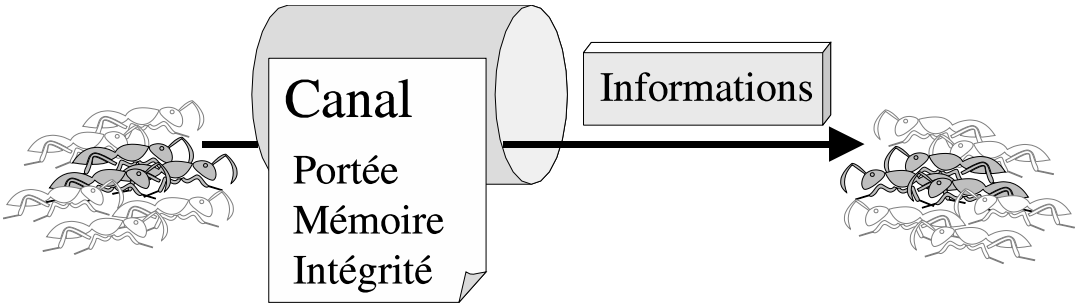


FIG. 3.2 – *L'information passe d'une partie de la population à une autre en empruntant un canal de communication ayant des propriétés spécifiques.*

présente section [Dréo and Siarry, 2001, Dréo and Siarry, 2002a, Dréo and Siarry, 2002c, Dréo and Siarry, 2002b, Dréo and Siarry, 2004c].

Cette section comprend six sous-sections. Dans la première, nous expliquons la notion d'algorithme hétérarchique. Puis nous introduisons les canaux de communication utilisés dans CIAC, et nous discutons du réglage de la méthode. Ensuite, nous présentons les résultats expérimentaux, et enfin nous concluons.

3.1.2 Algorithmes hétérarchiques

La notion d'hétérarchie dense, présentée plus haut, décrit le mode de gestion utilisé par les colonies de fourmis pour traiter les informations qu'elles reçoivent de leur environnement. Chaque fourmi pouvant communiquer avec n'importe quelle autre, à n'importe quel moment, l'information circule d'agent en agent à travers la colonie.

Nous proposons un formalisme simple pour appliquer cette notion à des problèmes d'optimisation, le principal concept d'un algorithme hétérarchique étant le canal de communication. Un canal de communication, pour prendre l'exemple des pistes dans la métaphore des colonies de fourmis, transmet une information (la localisation d'une source d'information) et présente des propriétés spécifiques (stigmergie, mémoire) (cf. Fig. 3.2).

Les principales propriétés des canaux de communication sont :

Portée : la façon dont l'information circule à travers la population. Un sous-groupe (de un à n agents) peut échanger des informations avec un autre sous-groupe d'agents.

Mémoire : la façon dont l'information persiste dans le système. L'information peut être stockée pendant une certaine période ou n'être que temporaire.

Intégrité : la façon dont l'information évolue. L'information peut être modifiée par un ou plusieurs agents, par un processus externe, ou rester inchangée.

Chaque canal pouvant combiner d'une manière différente ces différentes propriétés, une grande variété de canaux différents est possible.

L'information transmise durant la communication peut prendre plusieurs formes, depuis une simple valeur, jusqu'à un "objet" complexe, il est donc difficile de décrire des classes particulières. Retenons les informations les plus intuitives dans le cas d'un problème d'optimisation : le vecteur de coordonnées d'un point, ou la valeur de la fonction objectif associée à cette solution.

Prenons par exemple les propriétés d'un canal de communication de type "dépôt de piste". La *portée* est potentiellement la population entière, puisque chaque fourmi peut percevoir une piste de phéromone. Il y a aussi présence d'une forme de *mémoire*, car c'est un processus stigmergique, la piste persiste donc dans l'environnement pendant une certaine période. Finalement, l'*intégrité* du canal permet que l'information soit altérée au cours du temps, avec l'évaporation des phéromones.

3.1.3 Les canaux de communication de *CIAC*

La conception de l'algorithme *CIAC* est relativement simple, si on le considère comme un algorithme hétérarchique. Nous avons implémenté trois versions, chacune étant définie par les types de canaux de communication qu'elle utilise.

3.1.3.1 Communication par pistes

La première version de *CIAC* tente d'être aussi proche que possible d'une version continue de l'algorithme *ACO* original [Colorni et al., 1992] qui fut conçu pour des problèmes combinatoires. En conséquence, elle partage de nombreux points communs avec l'algorithme *CACO* [Bilchev and Parmee, 1995], également inspiré par les premiers algorithmes de colonies de fourmis.

Cette première implémentation n'utilise qu'un seul canal de communication, tiré du comportement de dépôt de piste des fourmis. Ici, chaque fourmi peut déposer une certaine quantité de phéromone sous la forme d'un "spot" sur l'espace de recherche, proportionnellement à l'amélioration de la fonction objectif qu'elle a trouvée en se déplaçant. Ces spots de phéromone peuvent être perçus par tous les membres de la population, et diffusent dans l'environnement. Les fourmis sont dès lors attirées par chaque spot en fonction de la distance fourmi-spot et de la quantité de phéromone contenue dans ce dernier. Plus précisément, les agents se déplacent vers le centre de gravité G_j du nuage de spots de phéromone.

La position de ce centre de gravité dépend de l'intérêt ω_{ij} de la $j^{\text{ème}}$ fourmi pour le $i^{\text{ème}}$spot.

$$G_j = \sum_{i=1}^{n} \left(\frac{x_i \cdot \omega_{ij}}{\sum_{i=1}^{n} (\omega_{ij})} \right) \quad \text{avec} \quad \omega_{ij} = \frac{\overline{\delta}}{2} \cdot e^{(-\theta_i \cdot \delta_{ij})} \tag{3.1}$$

Ici, n est le nombre de spots et x_i la position du $i^{\text{ème}}$ spot. Les variables mises en jeu dans le calcul de l'intérêt ω_{ij} sont : δ la distance moyenne entre deux agents dans la population, θ_i la quantité de phéromone déposée sur le spot et δ_{ij} la distance entre la $j^{\text{ème}}$fourmi et le $i^{\text{ème}}$ spot. Il est important de noter que chaque fourmi ne se déplace pas directement sur le centre de gravité. En effet, chaque fourmi a un paramètre de "portée" (noté ϕ_j), distribué selon une loi normale sur la population. Chaque fourmi tire aléatoirement une distance, dans la limite de son paramètre de portée, et se déplace de cette longueur dans la direction du centre de gravité pondéré qu'elle perçoit, un bruit aléatoire modifiant quelque peu la position finale.

Pour résumer ce comportement du point de vue du concept d'hétérarchie, ce canal de communication "par piste" possède les propriétés suivantes :

1. *Portée* : un spot de phéromone déposé par une fourmi peut être perçu par toutes les autres fourmis,
2. *Mémoire* : l'information persiste dans le système, indépendamment des agents,
3. *Intégrité* : l'information est modifiée au cours du temps, sous la forme d'une évaporation des phéromones.

On peut constater certaines similarités avec l'algorithme de "path-relinking" présenté par Glover [Glover and Laguna, 1997], puisque les fourmis se déplacent à travers un jeu de points d'intérêt.

3.1.3.2 Le canal inter-individuel

Comme nous l'avons précisé en introduction, certains travaux de biologie nous ont poussés à considérer d'autres voies que la voie "stigmergique" pour l'optimisation par une colonie de fourmis. Nous avons donc implémenté un autre canal de communication, possédant des propriétés d'interactions directes entre individus, qui peuvent être observées dans les sociétés de certaines espèces d'insectes, comme les fourmis.

Concrètement, dans cet algorithme, chaque fourmi artificielle peut envoyer des "messages" à une autre ; une fourmi recevant un message le stocke dans une pile avec les autres messages reçus. Dans un second temps, un message est aléatoirement lu dans la pile. Notons que ce processus est proche de certains travaux sur les problèmes de communication dans de grands systèmes multi-agents [Hewitt, 1977], plus particulièrement concernant l'implémentation de programmes parallèles. Ici, l'information envoyée est la position de

l'expéditeur — une fourmi, représentant une solution au problème — et la valeur de la fonction objectif associée. Le receveur compare la valeur de l'expéditeur avec la sienne et décide s'il doit se déplacer vers lui (si la valeur de l'expéditeur est meilleure) ou non. Dans le premier cas, la position finale est tirée aléatoirement dans une hyper-sphère ayant l'expéditeur pour centre et la portée du receveur pour rayon. Dans le second cas, le receveur envoie un message à une autre fourmi choisie aléatoirement, et supprime le message lu. On peut noter que le système nécessite une activation, le nombre de messages initialisés au lancement de l'algorithme est donc un paramètre important.

Ce canal de communication a les propriétés suivantes :

1. *Portée* : un message envoyé par une fourmi ne peut être perçu que par une seule autre fourmi,
2. *Mémoire* : l'information persiste dans le système, sous la forme des mémoires individuelles des fourmis,
3. *Intégrité* : les informations stockées sont statiques.

3.1.3.3 Algorithme final

Les deux algorithmes décrits précédemment sont conçus pour être complémentaires. Nous avons donc implémenté dans un même système ces deux canaux de communication, afin d'étudier comment ils fonctionnent ensemble. Cette combinaison est simple à effectuer, les canaux de communication n'ayant pas de processus concurrents.

3.1.4 L'algorithme *CIAC*

Cet algorithme comprend trois étapes principales (cf. figure 3.3). Dans la première, les paramètres sont initialisés, les portées des fourmis sont distribuées sur la population et les fourmis sont disposées aléatoirement sur l'espace de recherche. L'algorithme démarre alors, et les fourmis se déplacent, jusqu'à ce que le critère d'arrêt soit atteint : la différence entre deux meilleurs points consécutifs est inférieure à 10^{-3} pendant $(\eta \cdot 10)$ évaluations (voir plus bas, pour une définition de η) ou un nombre maximum d'évaluations de la fonction objectif a été atteint.

Quatre paramètres doivent être initialisés :

1. $\eta \in [0, +\infty[$: le nombre de fourmis dans le système,
2. $\sigma \in [0, 1]$: un pourcentage de la taille de l'espace de recherche ; utilisé pour définir l'écart-type de la distribution normale des portées des fourmis,
3. $\rho \in [0, 1]$: définit la persistance des spots de phéromone,
4. $\mu \in [0, +\infty[$: nombre de messages initial.

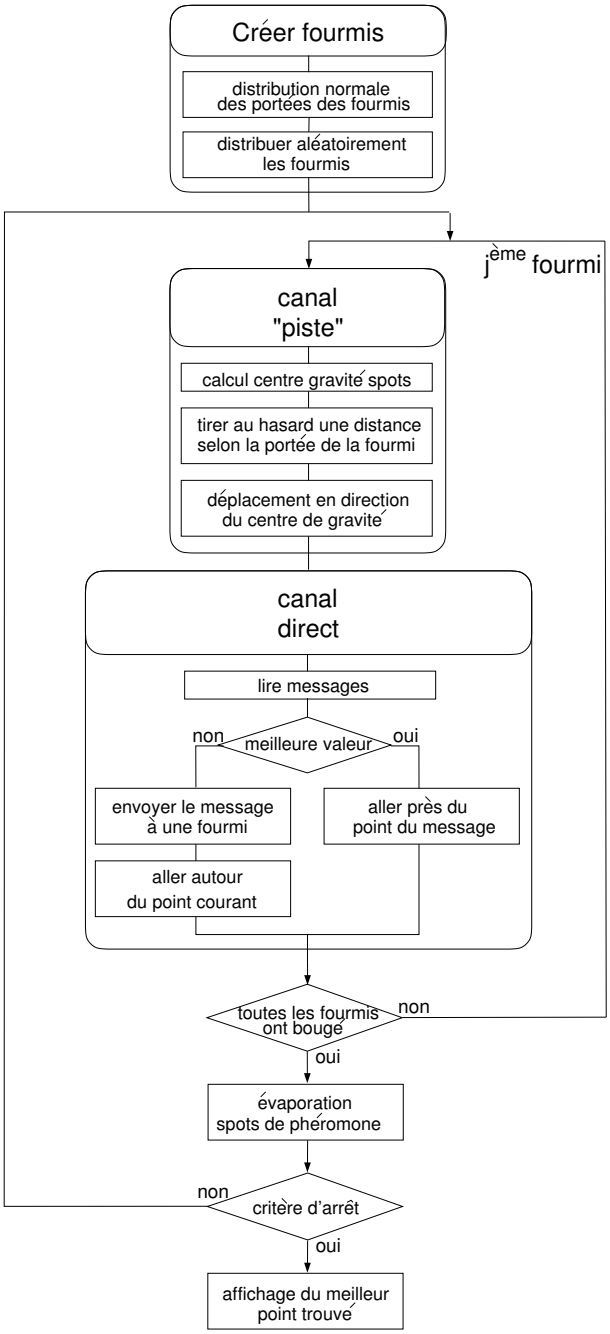


FIG. 3.3 – *Structure de l'algorithme CIAC.*

L'initialisation de ces quatre paramètres est discutée dans la section 3.1.5.

Quelques paramètres supplémentaires sont inclus dans l'algorithme, ou déduits des paramètres choisis :

1. les points de départ des fourmis sont répartis aléatoirement sur l'espace de recherche, selon une distribution uniforme,
2. ε : sous cette valeur seuil, un spot de phéromone disparaît ; il est généralement initialisé à la plus petite valeur possible (10^{-14} dans nos essais),
3. ς : l'amplitude du bruit ajouté à la position de la fourmi dans le canal "piste" ; c'est un pourcentage de la portée de la fourmi, initialisé conformément à l'équation 3.2 (voir la section 3.1.3.1 pour les variables utilisées) :

$$\begin{cases} \frac{\overline{\delta}}{\phi_j} & \text{si} \quad \overline{\delta} \leq \phi_j \\ 1 & \text{sinon} \end{cases} \tag{3.2}$$

3.1.5 Réglage de *CIAC*

Le nombre de fourmis η n'est pas un paramètre critique, il n'influence pas de façon critique la convergence globale de l'algorithme. Avec seulement 10 fourmis, l'algorithme optimise correctement des problèmes à deux dimensions, mais le nombre de fourmis doit être augmenté avec le nombre de dimensions. Un choix de 100 fourmis est un bon compromis, pour des problèmes allant de 2 à 100 dimensions.

Le pourcentage de la taille de l'espace de recherche σ influence quant à lui l'efficacité de l'algorithme. En effet, il définit la façon dont l'espace de recherche sera exploré : une valeur trop petite entraînera une diversification insuffisante, alors qu'une trop grande valeur tendra à provoquer des mouvements aléatoires. L'expérience montre qu'une valeur relativement élevée (typiquement 0.9) est recommandée.

Les deux paramètres suivants sont intéressants, car ils concernent les deux canaux de communication implémentés. Le premier, la persistance des spots de phéromone ρ, est assez sensible ; une valeur élevée peut aisément piéger les fourmis dans des minima locaux. Ici aussi, l'expérience suggère l'utilisation d'une faible valeur (typiquement 0.1) pour éviter ce problème sur la plus grande partie des fonctions de test. Le second paramètre est le nombre initial de messages μ, qui est spécifique au canal de communication direct. Il est relativement critique et nécessite d'être accordé avec le nombre de fourmis, car il définit des échanges d'individus à individus.

Après plusieurs essais empiriques sur le jeu de 13 fonctions tests, nous avons mis en place une procédure pour régler automatiquement les paramètres afin que seuls deux d'entre eux aient besoin d'être réglés manuellement. Les valeurs par défaut de ces deux

paramètres sont $\rho = 0.1$ et $\sigma = 0.9$. Ces réglages représentent un compromis sur le jeu de test utilisé et ne sont donc pas les meilleurs pour une fonction particulière. De plus, nous avons décidé de mettre l'accent sur l'efficacité de l'algorithme (sa capacité à trouver une valeur approximative de l'optimum global), plutôt que sur la rapidité (le nombre d'évaluations nécessaires de la fonction objectif) ou la précision (la distance finale à l'optimum global). Les règles du réglage automatique sont principalement dépendantes du nombre de dimensions du problème, celui-ci étant en effet souvent le plus gros générateur de complexité.

Le nombre de fourmis est initialisé selon la relation 3.3 :

$$\eta = \eta_{\max} \left(1 - e^{\left(-\frac{d}{p}\right)} \right) + \eta_0 \tag{3.3}$$

en désignant par :

— d le nombre de dimensions de la fonction objectif,
— $\eta_{\max} = 1000$ le nombre maximum de fourmis,
— $\eta_0 = 5$ le nombre minimal d'agents,
— $p = 10$ l'importance relative du nombre de fourmis,
— $\mu = (\eta \cdot 2/3)$ le nombre initial de messages.

Ce procédé permet d'ajuster la taille de la population au problème.

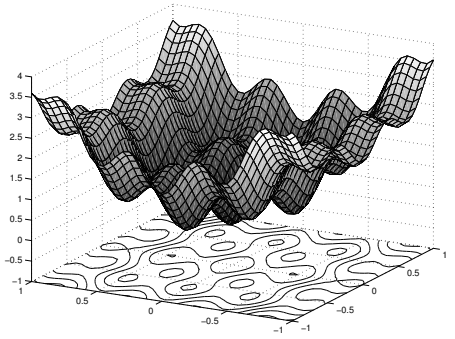
En résumé, les deux paramètres sensibles sont ceux spécifiquement mis en jeu dans les canaux de communication, et l'algorithme est généralement efficace quand il existe un compromis entre ρ et μ. Le réglage automatique proposé permet un tel équilibre.

3.1.6 Résultats expérimentaux et discussion

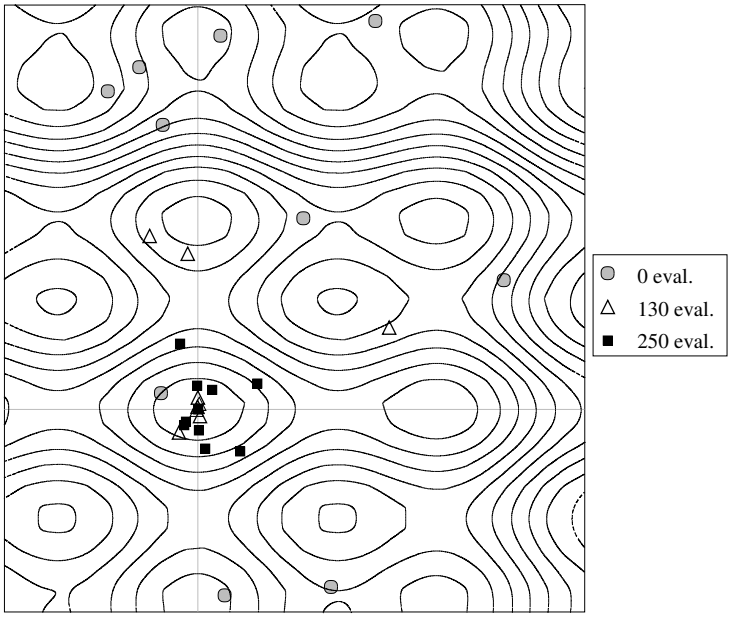
3.1.6.1 Comportement émergent

Afin d'illustrer le comportement de *CIAC*, nous l'avons testé sur la fonction B_2 (voir détails de ce problème dans l'annexe A.2.2).

Les deux canaux jouent des rôles complémentaires. En effet, le canal direct tend à induire une intensification, car il donne plus d'importance aux meilleurs points, sans prendre en compte les régions précédemment rencontrées. Au contraire, le canal "piste" — avec ses propriétés de mémoire — permet une diversification, en prenant en compte les points déjà évalués. La figure 3.4 montre comment l'algorithme procède avec les deux canaux. De façon générale, les fourmis se rassemblent autour de l'optimum global. Cependant, certaines sont attirées pendant quelques itérations dans des optima locaux par des spots résiduels, mais l'évaporation et le canal de communication direct les empêchent de rester piégées dans ces régions.



(a) La fonction B2 près de l'optimum global



(b) Optimisation de B2 par CIAC

FIG. 3.4 – CIAC *optimisant la fonction B_2 avec deux canaux de communication, trois étapes sont montrées : à la première itération, après 130 et 250 évaluations de la fonction objectif.*

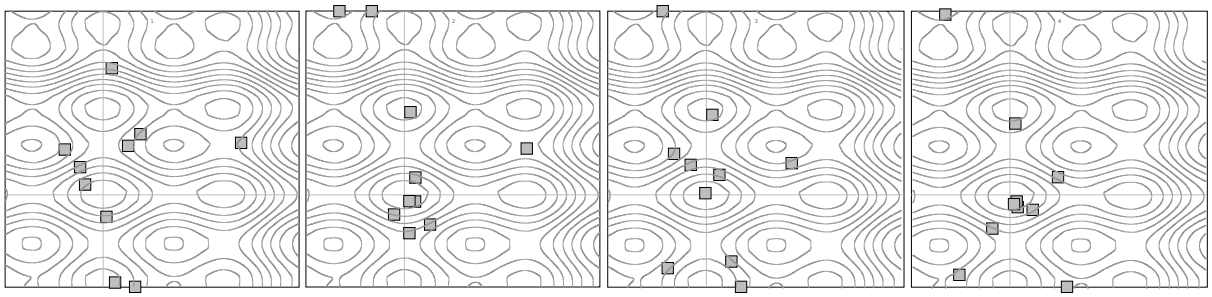


FIG. 3.5 – *CIAC optimisant la fonction B_2 uniquement avec le canal "piste", quatre étapes consécutives sont montrées (après une centaine d'évaluations de la fonction objectif).*

La mise en place d'une diversification par le canal "piste" est plus difficile à cerner. La figure 3.5 montre *CIAC* n'utilisant que le canal "piste", ces quatre étapes consécutives sont prises après 101, 102, 103 et 104 évaluations de la fonction objectif. Ici, les fourmis se déplacent autour d'un centre de gravité formé par l'optimum global, mais continuent à explorer l'espace de recherche, attirées par des minima locaux.

Un aspect intéressant de l'algorithme est la façon dont les deux canaux fonctionnent en *synergie*. En effet, sur la fonction B_2, le canal direct paraît le plus approprié, car il permet à *CIAC* de converger plus rapidement (figure 3.6a). Mais comme nous allons le voir dans la section 3.1.6.2, *CIAC* est alors piégé dans un minimum local. En observant la variation de l'écart-type de l'algorithme utilisant les deux canaux de communication (figure 3.6b), on peut constater qu'il existe des *oscillations*. Ce comportement signifie que la population tend à s'agréger autour d'une valeur à un temps donné, puis elle tend à se disperser. En d'autres termes, il y a alternance entre des courtes phases d'intensification (faible écart-type) et de diversification (écart-type élevé).

Ce comportement de l'algorithme *CIAC* ne peut être observé que quand les deux canaux sont utilisés ensemble (comme le montre la figure 3.7). En effet, avec un seul canal, l'écart-type diminue et l'amplitude des oscillations est plus faible. L'écart-type moyen (un indice de la dispersion) vaut pour les deux canaux $\sigma_{deux} = 1581$, pour le canal "piste" seul, $\sigma_{piste} = 1100$ et pour le canal direct seul, $\sigma_{direct} = 911$.

CIAC régule donc par lui-même la manière dont les canaux fonctionnent, jusqu'à ce qu'un état stable soit trouvé. Nous appelons ce comportement un modèle émergent, car il n'y a pas de contrôle centralisé du système, les fourmis interagissant localement sans accès à une information globale sur le système. Autrement dit, l'apparition d'un modèle global (ici, le comportement spécifique du système, lors de l'emploi des deux canaux) n'est pas planifié à l'avance.

3.1.6.2 Résultats

Nous avons d'abord testé l'algorithme *CIAC* sur un jeu de fonctions analytiques extrait de la littérature. Puis, dans l'optique de comparer *CIAC* avec d'autres méthodes d'optimisation continue (voir annexe A.1 pour la liste des algorithmes), nous avons choisi 13 fonctions de test en partie présentes dans la littérature sur les algorithmes de colonies de fourmis [Bilchev and Parmee, 1995, Mathur et al., 2000, Monmarché et al., 2000b], et en partie dans des articles sur d'autres méthodes d'optimisation continue [Storn and Price, 1995, Chelouah and Siarry, 2000a, Chelouah and Siarry, 2000b]. Ces problèmes de test sont énumérés dans l'annexe A.2, il s'agit indifféremment de problèmes de minimisation ou de maximisation.

Pour éviter certains problèmes dus au choix de la population initiale, nous avons effectué chaque test 100 fois, avec une initialisation du générateur de nombres pseudo-aléatoires différente à chaque essai. Les nombres d'évaluations de la fonction objectif (notés "evals") et l'erreur moyenne (notée "err") sont obtenus par un moyennage des résultats sur tous les tests (les écarts-type sont donnés entre parenthèses). Un test est considéré comme réussi (et donc comptabilisé dans le pourcentage de réussite, noté "% ok") si l'inégalité 3.4 est vérifiée :

$$|f^{\alpha} - f^{*}| < \epsilon_1 \cdot f^{*} + \epsilon_2 \tag{3.4}$$

en désignant par :

- f^{*} l'optimum global de la fonction objectif,
- f^{α} l'optimum trouvé par l'algorithme,
- ϵ_1 et ϵ_2 des paramètres de précision : dans chaque test de ce chapitre, $\epsilon_1 = \epsilon_2 = 10^{-4}$.

Nous avons tout d'abord testé les différentes versions de *CIAC*, avec un seul des deux canaux ou les deux canaux conjugués. Les résultats pour les trois variantes sont rassemblés dans le tableau 3.1.

On peut noter, en terme d'efficacité, que l'algorithme n'implémentant que le canal direct ne peut pas trouver l'optimum global et est facilement piégé dans des optima locaux. Au contraire, le canal "piste" permet d'éviter ce problème et mène à une meilleure efficacité. La combinaison des deux canaux permet de trouver l'optimum global plus souvent que lors de l'emploi d'un seul canal, mais cette amélioration a un coût, en terme de nombre d'évaluations nécessaires.

Afin de comparer *CIAC* avec deux autres algorithmes de colonies de fourmis, nous avons effectué des tests avec les mêmes problèmes de test que ceux utilisés dans les articles correspondants [Monmarché et al., 2000b, Bilchev and Parmee, 1995]. Les résultats sont

TAB. 3.1 – *Comparaison des versions de* CIAC*, pour un ensemble de huit fonctions tests.*

	Direct			Piste			Deux		
Fonction	%ok	evals	err	%ok	evals	err	%ok	evals	err
SM	1	39735(104)	7.53(2.9)	100	39899(58)	1e-8(2e-8)	100	50050(76)	9e-10(1e-11)
B_2	11	5868(5964)	1.95(2.01)	76	5964(51)	2e-3(3e-3)	100	11827	2e-8(1e-7)
R_2	18	5794(50)	16.90(33.54)	88	5943(54)	4e-3(9e-3)	100	11797(129)	3e-3(3e-3)
R_5	1	19594(192)	295(375)	75	19808(103)	1e-2(2e-2)	90	39546(289)	8e-3(7e-3)
GP	4	11561(30)	30(32)	54	11777(103)	1.63(2.59)	56	23391(48)	1.51(2.33)
Gr_{10}	0	50000(29)	1.1(0.24)	5	50000(33)	0.62(0.14)	52	50121(49)	0.05(0.05)
MG	2	5826(45)	0.71(0.53)	12	5901(53)	0.72(0.74)	20	11751(61)	0.34(0.19)
$S_{4,5}$	1	19458(86)	-1.96(0.73)	48	19755(154)	3.38(2.74)	5	39311(294)	6.34(1.01)

TAB. 3.2 – *Résultats de* CIAC *et de deux autres algorithmes de colonies de fourmis pour le domaine continu.*

	CACO			API			CIAC		
Fonction	%ok	evals	err	%ok	evals	err	%ok	evals	err
R_2	100	6842	0.00		[10000]	0.00 (0.00)	100	11797	3e-3(3e-3)
SM	100	22050			[10000]	0.00 (0.00)	100	50000	9e-10 (1e-11)
Gr_5					[10000]	0.18 (0.04)	63	48402	0.01(9e-3)
Gr_{10}	100	50000	0.0				52	50121	0.05(0.05)
GP	100	5330					56	23391	1.51 (2.33)
MG	100	1688					20	11751	0.34 (0.19)
St		[6000]	0.0				94	28201	1.96(1.61)
B_{f_1}		[200000]	544				0	50000	99521 (463)
B_{f_2}		[200000]	877				0	50000	99635 (512)
B_{f_3}		[200000]	54231				0	50000	99895 (86)

présentés dans le tableau 3.2. Les valeurs données entre crochets sont obtenues pour un nombre fixé d'évaluations, sans qu'aucun critère d'arrêt supplémentaire ne soit donné. Les cellules vides correspondent à des données manquantes dans la littérature.

En raison du manque d'informations dans la littérature, la comparaison ne peut être complète, notamment concernant l'efficacité et la rapidité de *API*, non spécifiées dans l'article correspondant, ainsi que certaines informations pour l'algorithme *CACO*. *CIAC* est cependant moins bon que *CACO* dans la plupart des cas disponibles, mais semble aussi performant que *API*. On peut noter que le nombre d'évaluations de *CIAC* est parfois plus élevé que ceux de *CACO* ou d'*API*, probablement à cause d'une meilleure précision de *CIAC*, et de l'importance donnée à l'efficacité dans nos tests. Les fonctions de Baluja sont un cas particulier : en deux dimensions, leurs graphiques ne présentent aucun minimum

TAB. 3.3 – *Comparaison des résultats de* CIAC *et de trois algorithmes d'optimisation pour des problèmes continus.*

	CGA			ECTS			DE		CIAC		
Fonction	%ok	evals	err	%ok	evals	err	%ok	evals	%ok	evals	err
St							100	1300	94	28201	1.96(1.61)
Gr_{10}							100	12804	52	50121	0.05(0.05)
SM	100	750	0.0002	100	338	3e-8	100	392	100	50000	9e-10(1e-11)
R_2	100	960	0.004	100	480	0.02	100	615	100	11797	3e-3(3e-3)
R_5	100	3990	0.15	100	2142	0.08			90	39546	8e-3(7e-3)
GP	100	410	0.001	100	231	2e-3			56	23391	1.51(2.33)
$S_{4,5}$	76	610	0.14	75	825	0.01			5	39311	6.34(1.01)

local, mais un unique minimum global (voir figures A.2bcd de l'annexe A.3). Dans de tels cas, un algorithme de descente pourrait être plus efficace.

Nous avons finalement testé *CIAC* sur des problèmes identiques à ceux utilisés pour d'autres métaheuristiques (un algorithme génétique en variables continues : *CGA* [Chelouah and Siarry, 2000a], une recherche avec tabous continue *: ECTS* [Chelouah and Siarry, 2000b], une méthode évolutionnaire : *DE* [Storn and Price, 1995]). Le tableau 3.3 montre les résultats correspondants. Pour la méthode *DE*, les valeurs d'erreur sont inconnues.

Pour toutes les fonctions de test, *CIAC* est loin d'atteindre la rapidité des autres algorithmes et est plus mauvais en terme d'efficacité. Mais il atteint la plupart du temps des précisions comparables (cf. tableau 3.3). Ces résultats montrent le coût en temps inhérent aux algorithmes d'optimisation du type colonies de fourmis.

3.1.7 Conclusion

Nous avons montré que le concept d'hétérarchie pouvait être intéressant pour la conception d'algorithmes de colonies de fourmis, particulièrement pour l'optimisation de fonctions continues présentant de multiples optima locaux. Un tel concept biologique n'avait pas été exploité jusqu'à présent pour la conception d'algorithmes d'optimisation, plus centrés sur les processus stigmergiques. Nous avons proposé d'étendre la métaphore des colonies de fourmis afin de prendre en compte la diversité des systèmes de communication utilisés par les insectes sociaux. *CIAC* est un algorithme hétérarchique implémentant deux canaux de communication complémentaires. Il possède des propriétés intéressantes, comme une auto-adaptation de l'influence relative des deux canaux.

Lors de la comparaison de *CIAC* avec d'autres algorithmes concurrents, nous avons montré les défauts hérités du concept de colonie de fourmis. Le principal problème est

le coût en terme de nombre d'évaluations de la fonction objectif et la mauvaise efficacité relative. Mais *CIAC* hérite également des qualités d'un tel concept, comme l'auto-gestion de son comportement global et sa flexibilité.

Dans sa forme actuelle, *CIAC* peut être utilisé pour une recherche globale des régions prometteuses de l'espace de recherche, mais il devra être hybridé avec une recherche locale, pour être plus rapide et précis dans la localisation des meilleures solutions.

Le fait que les colonies de fourmis ne soient pas aussi efficaces que des métaheuristiques plus "classiques" sur des fonctions de test statiques classiques ne doit pas laisser penser que cette métaphore n'est pas intéressante pour l'optimisation de problèmes continus. En effet, en regardant de plus près les avantages et les inconvénients de la métaphore des colonies de fourmis dans la littérature, nous pouvons conjecturer que ces algorithmes ont des performances moyennes dans le cas statique, mais semblent plus adaptés à des problèmes dynamiques, du fait de leur nature distribuée et adaptative [Bonabeau et al., 1999].

3.2 Hybridation avec un algorithme de recherche locale (*HCIAC*)

3.2.1 Introduction

Comme nous l'avons vu, l'algorithme *CIAC* est — comme le "Ant System" original — moins compétitif dans le domaine de la recherche locale, mais efficace en recherche globale. Pour diminuer l'impact de ces défauts, nous avons choisi de l'hybrider avec l'algorithme du "simplexe" de Nelder-Mead [Nelder and Mead, 1965], la méthode hybride est appelée "*HCIAC*" pour "Hybrid Continuous Interacting Ant Colony" [Dréo and Siarry, 2003d, Dréo and Siarry, 2004d].

Cette section comprend cinq sous-sections supplémentaires. Tout d'abord, les algorithmes de base utilisés pour concevoir *HCIAC* sont présentés. La section suivante est dédiée à la description détaillée de *HCIAC*. Le réglage de la méthode est ensuite étudié. Puis l'on présente et discute les résultats expérimentaux. Enfin, nous concluons avec la dernière section.

3.2.2 Algorithmes de base constitutifs de *HCIAC*

3.2.2.1 Algorithme de Nelder-Mead

Beaucoup d'algorithmes exploitant une population sont peu efficaces pour trouver rapidement et précisément des optima locaux, bien qu'ils soient meilleurs pour localiser des régions prometteuses. Les algorithmes de colonies de fourmis sont ainsi plus efficaces lors-

qu'ils utilisent une recherche locale complémentaire. Cette technique est souvent utilisée dans le domaine discret, pour rendre ces algorithmes compétitifs [Bonabeau et al., 1999].

L'algorithme de Nelder-Mead [Nelder and Mead, 1965] est une méthode simple et efficace de recherche locale, qui a l'avantage de ne pas être dépendante de la dérivée. Il manipule en effet une petite population de points sous la forme d'un "simplexe" non dégénéré. Un simplexe est une figure géométrique avec un volume non nul de n dimensions, qui est une enveloppe convexe de $n+1$ dimensions. La méthode de Nelder-Mead conçiste à déformer ce simplexe de quatre manières différentes : réflexion (coefficient ρ), expansion (γ), contraction (χ) et raccourcissement (δ) (voir figure 3.8). Les coefficients suivent typiquement les conditions ci-dessous :

$$\rho > 0, \chi > 1, \chi > \rho, 0 < \gamma < 1, 0 < \delta < 1$$

et sont généralement fixés avec les valeurs :

$$\rho = 1, \chi = 2, \gamma = \frac{1}{2}, \delta = \frac{1}{2}$$

Les quatre types d'altérations sont conçus pour que le simplexe suive le gradient de la fonction objectif (voir algorithme 3.1).

3.2.3 Algorithme *HCIAC*

3.2.3.1 Améliorations de *CIAC*

Nous avons relevé deux problèmes dus à la conception de *CIAC* et, pour chacun d'entre eux, nous proposons une solution. La première idée consiste à utiliser moins de spots pour structurer l'espace de recherche, afin d'accélérer l'algorithme ; la seconde amélioration concerne l'utilisation de seuils de décision pour réguler le choix entre les deux canaux de communication.

Dans cette section, nous discutons en détail ces améliorations ; la section 3.2.3.2 est consacrée à l'hybridation de l'algorithme *CIAC* corrigé avec la recherche locale de Nelder-Mead ; l'algorithme final *HCIAC* est décrit étape par étape dans la section finale.

Gestion des spots Dans l'algorithme *CIAC*, les spots sont utilisés pour décrire l'espace de recherche comme une fonction des intérêts des régions. Chaque fourmi peut déposer un spot et la fonction d'intérêt dépend plus du nombre de spots dans une région que de la concentration de ces spots. Ainsi, pour certaines fonctions au paysage particulièrement tourmenté, un nombre excessif de spots peut être déposé. Le problème dans ce cas est l'utilisation excessive de la mémoire vive de l'ordinateur, nécessaire pour gérer les vecteurs

utilisés, les temps de calcul peuvent être prohibitifs quand le nombre de variables de la fonction objectif est élevé.

La solution que nous avons choisie consiste à replacer l'information d'intérêt dans la concentration en phéromone et non plus dans la répartition spatiale des spots. Chaque fourmi décidant de déposer un spot de phéromone peut alors renforcer un spot existant plutôt qu'en déposer un nouveau. L'utilisation de la mémoire vive est ainsi limitée, le nombre de vecteurs à gérer étant maintenu bas.

En pratique, le choix entre le dépôt d'un nouveau spot et le renforcement d'un spot existant est fait selon un paramètre de "résolution". Ici encore, dans l'idée de maintenir une conception parallèle de l'algorithme, chaque fourmi possède son propre paramètre de résolution, qu'on peut comparer à une "zone visible". Le paramètre de résolution est modifié dynamiquement selon l'environnement de la fourmi pendant ses déplacements. Simplement, quand une fourmi termine une recherche locale et trouve finalement un spot dans sa zone visible, elle le renforce et le paramètre de résolution est réduit à la distance entre la fourmi et le spot visible le plus éloigné (voir figure 3.9).

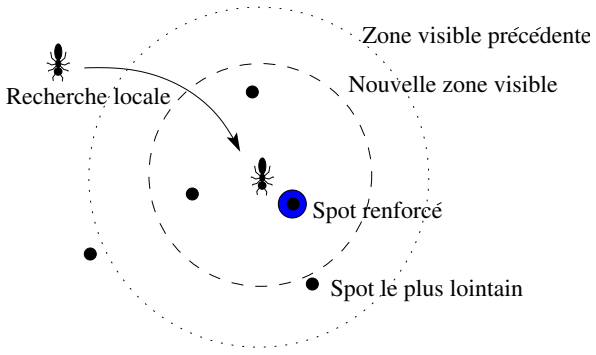


FIG. 3.9 – *Gestion des spots et du paramètre de résolution : après une recherche locale, si la fourmi trouve des spots, elle renforce le plus proche et sa zone visible est réduite à la distance du spot le plus lointain.*

Avec ce mécanisme, la zone visible de la fourmi est automatiquement ajustée pendant le processus de recherche, selon la granularité de l'espace de recherche. Du point de vue du système entier, c'est bien un paramètre de résolution, car il décrit la granularité locale de la fonction objectif et décroît pendant la recherche. On peut comparer ce comportement avec celui de la température du recuit simulé, mais avec l'avantage que la température décroît ici automatiquement en fonction des informations locales recueillies à partir du paysage de la fonction objectif (voir figure 3.10). Il faut noter que le comportement exponentiel observé sur la figure b n'est pas explicitement codé, mais émerge du système.

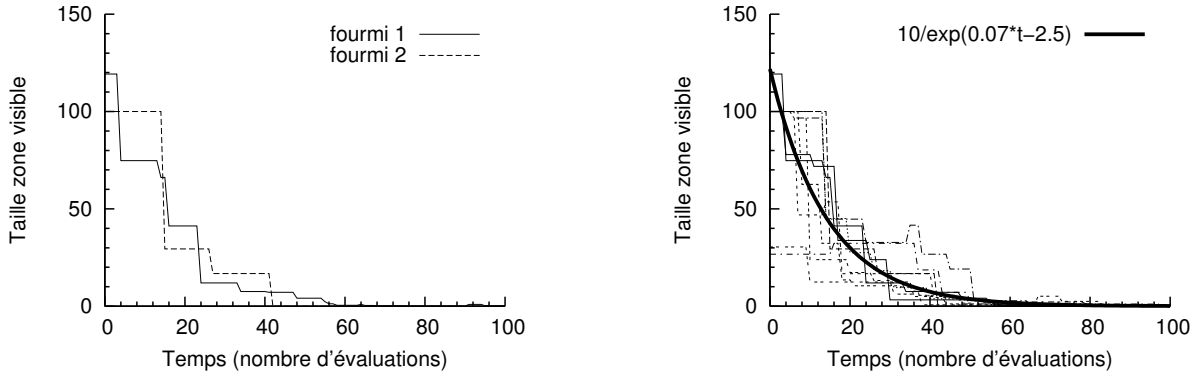


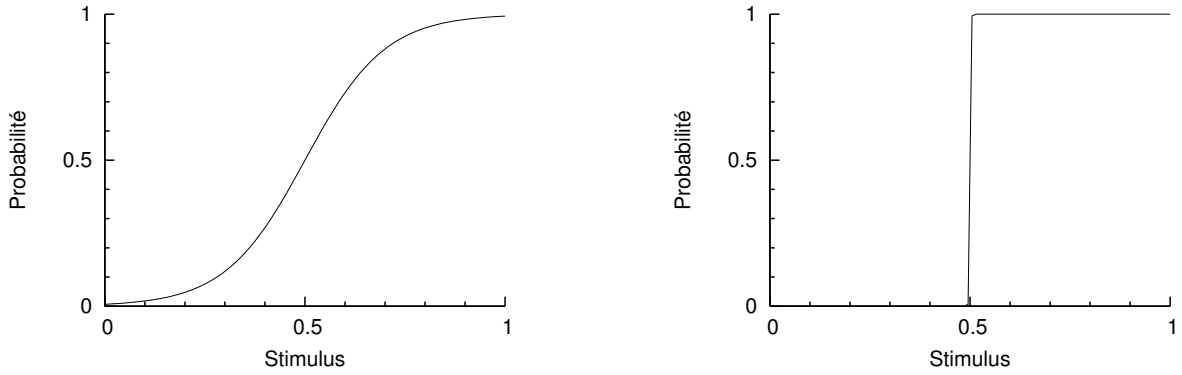
FIG. 3.10 – *Décroissance du paramètre de résolution (a) pour deux fourmis, (b) avec 10 fourmis, la dynamique globale exponentielle apparaît.*

Choix par seuils Dans des systèmes biologiques comme les colonies de fourmis, aucun choix n'est fait sur une logique binaire vrai/faux. En fait, le comportement est généralement basé sur ce que les biologistes appellent une fonction de stimulus/réponse. Ces fonctions décrivent comment des décisions — au niveau individuel — sont prises selon une certaine probabilité, suivant un état interne et un stimulus. Une fonction de stimulus/réponse est une équation simple qui donne la probabilité d'une décision $p(s)$ selon un stimulus s :

$$p(s) = \frac{1}{1 + e^{(-\rho \cdot s + \rho \cdot \tau)}}$$

en désignant par ρ la puissance (la "douceur" du choix) et τ le seuil (voir figure 3.11a).

Par exemple, un individu perçoit une piste de phéromone (c'est ici le stimulus), disons que cette piste a une concentration de $s = 0.5$ (dans une unité arbitraire, typiquement située dans $[0, 1]$). Cet individu est relativement sensible aux phéromones, disons de $\tau = 0.3$ (c'est l'état interne, le seuil). Avec $\rho = 10$, la probabilité pour la fourmi de laisser une piste (la décision, dans notre exemple) est alors de $p(s) = 0.88$.



(a) (b)

FIG. 3.11 – *Fonctions de stimulus/réponse : (a) choix progressif, (b) choix binaire.*

L'intérêt de telles fonctions pour un algorithme d'optimisation est d'accroître la flexibilité, en apportant plus d'informations dans le système. Comme dans les systèmes biologiques, si le but est de concevoir un algorithme générique, il est impossible de régler précisément les paramètres, un compromis doit être trouvé pour maintenir un bon niveau de performance. Mais avec des choix par seuils, une métaheuristique parallèle comme un algorithme de colonies de fourmis peut éviter ce réglage délicat : une fourmi peut faire un mouvement intéressant, là où une autre échouerait. On peut noter qu'avec cette technique on remplace un paramètre par deux paramètres (τ et ρ), ce qui pourrait être considéré comme une mauvaise solution au problème ; mais nous estimons que la réduction de la difficulté dans le choix des paramètres pour les utilisateurs de la métaheuristique et le gain de flexibilité, en combinaison avec un méta-réglage (voir section 3.2.4.1), justifient ce choix.

3.2.3.2 Hybridation

Lier les deux algorithmes Deux versions de l'algorithme hybride ont été implémentées (voir figure 3.12), suivant les relations possibles entre les deux méthodes : relation temporelle (la recherche locale est lancée à intervalles réguliers, à partir de la meilleure solution) ou relation spatiale (la recherche locale est lancée indépendamment par chaque fourmi). Nous qualifierons la première de "simple", car elle consiste simplement à améliorer la solution à un temps donné. La seconde hybridation permet, quant à elle, de maintenir l'aspect décentralisé qui caractérise les algorithmes de colonies de fourmis. En effet, chaque agent décide d'effectuer une recherche locale seulement sur la base des informations dont il dispose, et non pas (comme dans le premier cas) sur la base de l'information portée par le système dans son ensemble. On peut qualifier cette hybridation de "décentralisée".

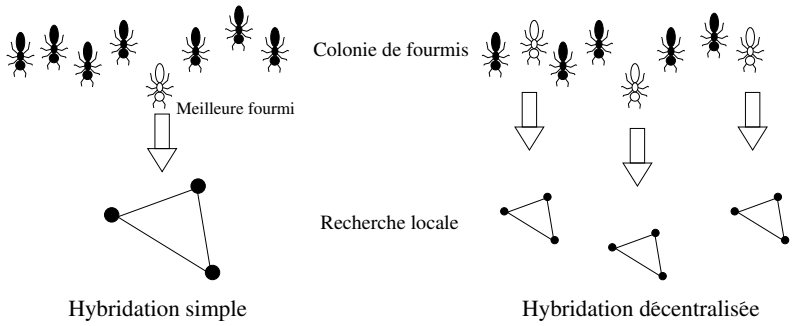


FIG. 3.12 – *Deux hybridations possibles entre les deux méthodes.*

L'hybridation simple est facile à implémenter et amène un comportement simple de l'algorithme. Mais elle perd la structure parallèle de l'algorithme, car elle nécessite un contrôle global déterminant la meilleure fourmi. De fait, l'hybridation décentralisée est plus efficace, car elle prend en compte l'information recueillie lors de la recherche locale, ne la considérant pas uniquement comme une manière d'améliorer un résultat final, mais comme une manière de simplifier le paysage de la fonction objectif.

Motivation Partant de l'idée que l'hybridation décentralisée est plus intéressante pour la préservation des propriétés des algorithmes de colonies de fourmis, nous devons décider du moment du lancement de la recherche locale. Dans l'optique de maintenir l'architecture parallèle de l'algorithme, la décision doit être prise par l'agent, en tenant compte d'informations *locales*. Un autre problème est d'éviter une périodicité simple de lancement de la recherche locale, car elle ne se conformerait alors pas à l'information recueillie par les fourmis. La recherche locale doit être utilisée par une fourmi seule, selon les informations locales et uniquement lorsque cela est nécessaire.

Les règles que nous avons choisies sont fondées sur la notion de "motivation", qui décrit une probabilité de démarrage d'une recherche locale. Chaque fourmi possède un "compteur interne", utilisé comme un stimulus dans une fonction de choix par seuils, pour décider si elle effectue une recherche locale ou envoie un message. La motivation s'accroît légèrement quand la fourmi ne perçoit aucun spot, n'a pas de message en attente et ne vient pas d'effectuer une recherche locale.

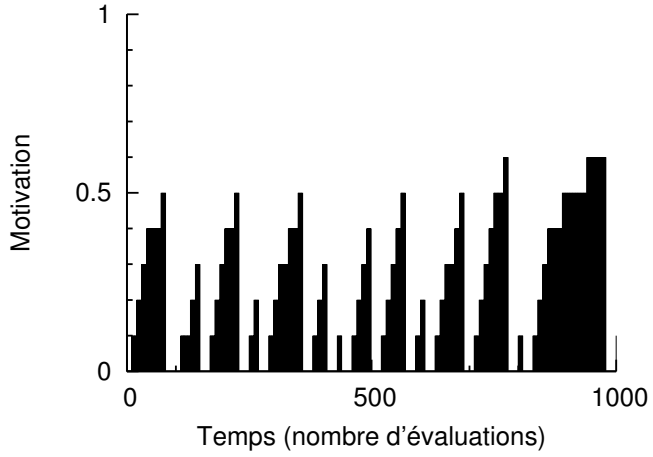


FIG. 3.13 – *Dynamique du paramètre de motivation pour une fourmi (le seuil est ici de* 0.5*).*

La dynamique de la motivation est représentée sur la figure 3.13. On peut noter que la recherche locale est lancée (quand la motivation retourne à 0) principalement quand la motivation atteint 0.5, ce qui correspond au seuil de la fonction de stimulus/réponse. Mais la recherche locale peut également être lancée avec une motivation de 0.1, conformément au caractère probabiliste du processus de décision.

3.2.3.3 Algorithme final

L'algorithme *HCIAC* est détaillé sur la figure 3.14.

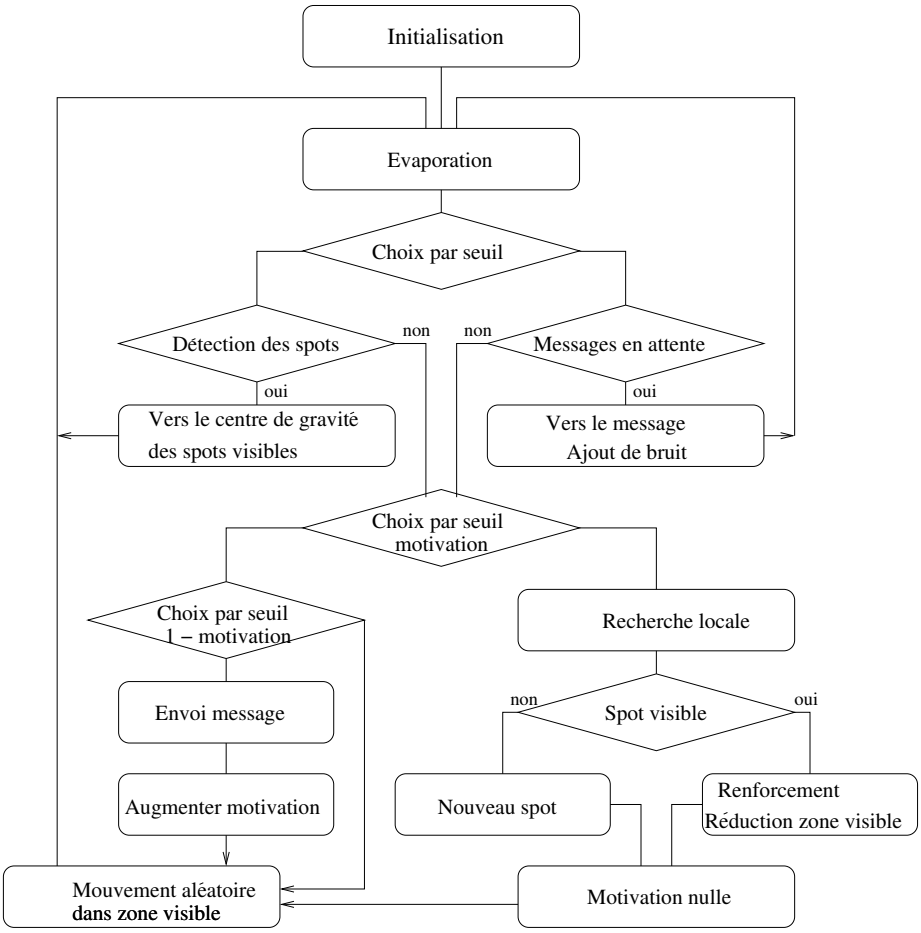


FIG. 3.14 – *L'algorithme* HCIAC.

La première étape de l'algorithme consiste à placer aléatoirement η fourmis sur l'espace de recherche, selon une distribution uniforme, et à initialiser tous leurs paramètres.

Les spots de phéromone s'évaporent alors, la valeur $\tau_{j,t+1}$ de chaque spot j à l'instant $t+1$ étant calculée selon :

$$\tau_{j,t+1} = \rho \cdot \tau_{j,t}$$

où ρ désigne le paramètre de persistance.

À cette étape, une décision est prise, conformément à une fonction stimulus/réponse : la fourmi choisit quel canal de communication elle va utiliser. Ici, deux paramètres de la fonction de choix sont appelés χ_τ pour le seuil et χ_ρ pour la puissance. Ces paramètres sont indiqués pour la population entière, chaque fourmi i ayant ses propres paramètres χ_i, initialisés à la première étape, selon une distribution normale $\mathcal{N}_{\chi_m,\chi_d}$.

Si la fourmi choisit le canal "piste", elle cherche un spot dans sa zone visible π_i (voir section 3.2.3.1), qui est initialisée selon $\mathcal{N}_{\pi_m,\pi_d}$ à la première étape. S'il existe des spots,

elle se déplace vers le centre de gravité des spots visibles, sinon elle va à l'étape de décision sur la motivation.

Au contraire, si la fourmi choisit le canal direct, elle inspecte sa pile de messages. Si des messages sont présents, elle se déplace vers le point indiqué par le message et ajoute alors un peu de bruit à son nouveau vecteur (i.e. elle se déplace aléatoirement dans sa zone visible), sinon, elle va à l'étape de décision sur la motivation.

S'il n'y a aucun spot ni aucun message, la fourmi prend une décision fondée sur sa motivation ω. Le choix est fait selon une fonction stimulus/réponse, le seuil ω_ρ et la puissance ω_τ sont initialisés à la première étape pour la population. Le stimulus ω_i est mis à zéro à la première étape.

Le premier choix mène au canal direct, et donc à la gestion des messages. À cette étape, une autre décision est prise selon une fonction de stimulus/réponse ayant les mêmes paramètres qu'à l'étape précédente, sauf que le stimulus est $(1 - \omega_i)$. Si le choix est fait de gérer les messages, un message est envoyé à une fourmi choisie au hasard et la motivation est augmentée d'une petite quantité ω_δ. Si le second choix est d'ignorer les messages, la fourmi se déplace alors aléatoirement.

Si la décision sur motivation est de prendre en compte la recherche locale, une recherche de Nelder-Mead est lancée avec la position de la fourmi comme point de départ et le rayon de la zone visible comme longueur de l'étape initiale, la recherche locale est limitée à ν évaluations de la fonction objectif. Après cette recherche locale, la fourmi recherche des spots visibles. S'il y a un spot, elle le renforce selon :

$$\tau_{j,t+1} = \tau_{j,t} + \frac{\tau_{j,t}}{2}$$

et le rayon de la zone visible est réduit à la distance du spot visible le plus lointain. Sinon, s'il n'y a aucun spot, la fourmi en dépose un nouveau, avec une concentration égale à la valeur de la fonction objectif en ce point. Après le canal "piste", la motivation est remise à zéro.

La dernière étape possible consiste à effectuer un mouvement aléatoire dans la zone visible. L'algorithme s'arrête s'il ne peut trouver de meilleur optimum pendant θ évaluations de la fonction objectif.

3.2.4 Réglage de *HCIAC*

3.2.4.1 Méta-réglage

Le réglage des paramètres est toujours une étape délicate dans la conception d'une métaheuristique. De fait, il s'agit d'un problème d'optimisation, la qualité des perfor-

mances de l'algorithme forme la fonction objectif et les variables du problème sont les paramètres de la métaheuristique [Grefenstette, 1986].

La fonction objectif est difficile à décrire, car il est difficile de qualifier les performances d'un algorithme d'optimisation. Un point de vue intéressant peut être amené si l'on considère les principaux aspects de la programmation à mémoire adaptative, développés dans la section 2.2. En effet, deux aspects sont notamment mis en avant quant à la conception de métaheuristiques : l'intensification et la diversification. Or, ces concepts recouvrent ce qui est le plus souvent demandé à une métaheuristique : respectivement être rapide (effectuer l'optimisation avec un minimum d'appels à la fonction objectif) et précise (trouver l'optimum global avec une erreur minimale). Ces objectifs sont contradictoires, un algorithme rapide aura tendance à être peu précis, et inversement ; il s'agit donc ici d'un problème d'optimisation biobjectif.

Complexité Cette idée nous a conduit à considérer le réglage des paramètres comme un problème d'optimisation biobjectif. Nous soutenons qu'il existe une perte de complexité, de telle sorte qu'optimiser un algorithme d'optimisation n'est pas un raisonnement sans fin. En effet, régler une métaheuristique se fait généralement de façon plus ou moins empirique, celle-ci utilisant rarement plus de dix paramètres sensibles, le problème d'optimisation correspondant n'est pas un problème très complexe. De plus, les métaheuristiques étant a priori conçues pour optimiser une grande variété de problèmes différents sans modifications, le réglage peut être effectué une fois pour toutes par le concepteur.

Optimisation biobjectif Pour nos tests sur le méta-réglage, nous avons utilisé deux algorithmes biobjectifs différents :

— MOGA, un algorithme génétique multiobjectif [Fonseca and Fleming, 1993], avec les paramètres utilisés dans [Collette, 2002].

— Tribes, un algorithme d'optimisation par essaim particulaire très simple [Clerc, 2004].

Le front de Pareto isolé par l'algorithme biobjectif dessine les optima dans l'espace des fonctions objectifs. Il traduit les compromis possibles entre les deux objectifs.

Indice i/d En outre, le front de Pareto permet de réduire le problème du réglage d'un grand nombre de paramètres à un réglage d'un seul paramètre, que nous appelerons *indice d'intensification/diversification* (indice *i/d*). En effet, avec la répétition des optimisations biobjectives, il est possible d'isoler un ensemble de fronts de Pareto tenant compte du bruit de la fonction objectif (les métaheuristiques étant généralement stochastiques). Le nuage de points ainsi formé peut — sous certaines conditions — être approché par un polynôme qui permettra de choisir des points régulièrement répartis sur l'espace de recherche. Ces

points, associés à un indice, par exemple sur une échelle de 10, dessinent une transition d'un comportement d'intensification vers un comportement de diversification.

Nous utiliserons les notations suivantes pour les fonctions objectifs utilisées : F_i la rapidité (intensification) et F_d la précision (diversification).

L'indice i/d est considéré comme étant utilisable pour la plupart des problèmes d'optimisation classique. Mais si un utilisateur doit gérer des problèmes plus spécifiques, il est alors nécessaire de lancer à nouveau un méta-réglage, en incluant les contraintes du problème dans la fonction biobjectif.

Étude de comportement Le front de Pareto est un bon indicateur du comportement de l'algorithme. Ainsi, son étude dans l'espace des paramètres permet de comprendre leur impact sur l'optimisation.

3.2.4.2 Réglage de *HCIAC*

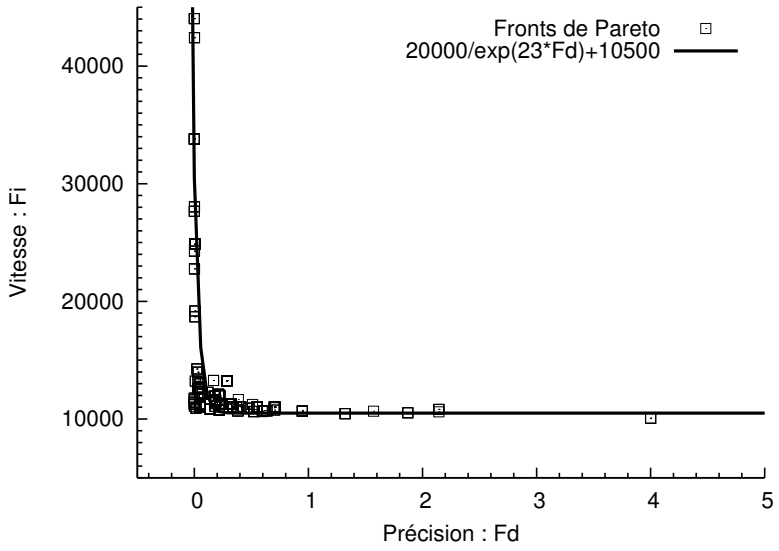


FIG. 3.15 – *Superposition de 20 fronts de Pareto pour les paramètres de HCIAC, dans l'espace des objectifs (la vitesse est exprimée en nombre d'évaluations).*

Comme le montre la figure 3.15, le front de Pareto est bien approché ($R^2 = 0.95$) par la fonction exponentielle :

$$F_i = \frac{2 \cdot 10^4}{e^{(23 \cdot F_d)}} + 105 \cdot 10^2$$

Pour permettre un réglage simple des paramètres, nous avons choisi des points sur le front de Pareto régulièrement répartis sur la fonction exponentielle, afin de définir l'indice

d'intensification/diversification. Par la suite, nous utiliserons la notation P_i pour désigner le point d'indice i sur l'indice i/d. Chaque point P_i est donc associé aux valeurs des paramètres correspondants, comme le montre le tableau 3.4. Les valeurs des paramètres ont été arrondies pour faciliter la lecture.

TAB. 3.4 – *Indice d'intensification/diversification pour les paramètres de* HCIAC.

P_i	1	2	3	4	5	6	7	8	9	10
F_i	1.9e-5	1e-4	5e-3	5e-3	0.16	0.2	0.7	1.3	1.8	4
F_d	44000	33800	20000	19000	14000	13200	11000	10500	10500	10000
ν	950	930	910	890	500	400	400	340	264	17
η	950	360	100	100	150	80	150	45	12	12

Comme le montre la figure 3.15, le front de Pareto forme un angle aigu, il existe donc un compromis satisfaisant entre les deux objectifs. Un bon choix pour la plupart des problèmes serait donc un indice i/d compris entre 4 et 7.

Nous n'avons pas reporté ici l'ensemble des paramètres, car seuls le nombre maximal d'évaluations par la recherche locale (ν) et le nombre de fourmis (η) sont significatifs : ils possèdent une forte variance sur l'ensemble du front de Pareto. Les autres paramètres possèdent uniquement une faible influence sur les performances de l'algorithme (faible variance), comparativement à ν et η. Nous avons constaté que les mêmes valeurs se retrouvent approximativement sur le front de Pareto et sont donc adaptées à être initialisées par défaut : $\rho = 0.5$, $\chi_m = 0.5$, $\chi_d = 0.2$, $\pi_m = 0.8$, $\pi_d = 0.5$, $\omega_\delta = 0.1$, $\chi_\tau = \omega_\tau = 0.5$, $\chi_\rho = \omega_\rho = 10$.

Pour le reste de ce travail, nous utiliserons uniquement une valeur de l'indice i/d pour effectuer les comparaisons avec les algorithmes concurrents. La raison de ce choix est que les résultats présentés dans la littérature le sont uniquement pour un jeu de paramètres. Afin de travailler dans les mêmes conditions, nous avons choisi l'indice P_4 (cf. tableau 3.4) qui permet un compromis souvent observé dans les tests sur des problèmes continus : une importance plus grande est donnée à la précision qu'à la rapidité.

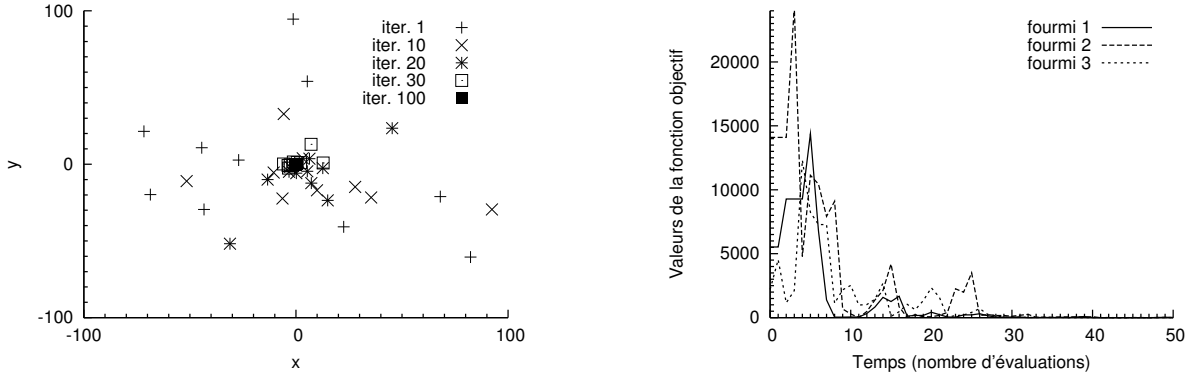
Finalement, un des paramètres les plus sensibles est θ, la sensibilité du critère d'arrêt. Ce dernier n'est pas considéré ici comme un paramètre de *HCIAC*, car il est fortement dépendant du problème d'optimisation. Nous avons donc choisi de le figer et de ne pas le prendre en compte dans le méta-réglage, la valeur choisie empiriquement est de $\theta = 10000$.

3.2.5 Résultats expérimentaux et discussion

3.2.5.1 Comportement de *HCIAC*

Afin d'illustrer le comportement de *HCIAC*, nous l'avons d'abord testé sur la fonction B_2 (définie dans l'annexe A.2.2).

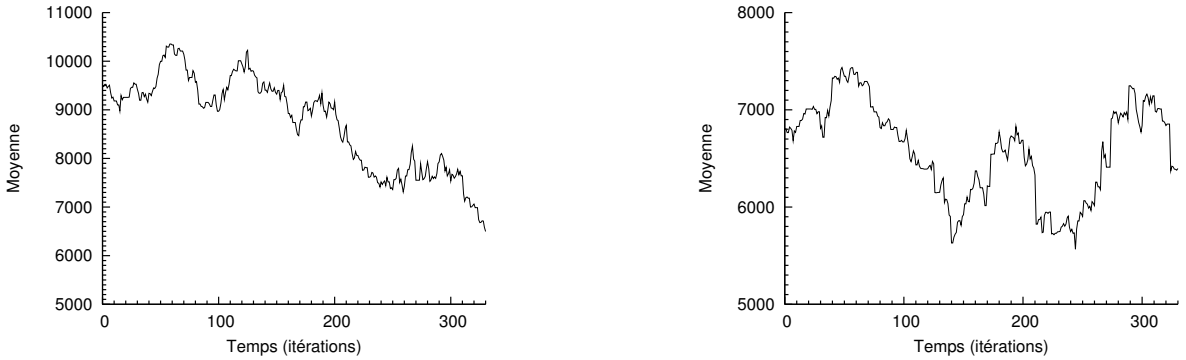
Nous avons utilisé uniquement 10 fourmis, afin de rendre les figures plus lisibles, tous les paramètres sont initialisés avec les valeurs décrites précédemment.



(a) (b)

FIG. 3.16 – *L'algorithme* HCIAC *optimisant la fonction* B_2. *(a) positions* x *et* y *des* 10 *fourmis à différentes étapes de l'optimisation, (b) valeurs de la fonction objectif trouvées par* 3 *fourmis durant les* 50 *premières évaluations.*

La figure 3.16 montre que la dynamique globale de l'algorithme consiste en une convergence vers l'optimum global. En effet, la recherche locale permet aux fourmis de progresser rapidement vers les optima locaux. Le canal direct permet de forcer les fourmis à sortir des optima locaux, en combinaison avec le canal "piste" qui élabore une description de l'espace de recherche. Comme nous l'avons décrit dans [Dréo and Siarry, 2004c], les deux canaux travaillent ensemble pour adapter le comportement de l'algorithme à la fonction objectif. En effet, avec l'utilisation conjointe des deux canaux apparaît une oscillation de la déviation de l'écart-type des valeurs de la fonction objectif trouvées par les fourmis. Ce phénomène est toujours présent dans *HCIAC*, comme indiqué par la figure 3.17.



(a) (b)

FIG. 3.17 – *L'algorithme* HCIAC *optimisant la fonction* $B2$. *(a) moyenne des valeurs de la fonction, (b) écart-type des même valeurs.*

3.2.5.2 Résultats

Nous avons testé l'algorithme *HCIAC* sur un jeu de fonctions analytiques trouvé dans la littérature. Afin de comparer *HCIAC* avec d'autres métaheuristiques d'optimisation continue, nous avons choisi un jeu de 13 fonctions, en partie dans des articles traitant des algorithmes continus de colonies de fourmis [Bilchev and Parmee, 1995, Mathur et al., 2000, Monmarché et al., 2000b] et en partie dans la littérature concernant l'optimisation continue [Storn and Price, 1995], [Chelouah and Siarry, 2000a] [Chelouah and Siarry, 2000b].

Pour éviter les problèmes dus au choix de la population initiale, nous avons lancé chaque test 100 fois, avec une initialisation différente du générateur aléatoire à chaque test. Le nombre d'évaluations de la fonction objectif ("evals") et l'erreur moyenne ("err") sont obtenus à travers un moyennage des résultats sur tous les tests (les écart-types sont donnés entre parenthèses). Un test est considéré comme étant réussi (contribuant ainsi au "%ok") si la condition suivante est vérifiée :

$$|f^{\alpha} - f^{*}| < \epsilon_1 \cdot f^{*} + \epsilon_2 \tag{3.5}$$

en désignant par :

f^{*} l'optimum global de la fonction objectif ;

f^{α} l'optimum trouvé par l'algorithme ;

ϵ_1 et ϵ_2 des paramètres de précision : dans tous les tests de cette section, $\epsilon_1 = \epsilon_2 = 10^{-4}$.

Afin de comparer *HCIAC* avec deux algorithmes de colonies de fourmis continus, nous avons effectué plusieurs tests avec les mêmes fonctions que celles utilisées dans les articles

correspondants. Les résultats sont présentés dans le tableau 4.4. Les valeurs données entre crochets décrivent des tests où le nombre maximum d'itérations est fixé, sans aucun autre critère d'arrêt, une cellule vide indique qu'aucune valeur n'est accessible dans la littérature [Monmarché et al., 2000b, Bilchev and Parmee, 1995].

TAB. 3.5 – *Résultats de HCIAC et d'algorithmes de colonies de fourmis concurrents.*

	CACO			API		
Fonction	% ok	evals	err	% ok	evals	err
$R2$	100	6842	0.00		[10000]	0.00 (0.00)
SM	100	22050			[10000]	0.00 (0.00)
$Gr5$					[10000]	0.18 (0.04)
$Gr10$	100	50000	0.0			
GP	100	5330				
MG	100	1688				
St		[6000]	0.0			
B_{f1}		[200000]	99456			
B_{f2}		[200000]	99123			
B_{f3}		[200000]	45769			

	CIAC			HCIAC		
Fonction	% ok	evals	err	% ok	evals	err
$R2$	100	11797	3e-3(3e-3)	100	18747(6697)	1e-8(4e-9)
SM	100	50000	9e-10(1e-11)	100	18616(6715)	5e-8(1e-8)
$Gr5$	63	48402	0.01(9e-3)	75	10870(1347)	1e-4(2e-4)
$Gr10$	52	50121	0.05(0.05)	18	23206(13132)	1e-3(4e-4)
GP	56	23391	1.51(2.33)	100	34533(4086)	0(0)
MG	20	11751	0.34(0.19)	100	24596(11413)	4e-9(4e-9)
St	94	28201	1.96(1.61)	100	10726(219)	0(0)
B_{f1}	0	50000	99521(463)	0	20388(7466)	9999(3e-3)
B_{f2}	0	50000	99635(512)	0	18734(6902)	9999(2e-3)
B_{f3}	0	50000	99895(86)	0	19685(6863)	9999(1e-3)

HCIAC est meilleur que *CIAC* pour la plupart des fonctions tests, sauf pour les fonctions Baluja et la fonction Gr_{10}. Concernant les autres algorithmes de colonies de fourmis, nous pouvons observer que la rapidité de *HCIAC* tend à être similaire, mais du fait du manque de données dans la littérature, la comparaison s'avère difficile. En résumé, comparativement aux autres algorithmes de colonies de fourmis, *HCIAC* effectue une optimisation précise et efficace, mais nécessite un grand nombre d'évaluations.

Nous avons enfin comparé *HCIAC* avec d'autres métaheuristiques d'optimisation continue, les résultats sont présentés dans le tableau 4.5.

TAB. 3.6 – *Résultats de* HCIAC *et de quatre métaheuristiques d'optimisation continue concurrentes.*

	CGA			ECTS			DE		
Fonction	% ok	evals	err	% ok	evals	err	% ok	evals	err
St							100	1300	
Gr_{10}							100	12804	
SM	100	750	0.0002	100	338	3e-8	100	392	
R_2	100	960	0.004	100	480		100	615	
R_5	100	3990	0.15	100	2142	0.08			
GP	100	410	0.001	100	231	2e-3			
$S_{4,5}$	76	610	0.14	75	825	0.01			

	CIAC			HCIAC		
Fonction	% ok	evals	err	% ok	evals	err
St	94	8201	1.96(1.61)	100	726(219)	0(0)
Gr_{10}	52	50121	0.05(0.05)	18	23206(13132)	1e-3(4e-4)
SM	100	50000	9e-10(1e-11)	100	18616(6715)	5e-8(1e-8)
R_2	100	11797	3e-3(3e-3)	100	18747(6697)	1e-8(4e-9)
R_5	90	39546	8e-3(7e-3)	100	19469(6606)	6e-8(2e-8)
GP	56	23391	1.51(2.33)	100	34533(4086)	0(0)
$S_{4,5}$	5	39311	6.34(1.01)	100	17761(5976)	0(0)

Nous pouvons constater que, si *HCIAC* ne concurrence pas les autres algorithmes en terme de rapidité, il surpasse *CIAC* en vitesse et efficacité. De plus, *HCIAC* surpasse les autres algorithmes en ce qui concerne la précision et l'efficacité (excepté pour les fonctions Griewangk, qui paraissent réellement difficiles pour *HCIAC*).

3.2.6 Conclusion

Nous avons proposé un algorithme (algorithme de colonies de fourmis interagissantes[1]) hybridé avec une recherche locale, pour améliorer ses performances. Nous avons montré que le nouvel algorithme *HCIAC* est meilleur que *CIAC*, particulièrement en termes de précision et d'efficacité.

En comparant *HCIAC* avec des algorithmes concurrents, nous avons montré le grand nombre d'évaluations nécessaires, hérité du concept d'algorithme de colonies de fourmis et de l'utilisation d'un critère d'arrêt moins sensible dans nos tests. Mais nous avons également montré l'efficacité et la précision de l'algorithme. Si un problème test demandait une plus grande vitesse, nous recommanderions l'utilisation de l'indice de diversification/intensification. En effet, c'est un bon outil pour adapter la méthode à un problème

[1] Hybrid Continuous Interacting Ant Colony, *HCIAC*

donné. Un utilisateur devant adapter l'algorithme à son problème pourrait tout simplement décider s'il veut un algorithme rapide ou précis. *HCIAC* hérite également des qualités du concept de colonies de fourmis, comme l'auto-gestion et la flexibilité.

3.3 Élaboration d'un algorithme pour l'optimisation dynamique (*DHCIAC*)

3.3.1 Introduction

Récemment, l'optimisation de problèmes dynamiques a suscité l'intérêt des chercheurs de domaines variés, permettant ainsi le développement de plusieurs algorithmes de plus en plus puissants. Contrairement à l'optimisation statique, où l'objectif final est de trouver l'optimum global, le but de l'optimisation dynamique est de trouver et de suivre l'évolution d'un optimum global, pendant tout le temps de l'optimisation.

L'optimisation dynamique a récemment été étudiée sous le nom "d'optimisation temps réel" (*Real Time Optimization*, RTO), principalement dans le champ de l'optimisation basée sur des modèles en temps réel d'usine chimique [Xiong and Jutan, 2003] [Yip and Marlin, 2003a] [Yip and Marlin, 2003b]. Elle a également été étudiée sur des "problèmes non-stationnaires" [Morrison and De Jong, 1999] [Branke, 2001] ou des "environnements dynamiques" [Cobb and Grefenstette, 1993] [Carlisle, 2002] [Ourique et al., 2002]. Il existe plusieurs domaines d'étude sur l'optimisation dynamique. Notre but est de nous concentrer sur l'utilisation de métaheuristiques pour l'optimisation de problèmes difficiles, continus et dynamiques. Peu de métaheuristiques ont été adaptées sur de tels problèmes, principalement des algorithmes génétiques [Branke, 2001] et des méthodes d'optimisation par essaims particulaires [Ourique et al., 2002]. Les algorithmes de colonies de fourmis ont été employés sur des problèmes dynamiques discrets [Schoonderwoerd et al., 1996, Di Caro and Dorigo, 1998a, Di Caro and Dorigo, 1998b], mais pas sur des problèmes continus. Enfin, un algorithme de recherche locale a été conçu pour de tels problèmes [Xiong and Jutan, 2003].

Nous proposons dans cette section un nouvel algorithme inspiré du comportement des colonies de fourmis. La méthode, appelée "Dynamic Hybrid Continuous Interacting Ant Colony" (*DHCIAC*), utilise une population d'agents, s'articule autour des notions d'hétérarchie et de canaux de communication [Dréo and Siarry, 2004c] et utilise un algorithme de recherche locale de Nelder-Mead [Dréo and Siarry, 2003d]. Deux versions de l'algorithme sont proposées, utilisant deux approches différentes pour optimiser des problèmes dynamiques. La méthode est testée sur un nouveau jeu de test dynamique,

que nous avons construit en tentant de couvrir un large panel de problèmes dynamiques différents [Dréo and Siarry, 2004b].

Cette section comprend cinq parties supplémentaires. Dans la section 3.3.2 nous décrivons les fonctions continues dynamiques que nous avons utilisées pour tester l'algorithme. Dans la section suivante, nous présentons l'algorithme *DHCIAC*. Le réglage des paramètres est étudié ensuite. Les résultats expérimentaux sont discutés dans la section 3.3.3.3 ; enfin la conclusion forme la dernière section.

3.3.2 Élaboration d'un jeu de fonctions de test

Le choix d'une méthode efficace pour comparer des algorithmes d'optimisation dynamique est quelque peu difficile. En effet, le domaine ne dispose actuellement pas d'un jeu de fonctions de test *complet* et *exhaustif*, pouvant être employé pour comparer les performances des méthodes. Un des objectifs de ce travail est de proposer un tel jeu de fonctions de test, pouvant être utilisé comme outil de comparaison.

Nous pouvons différencier deux composants principaux d'un problème d'optimisation dynamique : le premier est la *structure* de la fonction objectif et le second, l'*évolution* de cette structure. Par la suite, nous appelerons le premier composant la *fonction de structure*, et le second la *fonction de temps*. Un problème d'optimisation dynamique peut donc s'écrire :

$$D(x,t) = S(x, T(t))$$

en désignant par S la fonction de structure et T la fonction de temps. La relation entre S et T (i.e. *ce qui* change) est appelée la "cible de la variation".

3.3.2.1 Fonctions de structure

Les fonctions de structure utilisées sont des fonctions analytiques classiquement utilisées en optimisation. Afin de concevoir un jeu cohérent de fonctions de test, nous avons choisi de déterminer quelles caractéristiques sont les plus importantes. Nous avons déterminé douze caractéristiques pouvant être utilisées :

— continuité,
— convexité,
— symétrie de l'espace de recherche,
— symétrie de la fonction elle même,
— nature quadratique (i.e. utilisation d'une fonction polynomiale du second degré dans un problème),
— modalité (i.e. présence d'un ou plusieurs optima),
— périodicité,

— nombre d'optima locaux,
— nombre de dimensions,
— contraintes,
— position de l'optimum,
— aspect du bassin d'attraction des optima.

3.3.2.2 Cible de la variation

Les cibles de variations peuvent être classées dans plusieurs catégories :
— les variations de la fonction entière,
— les variations de l'optimum seul (ou de l'optimum *et* du bassin d'attraction),
— les variations de la fonction entière, sauf de l'optimum (l'optimum étant donc la seule partie à ne pas subir de variation).

Chaque cible peut changer en position (les coordonnées se déplacent dans l'espace de recherche) ou en valeur (la valeur de la fonction objectif varie). De plus, il existe des variations plus complexes :
— variation du nombre de dimensions,
— variation des contraintes,
— changement de phase (la cible effectue une rotation autour d'un axe),
— changement de la structure entière (la fonction objectif change radicalement, par exemple passant d'une structure convexe à une structure concave).

3.3.2.3 Fonctions de temps

La fonction de temps peut être linéaire (provoquant une évolution infinie du problème), périodique (le problème oscille d'un état à l'autre) ou plus généralement non-linéaire.

Un aspect important à gérer dans la fonction de temps est la *vitesse relative*. En effet, dans des applications temps-réel, nous devons considérer la vitesse de variation du problème *relativement* à la vitesse de l'algorithme. Nous avons donc trois fonctions de temps : lente, rapide et pseudo-discrète. Une fonction lente entraîne de petites variations entre deux appels (i.e. le modèle de la variation peut être trouvé). Une fonction de temps rapide entraîne de grandes variations entre deux appels, mais le modèle sous-jacent peut toujours être trouvé, bien que difficilement. Dans une fonction de temps pseudo-discrète, les variations sont si rapides que le problème apparaît comme différent entre deux itérations. Afin de déterminer quand la vitesse relative est lente ou rapide, nous utilisons la notion de *tour d'optimisation.* La longueur d'un tour est définie comme le nombre d'évaluations possibles entre deux variations. Par exemple, un tour de longueur 10 nécessitera dix évaluations de la fonction objectif.

Dans nos tests, un tour de 20 évaluations est considéré comme lent, et un tour de 1 évaluation comme rapide.

3.3.2.4 Jeu de test

Nous avons élaboré dix problèmes de test couvrant différents aspects de l'optimisation dynamique. Le jeu de test complet utilisé dans cette section est présenté dans l'annexe A.4. L'algorithme est ici testé sur chaque fonction, aux trois vitesses relatives, pour un total de trente problèmes d'optimisation différents.

3.3.3 L'algorithme *DHCIAC*

L'algorithme *DHCIAC* est fondé sur l'algorithme hybride statique *HCIAC* .

3.3.3.1 Principe

L'algorithme *HCIAC* peut être adapté à des problèmes dynamiques, grâce à sa nature flexible. En effet, *HCIAC* tire avantage du concept d'algorithme hétérarchique, notamment des canaux de communication pouvant être utilisés pour diffuser rapidement des informations sur les variations de la fonction objectif. De plus, *HCIAC* met en jeu une recherche locale, pouvant facilement être adaptée aux fonctions dynamiques.

Deux versions différentes de l'algorithme *DHCIAC* ont été développées : $DHCIAC_{find}$ et $DHCIAC_{track}$. En effet, il existe deux stratégies différentes pour trouver un optimum, lors de l'optimisation d'un problème dynamique [Berro, 2001] :

— trouver l'optimum très rapidement, dès que les changements apparaissent (stratégie "find"),

— suivre l'évolution d'un optimum (stratégie "track").

DHCIAC peut utiliser ces deux stratégies en adaptant la partie recherche locale de l'algorithme. La version $DHCIAC_{find}$ utilise une recherche de Nelder-Mead classique, alors que $DHCIAC_{track}$ utilise une version modifiée de cet algorithme [Xiong and Jutan, 2003], qui présente alors la particularité de n'employer que l'étape de réflexion.

***DHCIAC*$_{find}$** L'algorithme $DHCIAC_{find}$ possède peu de différences avec l'algorithme *HCIAC*, la stratégie "find" étant proche de celle employée pour les problèmes statiques. Les modifications visent simplement à modifier le comportement de convergence, et à améliorer la vitesse de l'algorithme.

Variations de la zone visible La zone visible permet à *HCIAC* de se focaliser sur la recherche de zones d'intérêt. Dans *HCIAC*, quand la zone visible d'une fourmi change, sa

taille prend la valeur de la distance au spot visible le plus lointain. Dans $DHCIAC_{find}$, la taille de la zone visible est égale à la distance du spot qui permet le plus petit changement de ce paramètre.

DHCIAC$_{track}$ La principale différence avec l'algorithme *HCIAC* est que, dans $DHCIAC_{track}$, la recherche locale n'est pas un algorithme de Nelder-Mead classique, mais un nouveau "simplexe dynamique" [Xiong and Jutan, 2003].

Simplexe dynamique Cet algorithme possède deux différences principales avec la recherche de Nelder-Mead standard. Tout d'abord, il utilise uniquement la réflexion, afin de garder une taille fixe, empêchant ainsi le simplexe de se bloquer avant d'avoir pu suivre l'optimum. Ensuite, il observe uniquement le meilleur point, à la recherche d'une variation. La méthode complète est présentée sur l'algorithme 3.2.

Taille initiale du simplexe Un paramètre crucial dans l'algorithme de recherche locale est la taille du simplexe, puisqu'il ne doit idéalement pas changer [Xiong and Jutan, 2003]. Dans *DHCIAC*, chaque fourmi peut lancer une recherche locale à chaque instant, selon sa "motivation". La taille du simplexe est donc initialisée à la taille de la zone visible par la fourmi.

Variations de la zone visible Dans $DHCIAC_{track}$, nous utilisons le même mécanisme que dans $DHCIAC_{find}$ pour initialiser les tailles des zones visibles.

3.3.3.2 Réglage de *DHCIAC*

Le réglage des paramètres est un problème délicat, particulièrement lorsque différentes métaheuristiques sont utilisées. Le concept de programmation à mémoire adaptative, décrit plus haut, souligne deux principaux composants dans les métaheuristiques : l'intensification et la diversification. Ces principes aident à régler les deux versions de la méthode *DHCIAC*. En effet $DHCIAC_{find}$ vise à chercher un nouvel optimum venant d'apparaître, nécessitant ainsi un comportement plus axé sur la diversification. À l'inverse, $DHCIAC_{track}$ tente de suivre un optimum en évolution, nécessitant donc plutôt un comportement d'intensification.

Dans l'algorithme *HCIAC*, seuls deux paramètres ont un rôle réellement significatif : le nombre maximal d'évaluations pour la recherche locale (ν) et le nombre de fourmis (η). Nous avons observé qu'il était possible de choisir un réglage correspondant à un compromis entre un comportement d'intensification et de diversification, en réglant ces deux paramètres (voir section 3.2.4). Dans le tableau 3.7, nous présentons différentes valeurs

pour ν et η suivant un indice P_i, qui représente un compromis entre une intensification maximale ($P_i = 1$) et une plus grande diversification ($P_i = 10$).

TAB. 3.7 – *Réglages des paramètres ν et η permettant un compromis intensification/diversification satisfaisant pour l'algorithme HCIAC, sur le problème O(P+V)P (voir annexe A.4.6).*

P_i	1	2	3	4	5	6	7	8	9	10
ν	95	93	90	90	50	40	40	34	25	15
η	95	35	10	10	15	8	15	5	3	3

La version *"track"* Comme $DHCIAC_{track}$ est conçu autour de la notion d'intensification, nous avons décidé d'utiliser les paramètres associés à $P_i = 3 : \nu = 90$ et $\eta = 10$.

La version "find" $DCHIAC_{find}$ a un comportement orienté vers la diversification, nous avons donc choisi d'employer les paramètres associés à $P_i = 7 : \nu = 40$ et $\eta = 15$.

Taux d'évaporation élevé Avec $DHCIAC_{track}$, il est nécessaire d'éviter d'être piégé dans des optima locaux. Il ne faut donc pas utiliser un taux d'évaporation bas (visant à mémoriser la position des zones d'intérêt) mais un taux d'évaporation élevé, permettant de mettre l'accent sur le suivi d'un optimum, plutôt que sur la recherche d'un nouvel optimum. La persistance est donc fixée à $\rho = 0.1$.

Taille initiale des zones visibles Un point particulier concernant la recherche locale utilisée dans $DHCIAC_{track}$ est la "longueur du simplexe". Afin de régler ce paramètre conformément à la "granularité" de la fonction objectif (i.e. la distance moyenne entre deux optima locaux), la taille du simplexe est réglée en fonction de la taille de la zone visible d'une fourmi. Il est donc nécessaire d'initialiser les paramètres de la distribution $\aleph_{\pi_m,\pi_d}$. Si la granularité est inconnue, la moyenne et l'écart-type sont fixés : $\pi_m = \pi_d = \frac{S}{n\sqrt{\eta}}$, en désignant par S la taille de l'espace de recherche (sur la plus petite dimension) et n le nombre de dimensions.

3.3.3.3 Résultats expérimentaux et discussion

Nous avons testé *DHCIAC* sur le jeu de test proposé. Pour éviter d'éventuels problèmes dus au choix d'une population initiale, nous avons effectué chaque test 100 fois avec, à chaque fois, une initialisation différente du générateur de nombres aléatoires. Afin de tester l'efficacité de l'algorithme, nous avons mesuré la distance moyenne à l'optimum global du meilleur point trouvé, et ce, pendant une session d'optimisation (fixée à un maximum de

100 évaluations dans nos tests). Quelques résultats sont présentés sur le tableau 3.8, les problèmes rapides y sont présentés en italique et les meilleures valeurs en gras.

TAB. 3.8 – *Moyenne et écart-type de l'erreur pour les deux versions de* DHCIAC *optimisant différents problèmes. Les noms des problèmes sont abrégés selon la notation suivante : la première et la deuxième lettre indiquent la cible (O : optimum, P :position, V :valeur, A :toute la fonction, D : dimension), la troisième lettre indique la fonction de temps (L :linéaire, P :périodique) et la dernière lettre donne la vitesse de variation (S :lente, F :rapide). Les valeurs présentées sont les moyennes sur 100 tests différents.*

	track		find	
Problème	*moyenne*	écart-type	*moyenne*	écart-type
OPLF	**0.56**	**0.33**	0.86	0.65
OPLS	***0.48***	***0.32***	*0.79*	*0.59*
APLF	**18.0**	**9.1**	19.9	17.1
APLS	***17.3***	***7.2***	*19.7*	*15.6*
OVPF	8.1	6.2	**5.0**	**4.7**
OVPS	*7.7*	*5.1*	***4.3***	***3.9***
AVPF	17.1	11.1	**9.5**	**8.2**
AVPS	*15.6*	*9.8*	***8.9***	***7.7***
ADLS	10.1	17.3	7.9	12.1
O(P+V)PS	11.2	8.6	10.9	9.1

La première observation que nous pouvons faire est que *DHCIAC* obtient de meilleures performances sur des problèmes lents. De plus, $DHCIAC_{track}$ est meilleur que $DHCIAC_{find}$ si l'on considère des problèmes rapides. Pour les problèmes possédant une cible de type "position", la version *track* permet une plus faible erreur moyenne, alors qu'elle rencontre des difficultés avec des problèmes fondés sur une cible "valeur". Au contraire, la version *find* atteint de meilleurs résultats sur des problèmes fondés sur la valeur que sur ceux fondés sur la position. Les problèmes mettant en jeu des variations du nombre de dimensions en valeur *et* position paraissent difficiles à optimiser pour les deux algorithmes.

La difficulté de *DHCIAC* sur des problèmes rapides peut facilement être comprise, car, en tant qu'algorithme à population, il nécessite un certain nombre d'évaluations pour effectuer une optimisation correcte. Pour des problèmes rapides, cette caractéristique entraîne un écart entre les mouvements de la première et de la dernière fourmi, ce qui peut perturber l'algorithme. La version $DHCIAC_{track}$ atteint de meilleures performances, car elle peut rapidement atteindre un optimum mouvant, grâce au simplexe dynamique, qui nécessite peu d'évaluations pour progresser.

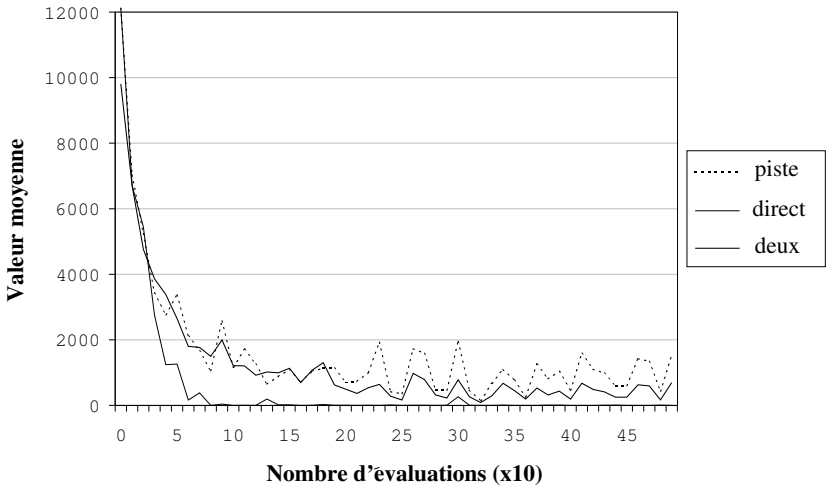
En outre, les deux stratégies qui nous ont guidés pour produire deux versions de *DHCIAC* se montrent efficaces sur des classes différentes de problèmes. En effet, la stratégie

d'intensification (implémentée dans la version *track*) atteint de meilleures performances pour des problèmes de variations de position, alors que la stratégie de diversification (la version *find*) est plus efficace sur des problèmes de variation de valeur.

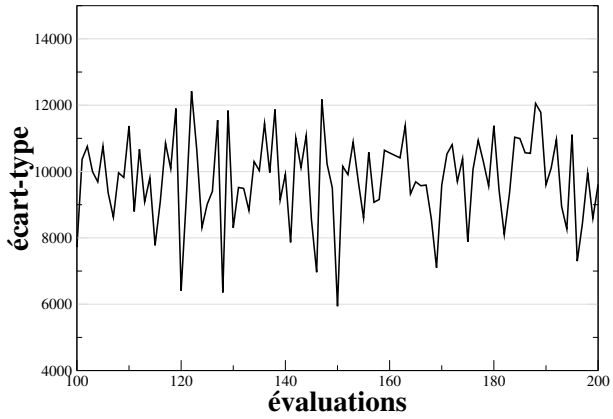
3.3.4 Conclusion

Nous avons montré que l'algorithme *HCIAC* pouvait être adapté à des problèmes d'optimisation continue dynamique, sous la forme de la méthode *DHCIAC*. Nous avons proposé l'utilisation de deux stratégies différentes, fondées sur l'idée de la programmation à mémoire adaptative, notamment sur les notions d'intensification et de diversification. Ces stratégies ont donné lieu à la conception de deux versions de l'algorithme : $DHCIAC_{track}$ pour le comportement d'intensification, et $DHCIAC_{find}$ pour la diversification. Nous avons de plus proposé un nouveau jeu de test pour métaheuristiques dynamiques, afin de disposer d'un ensemble cohérent de fonctions couvrant les principaux aspects de l'optimisation continue dynamique.

Nous avons observé que les deux stratégies sont adaptées à des classes différentes de problèmes, respectivement de variations en valeur pour $DHCIAC_{find}$ et en position pour $DHCIAC_{track}$. De plus, *DHCIAC* paraît être plus approprié pour des problèmes à dynamique lente.



(a) Moyenne



(b) Deux canaux

FIG. 3.6 – *Moyenne et écart-type des valeurs de la fonction objectif dans la population à différentes étapes.*

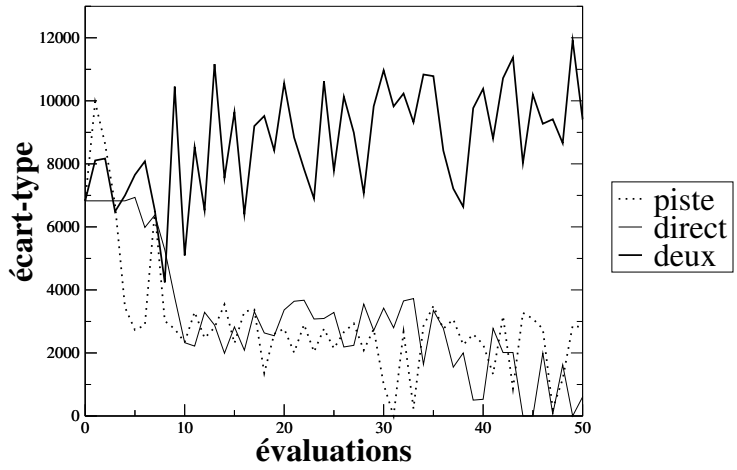
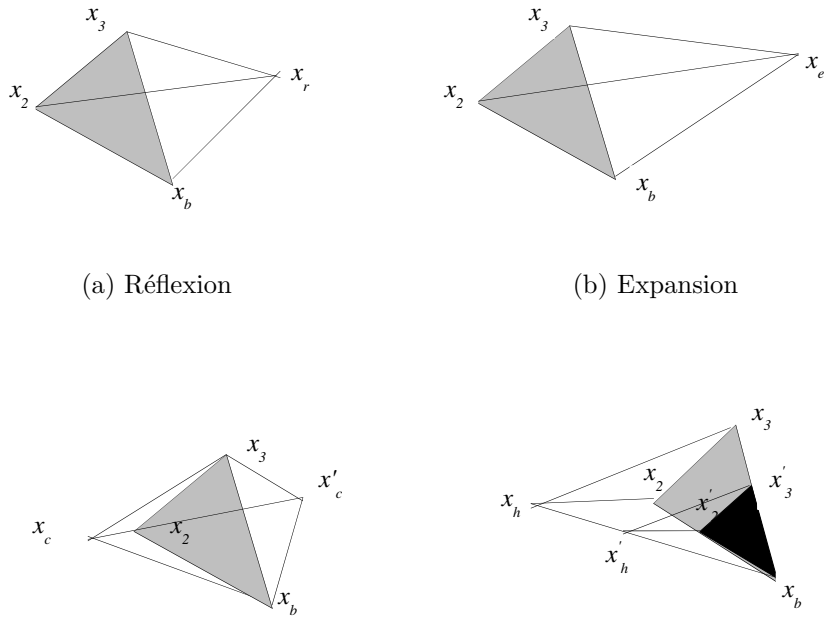


FIG. 3.7 – *Écart-type des valeurs de la fonction objectif dans la population à différentes étapes pour différents canaux.*



(a) Réflexion

(b) Expansion

(c) Contraction (x_c interne, x'_c externe)

(d) Multi-contractions

FIG. 3.8 – *Exemples de modifications du simplexe de Nelder-Mead sur un problème à deux dimensions : réflexion, expansion, contraction externe/interne, multi-contractions.* x_b *est le sommet de valeur minimale et* x_h *le sommet de valeur maximale.*

Algorithme 3.1 La méthode du simplexe de Nelder-Mead.

Initialiser un simplexe et calculer les valeurs de la fonction sur ses noeuds ;

Répéter les itérations $t, t \geq 0$ jusqu'au critère d'arrêt ;

Ordonner les noeuds $x_0^t, x_1^t, \ldots, x_n^t$ conformément à :

$$f\left(x_0^t\right) \leq f\left(x_1^t\right) \leq \ldots \leq f\left(x_n^t\right)$$

Calculer le centre de gravité :

$$\bar{x}^t = \frac{1}{n} \cdot \sum_{i=0}^{n-1} x_i^t$$

Réflexion, calculer le *point de réflexion* à partir de :

$$x_r^t = \bar{x}^t + \rho\left(\bar{x}^t - x_n^t\right)$$

Si $f\left(x_0^t\right) \leq f\left(x_r^t\right) < f\left(x_{n-1}^t\right)$ **alors** $x_n^t \leftarrow x_r^t$, itération suivante.

Expansion, si $f\left(x_r^t\right) < f\left(x_0^t\right)$, calculer le *point d'expansion* :

$$x_e^t = \bar{x}^t + \chi\left(x_r^t - \bar{x}^t\right)$$

Si $f\left(x_e^t\right) < f\left(x_r^t\right)$ **alors** $x_n^t \leftarrow x_e^t$, itération suivante ;
sinon $x_n^t \leftarrow x_r^t$, itération suivante.

Contraction

Extérieure

Si $f\left(x_{n-1}^t\right) \leq f\left(x_r^t\right) < f\left(x_n^t\right)$, effectuer une *contraction extérieure* :

$$x_{oc}^t = \bar{x}^t + \gamma\left(x_r^t - \bar{x}^t\right)$$

Si $f\left(x_{oc}^t\right) \leq f\left(x_r^t\right)$ **alors** $x_n^t \leftarrow x_{oc}^t$, itération suivante ;
sinon aller à l'étape *Raccourci*r.

Intérieure

Si $f\left(x_{n-1}^t\right) \leq f\left(x_r^t\right) \geq f\left(x_n^t\right)$, effectuer une *contraction intérieure* ;

$$x_{ic}^t = \bar{x}^t + \gamma\left(x_n^t - \bar{x}^t\right)$$

Si $f\left(x_{ic}^t\right) \leq f\left(x_n^t\right)$ **alors** $x_n^t \leftarrow x_{ic}^t$, itération suivante ;
Sinon aller à l'étape *Raccourci*r.

Raccourcir le simplexe autour de x_0^t :

$$x_i^t \leftarrow \hat{x}_i^t = x_i^t + \frac{1}{2}\left(x_0^t - x_i^t\right), i = 1, \ldots, n$$

Fin

Algorithme 3.2 Algorithme du simplexe dynamique.

1. Le simplexe de départ à l'itération k est $S_0 = \left\{x_j\right\}_{j=1}^{N+1}$ et les valeurs de la fonction objectif sur ces sommets valent $\left\{g_j\right\}_{j=1}^{N+1}$,
2. Trier les sommets du simplexe afin que $g1 \geq g2 \geq \ldots \geq g_{N+1}$,
3. Réflexions successives. Une réflexion du plus mauvais point du simplexe S_0, i.e., x_1, est effectuée

$$x_{N+2} = \frac{2}{N} \sum_{j=2}^{N+1} x_j = x_1$$

et on évalue $g_{N+2} = g\left(x_{N+2}\right)$. Formant ainsi un nouveau simplexe $S_1 = \left\{x_j\right\}_{j=2}^{N+2}$. Réfléchir le premier (le plus mauvais) point de ce nouveau simplexe S_1, i.e., x_2,

$$x_{N+3} = \frac{2}{N} \sum_{j=3}^{N+1} x_j = x_2$$

et obtenir l'évaluation $g_{N+3} = g\left(x_{N+3}\right)$. Nous obtenons ainsi le second nouveau simplexe $S_2 = \left\{x_j\right\}_{j=3}^{N+3}$. Continuer à réfléchir le premier point du simplexe nouvellement formé. Après que la réflexion se soit répétée M fois, nous obtenons $S_M = \left\{x_j\right\}_{j=M+1}^{N+M+1}$ et une série de nouveaux simplexes $\left\{S_p\right\}_{p=1}^{M}$.
4. Choisir le simplexe de départ pour l'itération $k+1$. Calculer la valeur moyenne de la fonction à partir des sommets $\left\{S_p\right\}_{p=1}^{M}$

$$\bar{g}_{S_p} = \frac{1}{N+1} \sum_{j=p+1}^{N+p+1} g_j,\ p = 1, 2, \ldots, M.$$

Sélectionner le simplexe S_q satisfaisant $\bar{g}_{S_q} = \min\left\{\bar{g}_{S_p}\right\}_{p=1}^{M}$
5. Re-mesurer la valeur au point x_{N+1}.
6. Si le nombre maximum d'évaluations est atteint, stopper,
sinon initialiser S_q comme nouveau simplexe de départ, et retourner en 2.

Chapitre 4

Élaboration d'algorithmes à échantillonnage

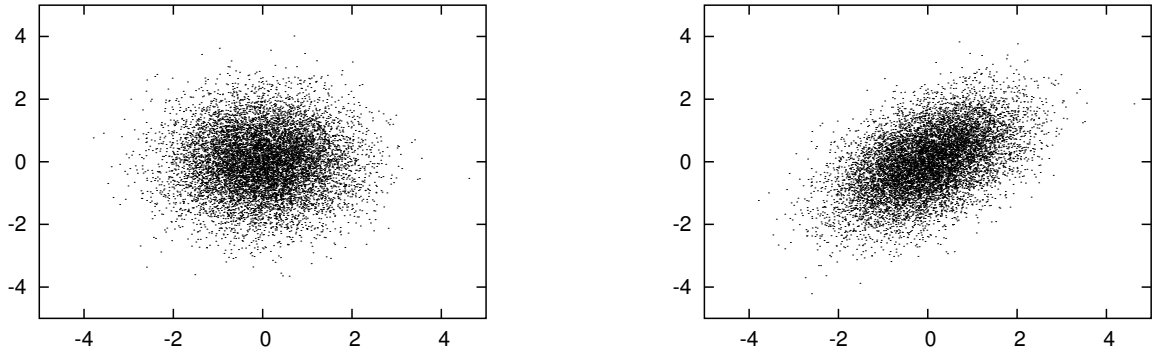
4.1 Introduction

Comme nous l'avons indiqué dans la section 2.4, les algorithmes de colonies de fourmis peuvent également être vus comme des méthodes à échantillonnage de distribution. De fait, il existe des cadres généraux dans lesquels ces algorithmes peuvent être intégrés, comme l'approche *IDEA* [Bosman and Thierens, 1999, Bosman and Thierens, 2000a] [Bosman and Thierens, 2000b] (voir section 2.4.4.5).

L'approche *IDEA* détermine le cadre général d'un algorithme à estimation de distribution. Elle décrit plusieurs étapes :

1. diversification : tirage aléatoire d'une population selon une distribution de probabilité donnée,
2. mémoire : choix d'une distribution de probabilité décrivant au mieux les données échantillonnées,
3. intensification : sélection des meilleurs points.

Quelques limites sont toutefois apportées, notamment seule la sélection est présentée comme méthode d'intensification. Dans le cadre de la *PMA*, nous avons donc hybridé un algorithme de type *IDEA* simple [Bosman and Thierens, 1999] avec une recherche locale de Nelder-Mead [Nelder and Mead, 1965], afin d'améliorer le comportement d'intensification. La méthode hybride est appelée *CHEDA* pour "Continuous Hybrid Estimation of Distribution Algorithm".



(a) somme de deux distributions normales mono-variantes

(b) distribution normale bi-variante

FIG. 4.1 – *Différence entre une somme de distributions normales mono-variantes et une distribution normale multi-variante. La seconde permet de prendre en compte une corrélation entre deux variables sous la forme de covariances. Les graphiques présentent la répartition en deux dimensions de* 10000 *points, avec un vecteur de moyenne nulle, une variance de* 1 *et une covariance de* $\frac{1}{2}$.

4.2 Algorithmes de base constitutifs de *CHEDA*

4.2.1 Algorithme de Nelder-Mead

L'algorithme de Nelder-Mead utilisé a été décrit en détail précédemment, dans la section 3.2.2.1.

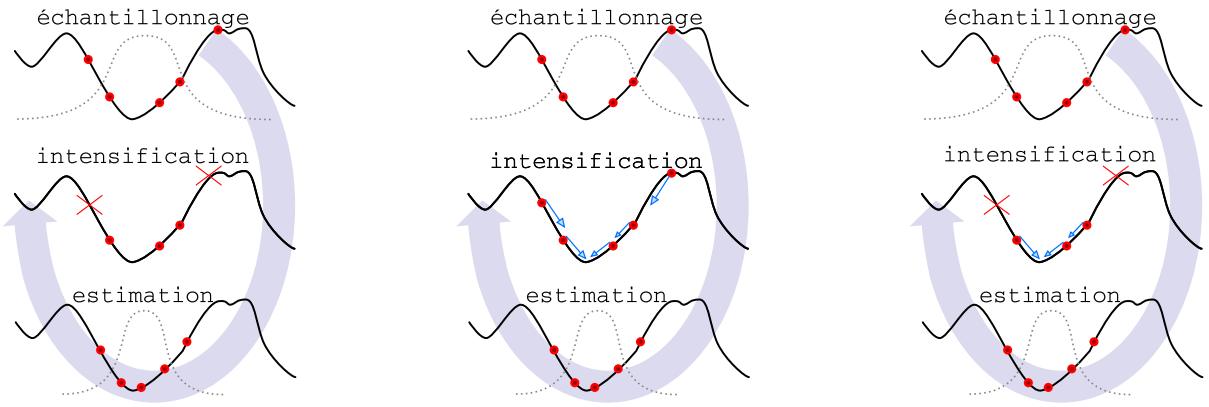
4.2.2 Algorithme à estimation de distribution

4.2.2.1 Distribution normale multi-variante

L'approche *IDEA* permet un certain nombre de variantes, nous avons choisi d'utiliser une distribution normale multi-variante comme base.

En effet, l'utilisation d'une somme de distributions normales mono-variantes ne permet pas de tenir compte des dépendances entre variables (comme le montre la figure 4.1), fréquentes dans les problèmes continus en ingénierie. De plus, ce modèle a l'avantage de permettre une manipulation aisée de la dispersion.

L'algorithme utilisé pour simuler la loi de probabilité normale multi-variante (ou loi multi-normale) est présenté dans l'annexe A.5, ainsi que l'estimation de la matrice de variance-covariance (dans l'annexe A.5.2.2). On notera par la suite $\mathcal{N}_{m,V}$ la distribution multi-normale ayant m pour vecteur de moyennes et V pour matrice de variance-covariance.



(a) $CHEDA_s$ (b) $CHEDA_nms$ (c) $CHEDA_s + nms$

FIG. 4.2 – *Variantes utilisées pour la méthode* CHEDA. *L'ensemble des trois étapes forme une itération de l'algorithme, itération répétée plusieurs fois au cours d'une optimisation.*

4.2.2.2 Principe de l'algorithme

La méthode choisie est présentée sur l'algorithme 4.1. La distribution utilisée est tronquée selon les bornes de l'espace de recherche.

Algorithme 4.1 Algorithme *EDA* utilisé.

Initialiser une population P_0 de π points dans $U_{x_{min},x_{max}}$

Jusqu'à critère d'arrêt :

> **Estimer** les paramètres m et V de l'échantillon P_{i-1}
>
> **Tirer** une population P_i de π points dans $\mathcal{N}_{m,V}$
>
> **Évaluer** P_i
>
> **Intensifier** à partir de P_i

Fin

4.3 Intensification : sélection et recherche locale

Nous avons comparé trois versions, différenciées d'après l'étape d'intensification : intensification par sélection, par recherche locale ou par sélection *et* recherche locale. La figure 4.2 présente les différences entre ces trois variantes.

4.3.1 $CHEDA_s$: sélection

Cette version est la plus proche des *EDA* classiques de l'approche *IDEA*. L'étape d'intensification consiste ici à sélectionner les $\beta * \pi$ meilleurs points de la population.

4.3.2 $CHEDA_n$: recherche locale

Dans cette variante, l'intensification est effectuée via une recherche locale de Nelder-Mead. Chaque point sert de base au lancement de la recherche, et la taille initiale du simplexe est donnée par l'écart moyen entre les points de la population. On autorise ν déformations du simplexe au maximum.

4.3.3 $CHEDA_{sn}$: sélection et recherche locale

Cet algorithme utilise une combinaison des deux intensifications précédentes : les meilleurs individus sont sélectionnés, et la recherche locale n'est alors lancée qu'à partir de cette population.

4.4 Réglage de *CHEDA*

L'algorithme *CHEDA* présente peu de paramètres, et puisque ceux-ci sont liés à une étape (diversification, intensification), il est relativement aisé de les régler.

Il existe trois paramètres à régler, selon les variantes :

π le nombre de points de l'échantillon ($CHEDA_s$, $CHEDA_n$ et $CHEDA_{sn}$),

β la proportion de points à sélectionner ($CHEDA_s$, $CHEDA_n$ et $CHEDA_{sn}$),

ν le nombre maximum de déformations du simplexe de Nelder-Mead ($CHEDA_n$ et $CHEDA_{sn}$).

Il est généralement nécessaire de maintenir un échantillonnage constant sur l'ensemble de la fonction objectif, afin d'effectuer correctement la diversification. Par exemple, si le problème a une seule dimension, et que l'on souhaite l'échantillonner à hauteur de 5 points, alors un problème similaire à 10 dimensions pourra être échantillonné à hauteur de 5^{10} points (ce qui, sur une répartition uniforme, permet en moyenne 5 points par dimension). Un minimum de 3^n points est en pratique nécessaire pour des problèmes de grandes dimensions à variables indépendantes. De fait, dans le cas d'applications réelles, plus le problème augmente en dimension, plus les chances de rencontrer des dépendances augmentent. On peut donc, en pratique, sur-échantillonner les problèmes à faible nombre de dimensions et sous-échantillonner les problèmes à grand nombre de dimensions. Une valeur de $\pi = 5^n$ est généralement un bon compromis.

L'intensification est réglée par β et ν. La proportion de points à sélectionner doit tenir compte de π ; en effet, si π est faible, il ne faut pas que β (donné en pourcentage) soit également faible, sinon il y a risque de mal estimer les paramètres de la distribution à cause d'un échantillon trop faible. À l'inverse, plus π est grand – par rapport au nombre de dimensions – plus β peut augmenter également. Par exemple si $\pi = 10^n$ (problème à n dimensions), il est possible d'aller jusqu'à $\beta = 1 - \frac{5^n}{\pi}$ maximum.

Le nombre ν associé à l'algorithme de Nelder-Mead doit, quant à lui, être fixé en fonction du nombre de dimensions. En effet, plus le nombre de dimensions augmente, plus le simplexe doit effectuer de mouvements pour progresser significativement. Si ν est trop élevé, le temps de calcul risque d'être prohibitif ; en effet, la méthode de Nelder-Mead consomme beaucoup d'évaluations de la fonction objectif. À l'inverse, si ν est trop faible, l'impact du simplexe sera très faible. En pratique, $\nu = 10n$ permet généralement d'atteindre de bons résultats.

4.5 Résultats expérimentaux

4.5.1 Comportement de convergence

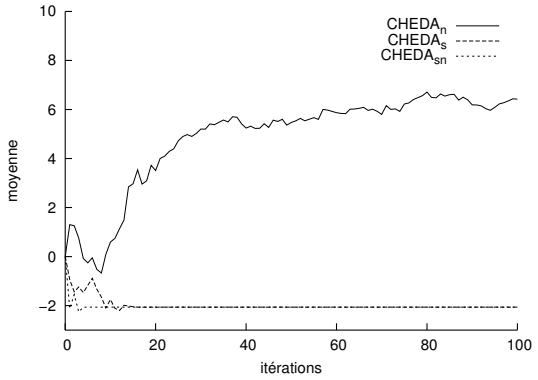
Pour illustrer le comportement de l'algorithme, nous avons choisi un réglage se voulant le plus simple possible, permettant la meilleure compréhension et non les meilleurs résultats. Les paramètres sont donc initialisés ainsi : $\pi = 100$, $\beta = 50\%$ et $\nu = 10$.

4.5.1.1 Dispersion

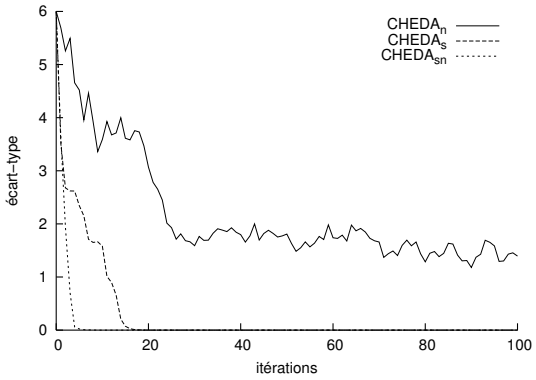
Afin de présenter le comportement de l'algorithme, nous avons utilisé des fonctions à une variable, facilitant notamment la présentation de la dispersion de la population de points. Les figures 4.3 et 4.4 présentent l'évolution des paramètres estimés de la distribution pour chaque itération, sur les problèmes D_1 (cf. annexe A.2.1) et Gr_1 (cf. annexe A.2.11).

On peut constater que la variante $CHEDA_n$ ne converge pas, en écart-type comme en moyenne. À l'inverse, l'algorithme $CHEDA_{sn}$ converge, de façon plus rapide que $CHEDA_s$.

De fait, la recherche locale de Nelder-Mead est rapidement piégée dans les optima locaux dans les régions de peu d'intérêt. Les différents simplexes lancés tendent à se regrouper dans les optima locaux, biaisant l'estimation des paramètres de la distribution et empêchant ainsi d'extraire l'information de convexité globale, qui permettrait la convergence vers l'optimum.

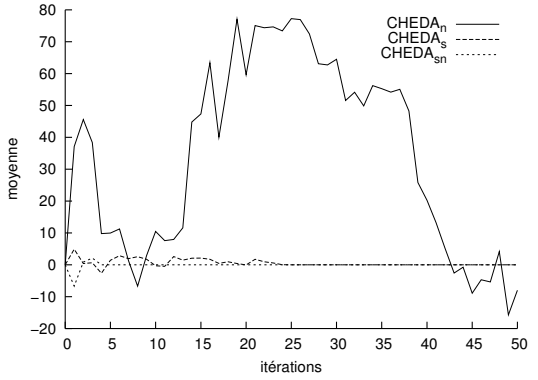


(a) moyennes

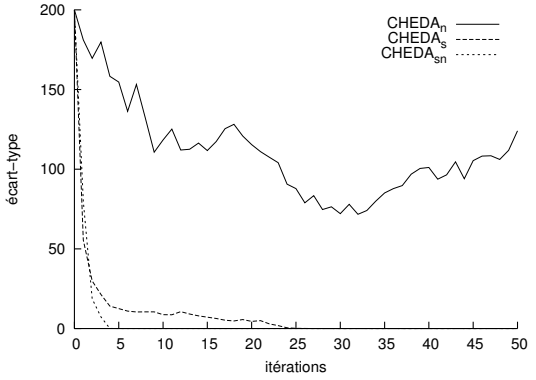


(b) écarts-types

FIG. 4.3 – *Évolution de la moyenne et de l'écart-type de la population pour chaque variante, sur le problème Dreo1.*

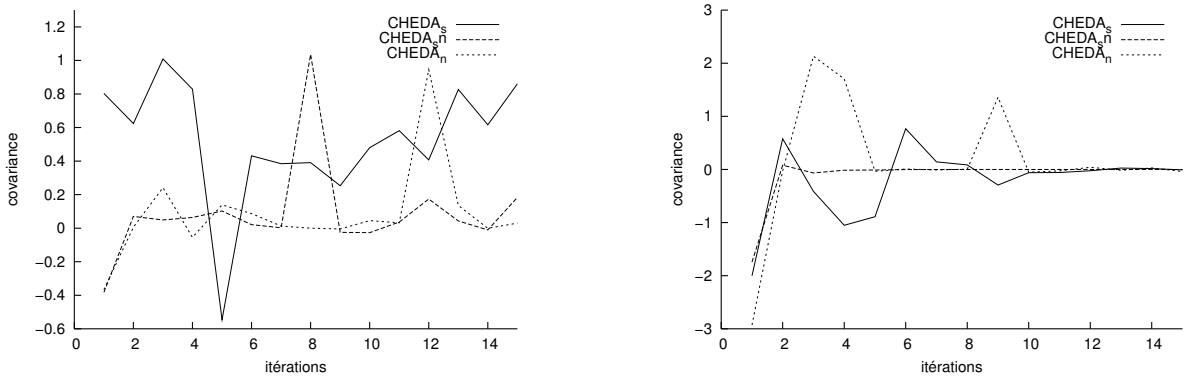


(a) moyennes



(b) écarts-types

FIG. 4.4 – *Évolution de la moyenne et de l'écart-type de la population pour chaque variante, sur le problème Griewangk (à une dimension).*



(a) variables dépendantes

(b) variables indépendantes

FIG. 4.5 – *Évolution de la covariance de la population pour chaque variante, sur les problèmes Rosenbrock (R_2) et B_2 (à deux dimensions).*

4.5.1.2 Covariance

Afin d'étudier comment la méthode $CHEDA$ utilise l'information de covariance, nous l'avons testée sur deux problèmes à deux dimensions (cf. figure 4.5), l'un ne montrant aucune corrélation entre les variables (B_2, cf. annexe A.2.2) et l'autre montrant au contraire une légère corrélation, variant selon les régions de l'espace de recherche (R_2, cf. annexe A.2.12).

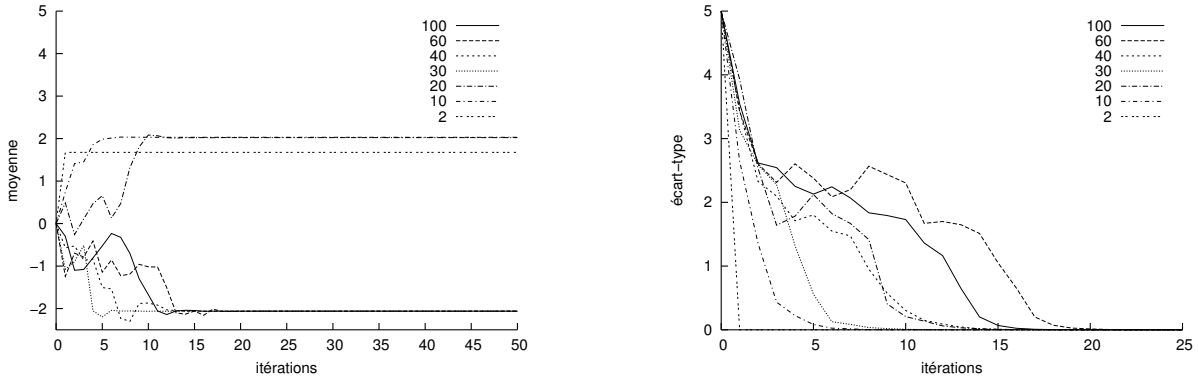
On peut constater que la covariance fluctue d'avantage sur le problème montrant une corrélation, alors qu'elle s'amortit sur le problème à variables indépendantes. Le problème Rosenbrock ne présente pas la même corrélation sur l'ensemble de l'espace de recherche, la valeur exacte de la covariance dépend donc de l'état de convergence de l'algorithme.

4.5.2 Influence des paramètres

4.5.2.1 Diversification, π

Le paramètre contrôlant la diversification est π , la taille de l'échantillonnage. La figure 4.6 présente l'évolution des paramètres estimés selon différents réglages de π.

Le premier graphique montre qu'un trop faible nombre de points entraîne une mauvaise estimation de la dispersion, et en conséquence, une convergence sur un mauvais optimum. L'évolution de l'écart-type montre en effet que la baisse de dispersion intervient trop vite dans ce cas, du fait d'une sous-estimation de la variance.



(a) moyennes

(b) écarts-types

FIG. 4.6 – *Évolution de la moyenne et de l'écart-type de la population dans $CHEDA_s$, pour différentes valeurs de π, sur le problème Dreo1.*

4.5.2.2 Intensification, β

Un des deux paramètres contrôlant l'intensification est β , le taux de sélection des meilleurs points ; plus β est petit, moins le nombre de points gardés pour estimer les paramètres de la distribution est faible. La figure 4.7 présente l'évolution des paramètres estimés selon différents réglages de β.

Le rôle de β est relativement simple et concerne principalement la vitesse de la convergence. Plus β est grand, plus la convergence est lente.

4.5.2.3 Intensification, ν

Le second paramètre contrôlant l'intensification est ν , le nombre d'évaluations accordé à l'algorithme de Nelder-Mead. Dans l'exemple suivant, la plus grande valeur de ν vaut 10 car, au delà, la recherche locale devient trop précise pour être maintenue, elle est donc stoppée.

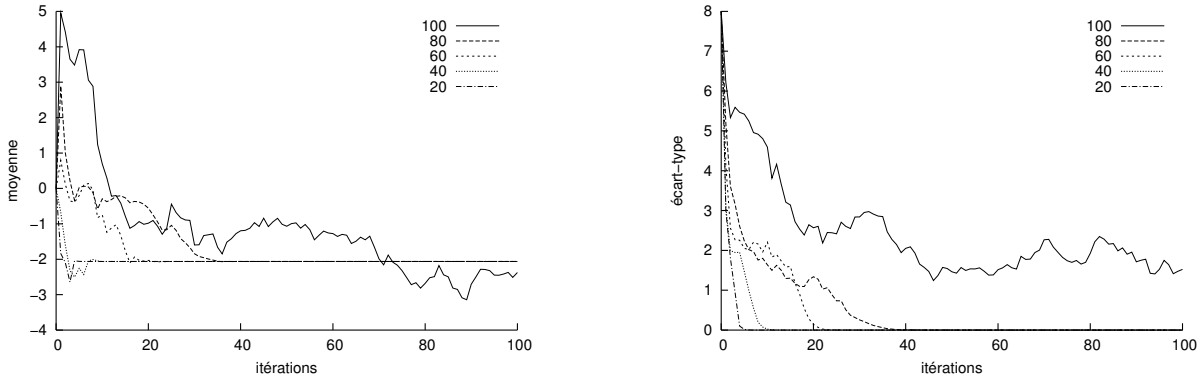
On peut constater que plus ν est grand, plus la convergence est rapide.

4.5.3 Résultats

4.5.3.1 Comparaisons avec des algorithmes à estimation de distribution

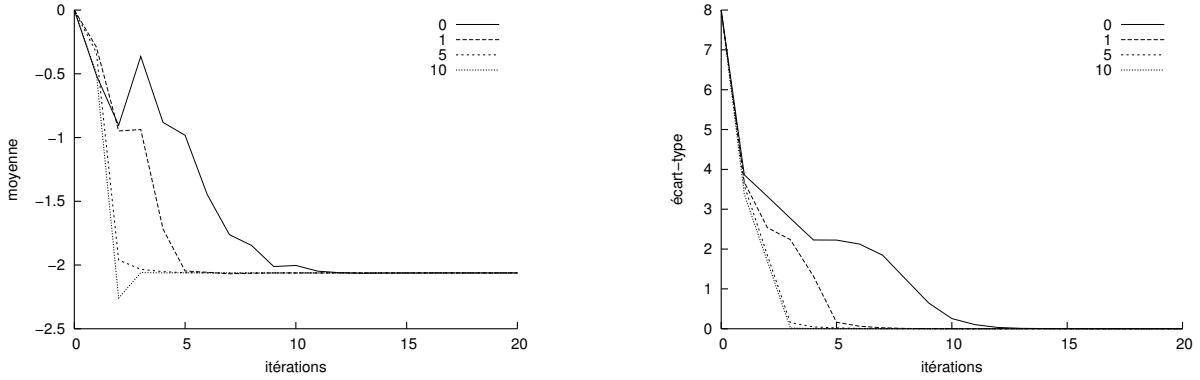
Nous avons comparé $CHEDA_s$ et $CHEDA_{sn}$ avec quelques algorithmes à estimation de distribution pour des problèmes continus. Les expériences sont lancées en suivant le même protocole que [Bengoetxea et al., 2002], d'où les résultats sont tirés :

— 10 lancements différents par test,



(a) moyennes (b) écarts-types

FIG. 4.7 – *Évolution de la moyenne et de l'écart-type de la population dans $CHEDA_s$, pour différentes valeurs de β, sur le problème Dreo1.*



(a) moyennes (b) écarts-types

FIG. 4.8 – *Évolution de la moyenne et de l'écart-type de la population dans $CHEDA_{sn}$, pour différentes valeurs de ν ; sur le problème Dreo1.*

TAB. 4.1 – *Comparaison de $CHEDA$ avec d'autres algorithmes à estimation de distribution, sur le problème* Sphere *à 10 dimensions (SM_{10}) .*

Algorithme	erreur	évaluations
$UMDA_c$	6.7360e-6 ± 1.26e-6	74163.9± 1750.3
$MIMIC_c$	7.2681e-6 ± 2.05e-6	74963.5 ± 1053.5
$EGNA_{BIC}$	2.5913e-5 ± 3.71e-5	77162.4 ± 6335.4
$EGNA_{BGe}$	7.1938e-6 ± 1.78e-6	74763.6 ± 1032.2
$EGNA_{ee}$	7.3713e-6 ± 1.98e-6	7396.4± 163201
$EMNA_{global}$	7.3350e-6 ± 2.24e-6	94353.8 ± 842.8
$EMNA_a$	4.8107e4 ± 1.32e4	301000 ± 0.0
$CHEDA_s$	**6.6493e-7 ± 1.82e-7**	**124301.6 ± 1027.2**
$CHEDA_{sn}$	**8.2566e-7 ± 1.31e-7**	**103420.3 ± 821**

TAB. 4.2 – *Comparaison de $CHEDA$ avec d'autres algorithmes à estimation de distribution, sur le problème* Griewangk *à 10 dimensions (Gr_{10}).*

Algorithme	erreur	évaluations
$UMDA_c$	6.0783e-2 ± 1.93e-2	301850± 0.0
$MIMIC_c$	7.3994e-2 ± 2.86e-2	301850± 0.0
$EGNA_{BIC}$	3.9271e-2 ± 2.43e-2	301850± 0.0
$EGNA_{BGe}$	7.6389e-2 ± 2.93e-2	301850± 0.0
$EGNA_{ee}$	5.6840e-2 ± 3.82e-2	301850± 0.0
$EMNA_{global}$	5.1166e-2 ± 1.67e-2	301850± 0.0
$EMNA_a$	12.9407± 3.43	301850± 0.0
$CHEDA_s$	**6.1664e-7± 1.97**	**123639.6 ± 1258.9**
$CHEDA_{sn}$	**3.7579e-7 ± 0.84**	**105769.2± 945.6**

— nombre de points de l'échantillonnage : $\pi = 2000$,
— sélection de la moitié des points : $\beta = 50\%$,
— critère d'arrêt :
  – si le nombre d'évaluations atteint 301850 ; ou
  – si l'erreur est inférieure à $1e-6$;

Les résultats de ces tests sont présentés dans les tableaux 4.1, 4.2 et 4.3.

Pour le problème SM_{10} , les deux variantes de $CHEDA$ montrent sensiblement les mêmes résultats. La précision atteinte est meilleure d'un facteur 10 que celle des autres algorithmes, mais la vitesse de convergence est une fois et demie plus faible que celle des meilleurs algorithmes.

Pour le problème Gr_{10}, les résultats atteints sont meilleurs que ceux des autres algorithmes, tant du point de vue précision que vitesse. En ordre de grandeur, $CHEDA$ est environ 2 fois plus rapide et 5 fois plus précis. La bonne précision doit être tempérée par

TAB. 4.3 – *Comparaison de $CHEDA$ avec d'autres algorithmes à estimation de distribution, sur le problème* Rosenbrock *à 10 dimensions (R_{10}).*

Algorithme	erreur	évaluations
$UMDA_c$	8.7204 ± 3.82e2	301850 ± 0.0
$MIMIC_c$	8.7141 ± 1.64e-2	301850 ± 0.0
$EGNA_{BIC}$	8.8217 ± 0.16	268066.9 ± 69557.3
$EGNA_{BGe}$	8.6807 ± 5.87e2	164518.7 ± 24374.5
$EGNA_{ee}$	8.7366 ± 2.23e-2	301850 ± 0.0
$EMNA_{global}$	8.7201± 4.33e-2	289056.4 ± 40456.9
$EMNA_a$	3263.0010 ± 1216.75	301000 ± 0.0
$CHEDA_s$	**8.9980 ± 7.79e-2**	**301304.4 ± 45.7**
$CHEDA_{sn}$	**8.5490 ± 1.27e-1**	**302400.1 ± 33.4**

le fait que l'écart-type est plus grand, ce qui signifie que la robustesse de l'algorithme est plus faible que celle de ses concurrents. De plus, la version $CHEDA_{sn}$ est plus précise que $CHEDA_s$.

Sur le problème R_{10}, la précision atteinte est comparable à celle de ses meilleurs concurrents, tout comme la rapidité. La variante avec recherche de Nelder-Mead est légèrement plus précise, mais moins robuste.

4.5.3.2 Comparaison avec d'autres métaheuristiques

Nous avons également comparé $CHEDA$ avec d'autres métaheuristiques : notamment des algorithmes de colonies de fourmis (*CACO* [Bilchev and Parmee, 1995, Mathur et al., 2000], *API* [Monmarché et al., 2000b], *CIAC* et *HCIAC*) des algorithmes évolutionnaires (*ES* [Storn and Price, 1995] et *CGA* [Chelouah and Siarry, 2000a, Chelouah and Siarry, 2000b]) et une recherche avec tabou (*ECTS* [Chelouah and Siarry, 2000b]). Nous avons utilisé un réglage proche de celui utilisé pour *CIAC* et *HCIAC* :

— 50000 évaluations maximum,
— $\pi = 100$ points par échantillonnage,
— sélection de la moitié des points : $\beta = 50$,
— $\nu = 20$ itérations maximum pour la recherche de Nelder-Mead,
— arrêt si $\theta = 10000$ évaluations sans amélioration.

Les variantes de $CHEDA$ sont, d'une manière générale, plus rapides que leurs homologues, la version $CHEDA_{sn}$ présentant également une plus grande précision. À précision comparable avec $HCIAC$, $CHEDA_{sn}$ a la plus grande rapidité. Les résultats sur le problème Gr montrent une tendance inverse de ceux observés pour les algorithmes $CIAC$, l'augmentation du nombre de dimensions entraînant les meilleurs résultats pour $CHEDA$

TAB. 4.4 – *Résultats de* CHEDA *et d'algorithmes de colonies de fourmis concurrents.*

	CACO			API		
Fonction	% ok	evals	err	% ok	evals	err
R_2	100	6842	0.00		[10000]	0.00 (0.00)
SM	100	22050			[10000]	0.00 (0.00)
Gr_5					[10000]	0.18 (0.04)
Gr_{10}	100	50000	0.0			
GP	100	5330				
MG	100	1688				
St		[6000]	0.0			

	CIAC			HCIAC		
Fonction	% ok	evals	err	% ok	evals	err
R_2	100	11797	3e-3(3e-3)	100	18747(6697)	1e-8(4e-9)
SM	100	50000	9e-10(1e-11)	100	18616(6715)	5e-8(1e-8)
Gr_5	63	48402	0.01(9e-3)	75	10870(1347)	1e-4(2e-4)
Gr_{10}	52	50121	0.05(0.05)	18	23206(13132)	1e-3(4e-4)
GP	56	23391	1.51(2.33)	100	34533(4086)	0(0)
MG	20	11751	0.34(0.19)	100	24596(11413)	4e-9(4e-9)
St	94	28201	1.96(1.61)	100	10726(219)	0(0)

	$CHEDA_s$			$CHEDA_{sn}$		
Fonction	% ok	evals	err	% ok	evals	err
R_2	26	3026.7(453.8)	0.86(0.31)	33	3016.9(511.3)	0.11(0.23)
SM	100	6567.2(2478)	8.92e-5(4.4e-4)	100	6972(2940)	1.26e-8(1.02e-7)
Gr_5	75	62691.1(16.9)	7.14e-2(1.93e-2)	90	62679.2(17.5)	1.23e-02(4.05e-03)
Gr_{10}	35	6567.2(2478)	8.93e-5(4.39e-4)	86	6972(2940)	1.26e-8(1.02e-7)
GP	94	3634.6(901.5)	0.14(0.51)	100	2998.7(57.2)	0(0)
MG	22	3241.9(1448.7)	0.31(0.42)	100	5867.4(1334.1)	5.09e-10(1.49e-9)
St	96	80889.2(673.9)	1.4(0.96e-1)	97	74301(597.9)	1.34(1.13)

TAB. 4.5 – *Résultats de* CHEDA *et de quatre métaheuristiques d'optimisation continue concurrentes.*

	CGA			ECTS			DE		
Fonction	% ok	evals	err	% ok	evals	err	% ok	evals	err
St							100	1300	
Gr_{10}							100	12804	
SM	100	750	0.0002	100	338	3e-8	100	392	
R_2	100	960	0.004	100	480		100	615	
GP	100	410	0.001	100	231	2e-3			
$S_{4,5}$	76	610	0.14	75	825	0.01			

	CIAC			HCIAC		
Fonction	% ok	evals	err	% ok	evals	err
St	94	8201	1.96(1.61)	100	10726(219)	0(0)
Gr_{10}	52	50121	0.05(0.05)	18	23206(13132)	1e-3(4e-4)
SM	100	50000	9e-10(1e-11)	100	18616(6715)	5e-8(1e-8)
R_2	100	11797	3e-3(3e-3)	100	18747(6697)	1e-8(4e-9)
GP	56	23391	1.51(2.33)	100	34533(4086)	0(0)
$S_{4,5}$	5	39311	6.34(1.01)	100	17761(5976)	0(0)

	$CHEDA_s$			$CHEDA_{sn}$		
Fonction	% ok	evals	err	% ok	evals	err
St	96	80889.2(673.9)	1.4(0.96e-1)	97	74301(597.9)	1.34(1.13)
Gr_{10}	35	6567.2(2478)	8.93e-5(4.39e-4)	86	6972(2940)	1.26e-8(1.02e-7)
SM	100	6567.2(2478)	8.92e-5(4.4e-4)	100	6972(2940)	1.26e-8(1.02e-7)
R_2	26	3026.7(453.8)	0.86(0.31)	33	3016.9(511.3)	0.11(0.23)
GP	94	3634.6(901.5)	0.14(0.51)	100	2998.7(57.2)	0(0)
$S_{4,5}$	17	4188.3(1732.3)	3.32(2.84)	100	3769.5(169.7)	0.33(8.93e-15)

sur Gr_{10}. Sur les fonctions GP et MG, l'algorithme $CHEDA_{sn}$ présente les meilleurs résultats parmi tous les algorithmes, notamment en vitesse. Enfin le problème St semble poser des problèmes de rapidité aux deux variantes, par rapport aux autres algorithmes.

Comparé aux autres métaheuristiques présentées, $CHEDA$ est généralement plus lent, mais plus précis. Ainsi, sur la fonction St, le nombre d'évaluations nécessaires est bien plus élevé. À l'inverse, la fonction Gr_{10} est optimisée plus rapidement par $CHEDA$ que par ses concurrents. Sur le problème SM, à précision comparable, la méthode $ECTS$ est plus rapide que $CHEDA_{sn}$. Sur la fonction GP, $CHEDA_{sn}$ a la meilleure précision, mais au prix d'un plus grand nombre d'évaluations que CGA ou $ECTS$. Les algorithmes $CHEDA$ sont plus rapides que les variantes de $CIAC$ sur $S_{4,5}$, mais restent néanmoins plus lents que les autres métaheuristiques.

4.6 Discussion

Les différents résultats présentés nous permettent de cerner le comportement et l'intérêt de la méthode $CHEDA$. Celle-ci est en effet plus adaptée aux problèmes globalement convexes, le nombre de dimensions ne semblant pas avoir un effet néfaste sur l'efficacité de l'optimisation.

L'hybridation avec la recherche de Nelder-Mead apporte un gain en précision, mais doit être utilisée en combinaison avec une autre méthode d'intensification, afin d'éviter que l'algorithme ne perde la structure de la distribution lors de l'estimation. En effet, la méthode de Nelder-Mead peut très rapidement converger sur les optima locaux, ce qui tend à provoquer la perte de l'information de dispersion, si beaucoup de simplexes se rejoignent dans des optima proches.

La méthode $CHEDA$, en utilisant une distribution normale, permet d'extraire facilement l'information de convexité du problème, ainsi que l'information de corrélation entre variables. Cependant, dans le cas de problèmes où la position de l'optimum n'est pas indiquée par la convexité globale, le biais induit par cette distribution est problématique.

D'une manière générale, le choix de $CHEDA_{sn}$ par rapport à $CHEDA_s$ doit se faire sur des critères de précision, $CHEDA_{sn}$ permettant des erreurs plus faibles. Ces deux variantes sont par ailleurs sensiblement équivalentes en terme de nombre d'évaluations nécessaire, $CHEDA_{sn}$ nécessite cependant plus de temps de calcul, du fait de l'utilisation de la recherche de Nelder-Mead.

La métaheuristique $CHEDA$ reste néanmoins gourmande en terme d'évaluations, du fait de la nécessité d'échantillonner correctement la fonction objectif. Le nombre d'évaluations croît donc avec le nombre de dimensions, si l'on veut parcourir de manière équivalente toutes les variables sur un problème sans dépendances.

4.7 Conclusion

Nous avons montré qu'il était simple d'utiliser l'estimation de distribution comme méthode de conception d'algorithme de colonies de fourmis. La méthode proposée utilise une distribution normale multivariante et est hybridée avec une recherche locale de Nelder-Mead.

Nos résultats montrent que cette hybridation permet une amélioration de la précision des résultats. De même, les algorithmes ainsi conçus possèdent des qualités supérieures aux variantes de $CIAC$, notamment en terme de rapidité et de simplicité de manipulation.

Les défauts généralement présentés comme étant propres aux algorithmes de colonies de fourmis sont également présents, le coût en nombre d'évaluations restant en effet plus élevé que dans d'autres métaheuristiques. L'approche par estimation de distribution perd

également les bénéfices apportés par la flexibilité d'une méthode centrée sur la communication, mais gagne beaucoup en terme de simplicité de conception et de manipulation.

Chapitre 5

Application au recalage d'images d'angiographie rétinienne

5.1 Recalage rigide

5.1.1 Introduction

Le recalage est un outil important pour résoudre plusieurs problèmes d'analyse d'images médicales. Beaucoup de stratégies de minimisation classiques ont été appliquées au problème de recalage d'images, comme la recherche exhaustive, la descente de gradient, la méthode du simplexe, le recuit simulé, les algorithmes génétiques et la méthode de Powell [Ritter et al., 1999, Jenkinson and Smith, 2001].

Dans la plupart des cas, le recalage est effectué en deux étapes : traitement de l'image et optimisation d'un critère de similarité. Le traitement de l'image vise à améliorer la qualité de l'image et à extraire l'information pertinente permettant d'effectuer au mieux l'étape d'optimisation. Celle-ci a pour but de trouver les déformations optimales, conformément à une fonction objectif décrivant la qualité du recalage. L'étape d'optimisation est souvent réalisée via des méthodes exactes, qui ne sont généralement utilisables que pour une recherche locale, et qui peuvent ne pas trouver l'optimum global.

Puisque le traitement de l'image et le calcul de la fonction objectif sont coûteux en temps de calcul, les méthodes d'optimisation globale comme les métaheuristiques (qui requièrent un nombre important d'évaluations) sont souvent écartées au profit de méthodes d'optimisation locale, qui sont souvent plus rapides pour trouver un optimum. Mais si l'on considère des problèmes complexes mettant en jeu des optima locaux, l'utilisation de méthodes d'optimisation globale peut s'avérer fructueuse [Jenkinson and Smith, 2001].

La combinaison d'images temporelles ou de modalités différentes est fréquemment utilisée pour aider les médecins dans leur diagnostic. Durant l'acquisition d'une séquence

d'images d'angiographie rétinienne, il y a inévitablement des mouvements de l'oeil et il est donc essentiel de corriger ce phénomène, avant toute analyse quantitative. Les méthodes de recalage permettent donc le calcul d'un champ de déformations qui reflètent la transformation des structures présentes d'une image à l'autre. Le but du présent travail est d'améliorer la qualité des analyses ophtalmologiques, en améliorant le recalage d'images d'angiographie rétinienne.

Ce travail présente le recalage d'angiographies rétiniennes à l'aide d'une métaheuristique. Le chapitre est composé de cinq parties supplémentaires. Il examine tout d'abord le problème du recalage et du traitement d'image utilisé et introduit ensuite les outils d'optimisation choisis. Les résultats sont présentés dans la section 5.1.4, et une discussion est présentée dans la section 5.1.5. Enfin, la conclusion termine le chapitre.

5.1.2 Recalage d'angiographies rétiniennes

5.1.2.1 Méthodes existantes

Des expériences ont montré qu'une méthode proposée précédemment dans [Nunes et al., 2004a], fondée sur une recherche locale, n'était pas suffisamment robuste dans le cas de fortes variations inter-images. Dans la majorité des travaux publiés sur le recalage automatique d'images rétiniennes [Berger et al., 1999, Can and Stewart, 1999, Hampson and Pesquet, 2000, Mukhopadhyay and Chanda, 2001, Pinz et al., 1998] [Ritter et al., 1999, Simo and de Ves, 2001, Zana and Klein, 1999], une étape préalable de détection des structures vasculaires est nécessaire. Cependant, dans la phase tardive de l'angiographie (quand le produit de contraste disparaît), l'image est caractérisée par un contraste local faible. La méthode proposée peut pallier les problèmes rencontrés pour la détection des structures vasculaires.

Images d'angiographies rétiniennes Normalement, une série d'images du fond de l'oeil est obtenue par angiographie à la fluorescéine et/ou à l'ICG (*indo-cyanine green*). Ces deux techniques sont utiles pour évaluer la circulation rétinienne et choroïdale, ainsi que dans le diagnostic et le traitement de plusieurs maladies de la rétine, comme la dégénérescence maculaire liée à l'âge (DMLA), les rétinites à cyto-megalo-virus (RCMV) et la rétinopathie diabétique (RD) [Richard et al., 1998].

Cette technique consiste en une injection d'un produit de contraste, la fluorescéine ou le vert d'indo-cyanine dans la veine cubitale (dans le bras), suivie de l'observation de sa distribution à certains moments sur les vaisseaux rétiniens. Une angiographie rétinienne (jusqu'à 36 images) [Richard et al., 1998] est généralement divisée en trois phases : précoce,

moyenne et tardive. Elle permet ainsi de visualiser l'arbre vasculaire ou les structures choroïdales, et donne la possibilité de détecter des pathologies existantes.

Le recalage est nécessaire afin de détecter et de quantifier les pathologies rétiniennes sur les zones d'intérêt (fovéa, nerf optique et structures vasculaires). Ces zones d'intérêt sont celles que l'opthalmologiste va analyser.

Les vaisseaux sont la seule structure visible significative [Richard et al., 1998], dans toutes les images utilisées dans le recalage d'angiographies. Cependant, des variations locales d'intensité peuvent donner lieu à plusieurs difficultés, notamment à un fond non uniforme de l'image, un contraste local faible, des mouvements oculaires, plusieurs types de bruits, ainsi que la présence de vaisseaux sanguins ayant des variations de luminances par rapport au fond de l'image.

La figure 5.1 montre deux exemples de trois images successives, prises par angiographie à la fluorescéine et à l'ICG. Comme les images sont très bruitées, la procédure de recalage requiert une étape de filtrage de l'arbre vasculaire.

Prétraitement Les images originales présentent un certain nombre de caractéristiques pouvant géner le recalage, il est donc nécessaire d'effectuer un prétraitement, afin d'isoler les informations pertinentes.

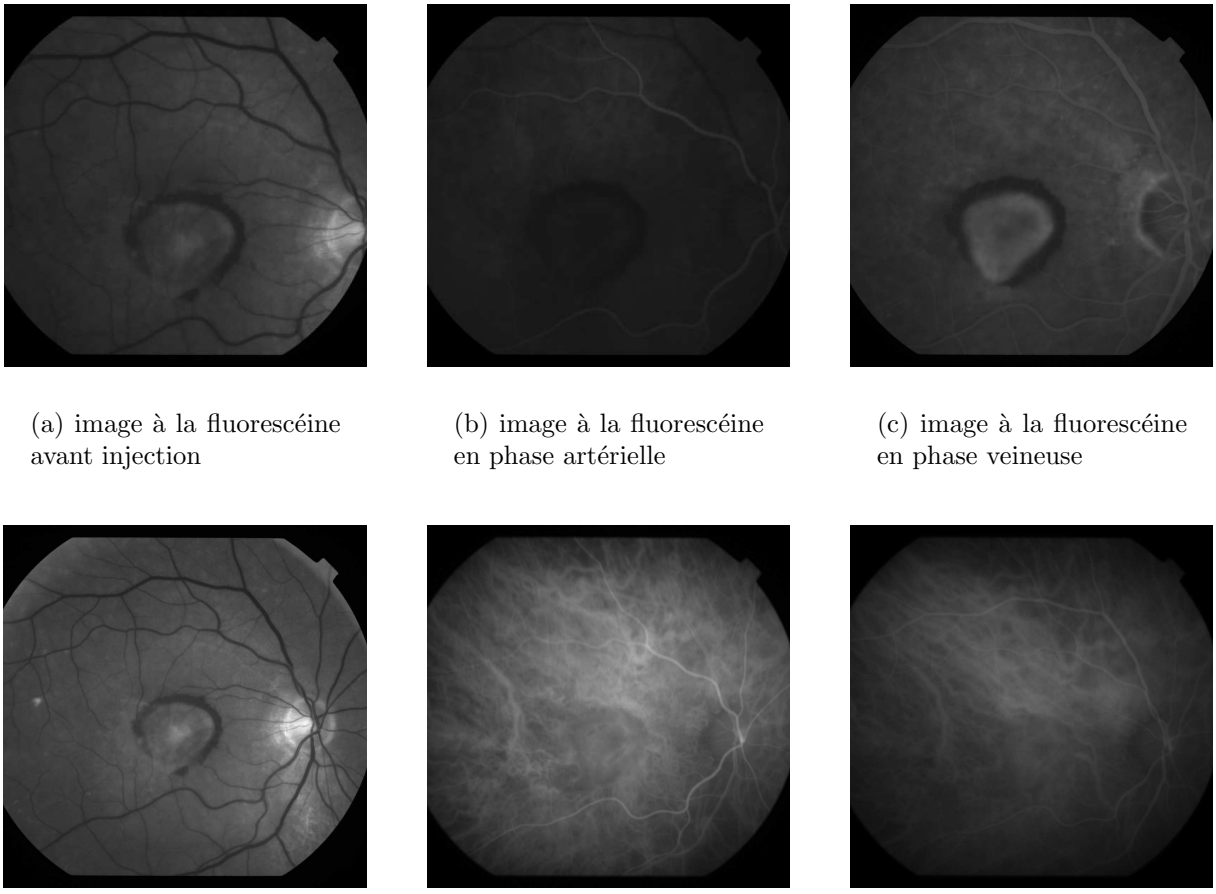
L'étape de filtrage (ou réduction de bruit), permet un recalage plus robuste des images. Ce filtre est fondé sur des statistiques locales, estimées à partir d'un voisinage de 5×5 pixels pour chaque pixel. Il a l'avantage de préserver les bords de la structure vasculaire.

Durant l'angiographie, il y a également des variations d'intensité, notamment sur le fond de l'image. Le calcul du gradient morphologique des images devient alors nécessaire pour pallier ce problème. Le gradient morphologique d'une image est défini comme la différence entre l'image dilatée et l'image érodée.

Un filtrage supplémentaire peut être employé pour lisser l'image gradient. Cette étape dite "filtrage médian" remplace la valeur de chaque pixel par la valeur médiane des pixels adjacents, dans un voisinage de 3×3.

On trouvera les détails sur les filtres utilisés dans [Nunes, 2003] et [Nunes et al., 2004b], la figure 5.2 illustre le prétraitement pratiqué.

Méthodes de recalage classiques La plupart des stratégies classiques ont été utilisées dans l'analyse de mouvement et les problèmes de recalage [Bangham et al., 1996, Hart and Goldbaum, 1994, Irani et al., 1994, Kim et al., 2001, Odobez and Bouthemy, 1994, Zhang and Blum, 2001]. Une approche commune à la plupart des techniques est l'application de méthodes multi-résolutions [Odobez and Bouthemy, 1994, Zhang and Blum, 2001] fondées sur le flot optique, où la recherche est effectuée à des résolutions croissantes. Dans le domaine du recalage d'images rétiniennes, l'extraction d'éléments caractéristiques de



(a) image à la fluorescéine avant injection

(b) image à la fluorescéine en phase artérielle

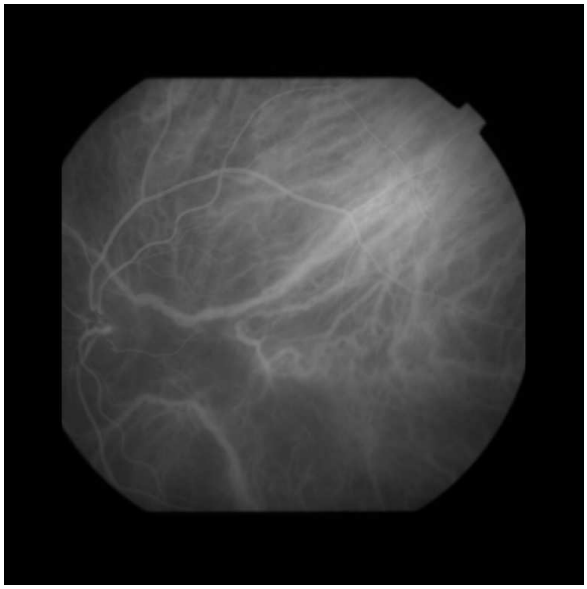
(c) image à la fluorescéine en phase veineuse

(d) image à l'ICG avant injection

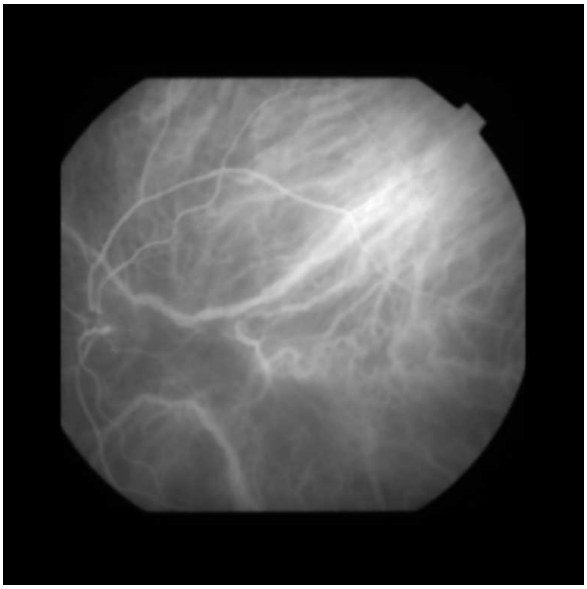
(e) image à l'ICG en phase artérielle

(f) image à l'ICG en phase veineuse

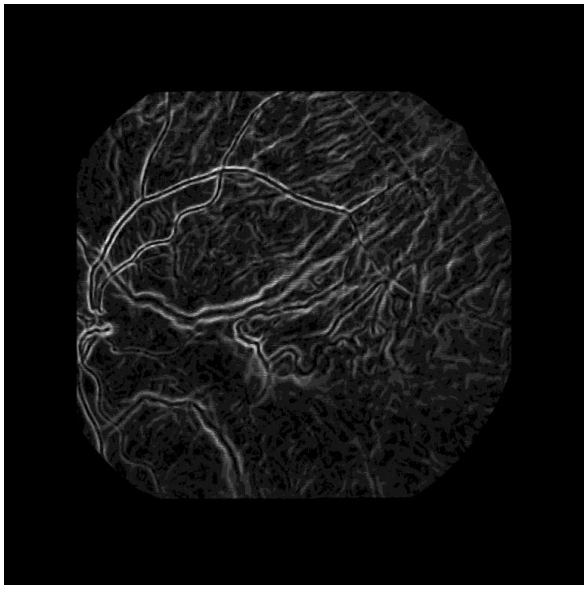
FIG. 5.1 – *Images successives d'angiographies à la fluorescéine (rangée supérieure) et à l'ICG (rangée inférieure) du fond de l'oeil.*



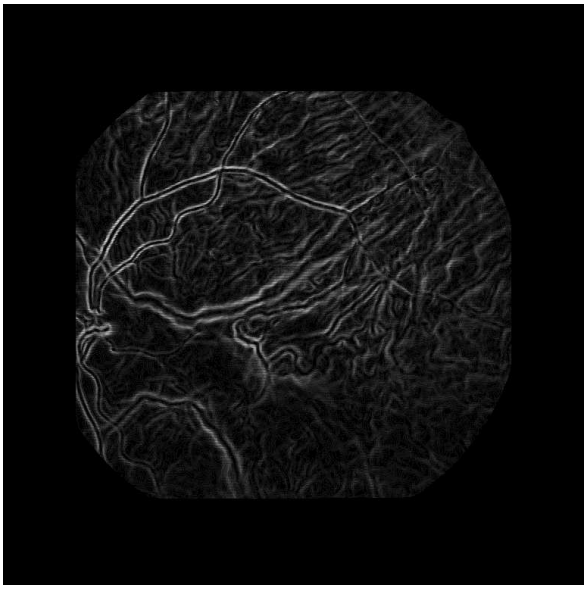
(a) image originale



(b) image filtrée



(c) image gradient



(d) image gradient filtrée

FIG. 5.2 – *Étapes de prétraitement avant le recalage.*

l'image est généralement employée [Berger et al., 1999], mais la présence de contrastes locaux faibles et de bruit rend la détection de la structure vasculaire difficile. Le cas du recalage d'images d'angiographies à l'ICG n'a de plus été traité que récemment, par une technique fondée sur le *flot optique* (qui calcule le champ de déplacement entre deux images [Nunes et al., 2004a]).

Un état de l'art de plusieurs algorithmes de recalage peut être trouvé dans [Brown, 1992, Maintz and Viergerver, 1998, Hill et al., 2001]

5.1.2.2 Méthode proposée pour le recalage

Dans ce travail, nous étudierons uniquement la transformation de translation, car c'est à la fois la plus significative et la plus difficile à obtenir dans le cas des images d'angiographie rétinienne. Le recalage consistera donc à obtenir le meilleur déplacement en ligne et en colonne entre deux images de la séquence d'angiographie.

Algorithme L'algorithme proposé comprend les étapes suivantes :

- filtrage de Wiener des images originales,
- calcul du gradient morphologique,
- filtrage médian (étape optionnelle),
- optimisation,
  - calcul du critère de similarité (somme des différences d'intensité) entre deux images pour la transformation courante ;
  - recherche d'une meilleure solution.

Critère de similarité Ces dernières années, des techniques fondées sur l'intensité des images sans détection préalable (techniques dites "iconiques") ont été appliquées à un grand nombre de problèmes de recalage. Leur principe de base est de maximiser un critère mesurant la similarité d'intensité des pixels superposés.

La mesure de similarité est le calcul utilisé pour juger de la qualité du recalage entre deux images. Dans [Roche et al., 1999], Roche *et al.* ont démontré quelles sont les hypothèses correspondant à un grand nombre de mesures de similarité, afin de comprendre leurs utilisations, et de permettre de choisir quelle méthode est la plus appropriée à une classe de problèmes donnée. La somme du carré des différences (SDS) [Brown, 1992], somme des valeurs absolues des différences (SAD) [Yu et al., 1989], l'intercorrélation [Cideciyan et al., 1992], l'entropie, etc. sont faciles à calculer, et offrent souvent des problèmes de minimisation relativement simples. Beaucoup d'algorithmes emploient la somme du carré des différences entre un couple d'images comme mesure de similarité,

elle est calculée comme suit :

$$\text{Précision} = \frac{\sum_{(i,j)=(1,1)}^{(\text{longueur,largeur})} |I_1(i,j) - I_2(i,j)|}{(\text{longueur} \cdot \text{largeur})} \tag{5.1}$$

où I_1et I_2 sont les deux images à recaler.

Toutes ces mesures classiques ne sont pas équivalentes en termes de robustesse et de précision, mais aucune d'entre elles n'est vraiment capable de faire face aux changements d'intensité relative d'une image à l'autre. Un des défauts de ces mesures est que les images qui seraient qualitativement considérées comme étant bien recalées peuvent toujours présenter des erreurs d'alignement [Roche et al., 1999], du fait de ces changements d'intensité. Cependant, dans le cadre de cette étude, la somme du carré des différences est une métrique valable, puisqu'elle est calculée à partir des images gradients filtrées.

5.1.3 Optimisation

Dans le problème qui nous occupe, la fonction objectif doit décrire au mieux la qualité d'un recalage donné, nous avons donc choisi une fonction de similarité. L'espace de recherche à parcourir est l'ensemble des mouvements possibles de recalage.

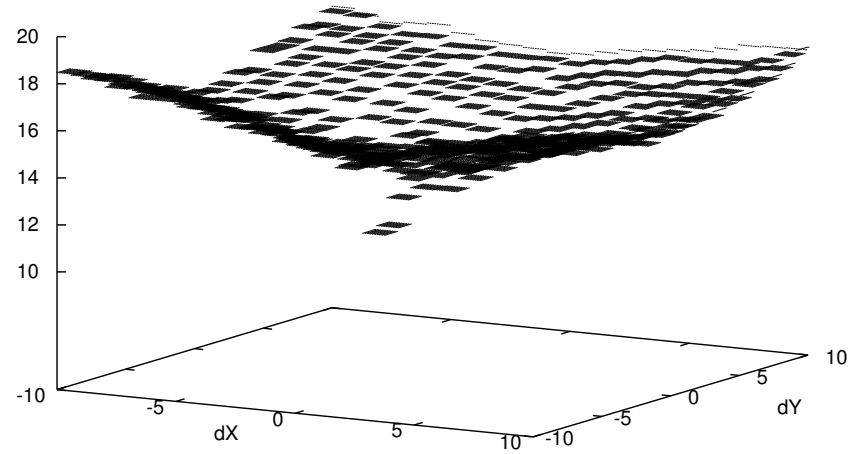
Il est potentiellement possible d'employer plusieurs techniques d'optimisation ; dans ce travail, nous avons testé deux méthodes : une recherche locale de Nelder-Mead (*Nelder-Mead Search*, *NMS* [Nelder and Mead, 1965]) et nos algorithmes *HCIAC* et *CHEDA*.

5.1.3.1 Fonction objectif

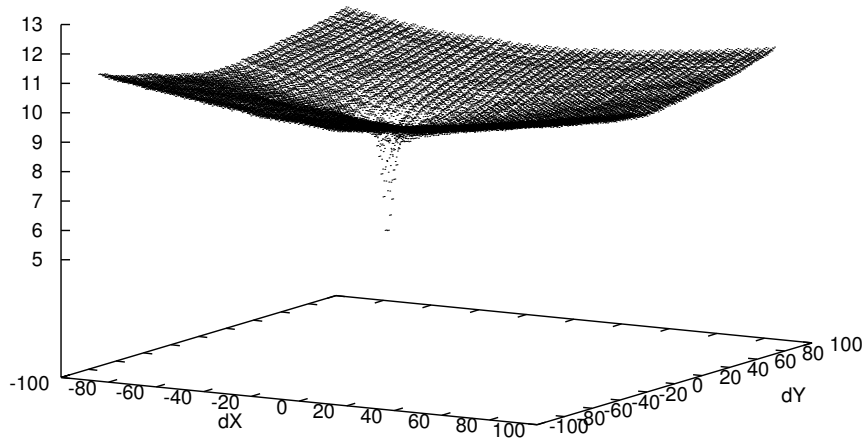
La fonction objectif (précédemment définie en détail dans la section 5.1.2.2) est définie comme $f : \mathbb{N} \rightarrow \mathbb{R}$ et doit être minimisée. La figure 5.3 montre l'aspect globalement convexe d'un problème de recalage simple.

5.1.3.2 Algorithmes *HCIAC*, *CHEDA* et recherche de Nelder-Mead

La métaheuristique *HCIAC* et l'algorithme de Nelder-Mead ont été décrits en détail dans le chapitre 3. La méthode *CHEDA* été présentée dans le chapitre 4. Le problème étant défini sur des variables entières, les méthodes ont dû être adaptées. Ainsi, nous avons simplement utilisé un arrondi sur le vecteur de la solution demandée à un instant donné. Nos tests tendent en effet à prouver que le comportement de l'algorithme n'est pas fondamentalement modifié si la structure de la fonction objectif ne présente pas de paliers étendus par rapport à la structure de l'espace de recherche. Comme le montre la figure



(a) basse résolution



(b) haute résolution

FIG. 5.3 – *Échantillonnage de la fonction objectif pour un problème simple de recalage (décrit dans la section 5.1.4.1).*

5.3, le problème fait apparaître des paliers surtout pour une basse résolution, mais ceux-ci ne changent pas la structure de la fonction objectif, qui demeure globalement convexe.

5.1.3.3 Réglage

L'algorithme *HCIAC* est utilisé avec le réglage classique suivant : $\rho = 0.5$, $\chi_m = 0.5$, $\chi_d = 0.2$, $\pi_m = 0.8$, $\pi_d = 0.5$, $\omega_\delta = 0.1$, $\chi_\tau = \omega_\tau = 0.5$, $\chi_\rho = \omega_\rho = 10$ (voir sections 3.2.3.3 et 3.2.4 pour les définitions des paramètres et leurs réglages). Les paramètres cruciaux que sont le nombre de fourmis et le nombre d'itérations du simplexe sont respectivement initialisés à : $\nu = 3$, $\eta = 10$. Ces réglages correspondent à un comportement "rapide" (i.e. permettant un faible nombre d'évaluations). Dans ce travail, la limite supérieure est fixée à 200 évaluations de la fonction objectif.

La méthode *CHEDA* est utilisée avec les réglages suivants : $\pi = 20$, $\beta = 30\%$, $\nu = 10$ (voir section 4.4 pour de plus amples précisions). L'algorithme s'arrête après un maximum de 200 évaluations.

L'algorithme *NMS* utilise ses réglages par défaut, comme nous l'avons présenté dans la section 3.2.2.1. L'algorithme s'arrête si la taille du simplexe est inférieure à 10^{-8}, ou s'il a atteint la limite de 200 évaluations.

5.1.4 Résultats

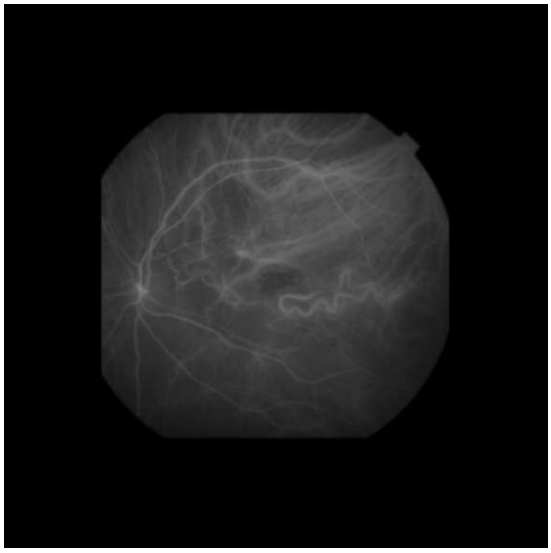
5.1.4.1 Tests préliminaires

Toutes les angiographies présentées dans ce travail ont été digitalisées à partir d'un signal vidéo à une résolution de 1024×1024 pixels et à une profondeur de niveaux de gris de 8 bits par pixel. Les algorithmes ont été testés sur différentes résolutions depuis 10% jusqu'à 100% de la taille de l'image originale, ainsi qu'avec ou sans filtrage médian de l'image gradient.

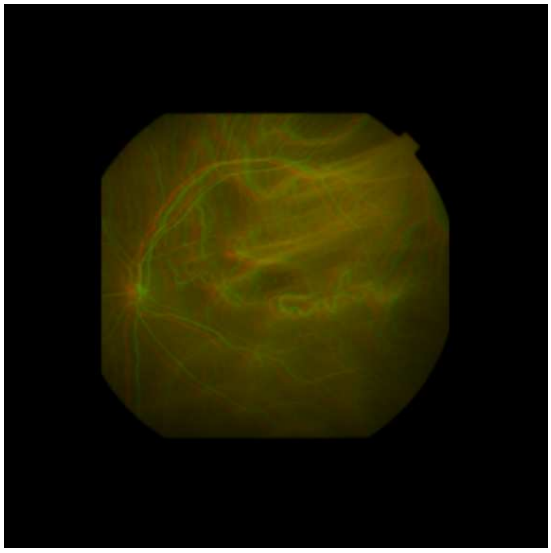
Afin de décrire le comportement de l'algorithme d'optimisation et de tester sa consistance sur le problème, nous utilisons un problème simple de recalage, où l'optimum global est connu (voir figure 5.4).

Nous ne présentons dans un premier temps que les résultats de *HCIAC*, afin de ne pas surcharger ce chapitre. Les résultats avec *CHEDA* mènent aux mêmes conclusions et seront présentés dans la section 5.1.4.3.

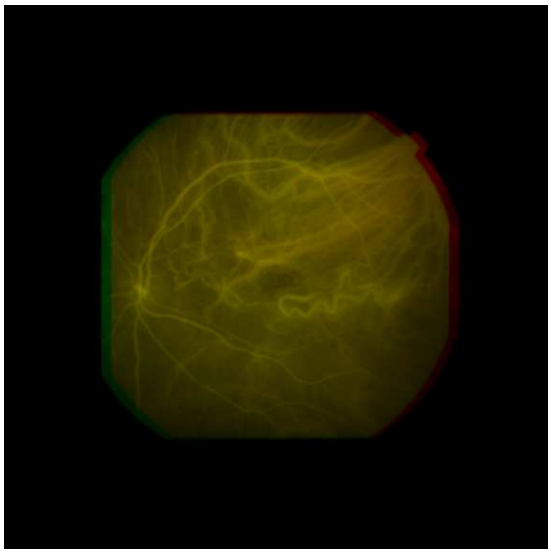
Les techniques d'optimisation classiques demandent moins de temps de calcul lors d'un recalage à basse résolution. Cependant, dans le cas des problèmes en haute résolution, le temps de calcul peut augmenter de manière drastique (comme le montre le tableau 5.1 et la figure 5.5). Au contraire, le temps de calcul de la métaheuristique reste pratiquement inchangé aux différentes résolutions (voir figure 5.6).



(a) Première image à l'ICG



(b) deux images non recalées



(c) images à l'ICG recalées

FIG. 5.4 – *Problème de recalage typique utilisé pour tester la consistance.*

TAB. 5.1 – *Comparaison des temps total de calcul (en secondes) pour les deux méthodes d'optimisation. Les valeurs sont notées avec un * quand l'algorithme n'a pu trouver d'optimum. Pour* HCIAC, *le temps indiqué comprend le temps nécessaire à la fois à l'algorithme d'optimisation et au calcul de la fonction objectif (le temps d'évaluation de cette dernière étant proportionnel à la résolution).*

Res. / Algo.	0.1 min	0.1 max	0.5 min	0.5 max	1.0 min	1.0 max
Flot optique	7	43*	124	1090*	704	4531*
HCIAC	13	30	86	195	367	826

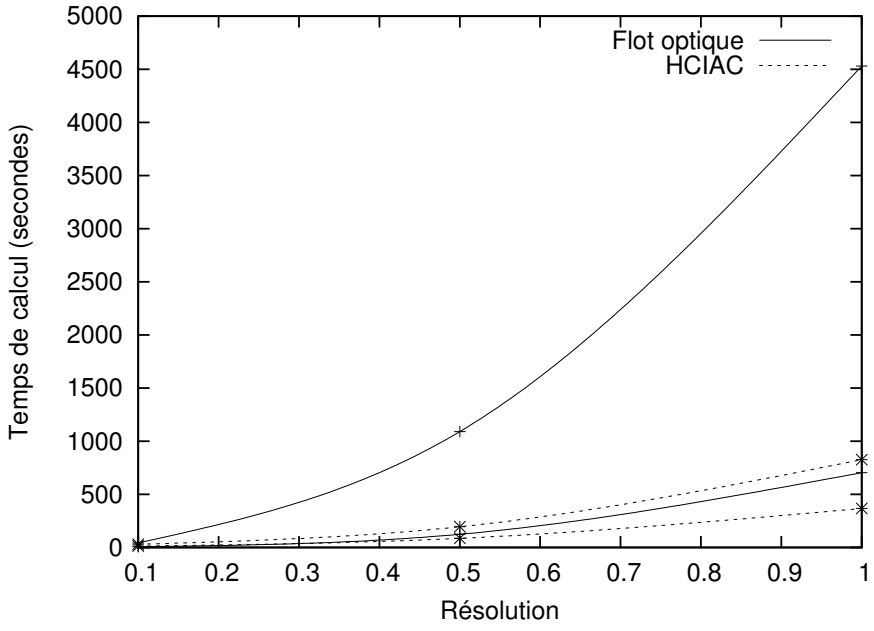


FIG. 5.5 – *Comparaison des temps de calcul (en secondes) pour les deux méthodes d'optimisation, les temps minimums et maximums sont indiqués en traits pointillés.*

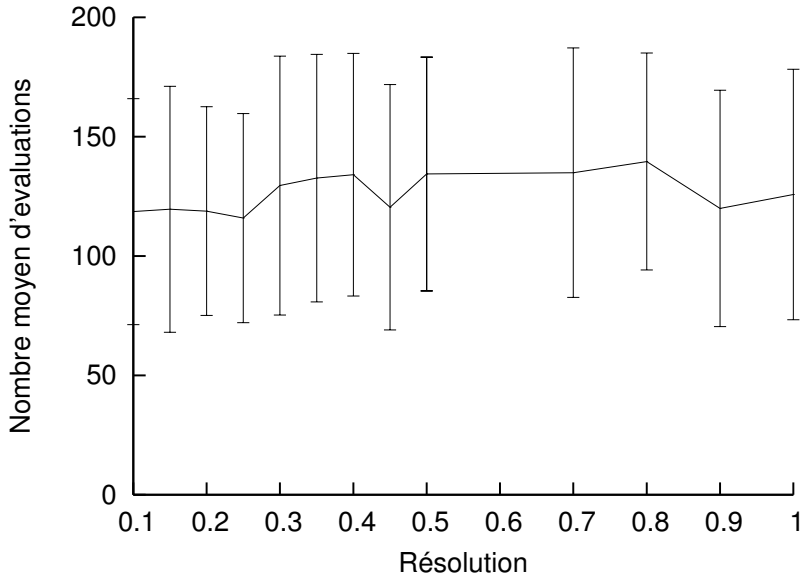


FIG. 5.6 – *Nombre d'évaluations de la fonction objectif nécessaires à* HCIAC.

5.1.4.2 Robustesse

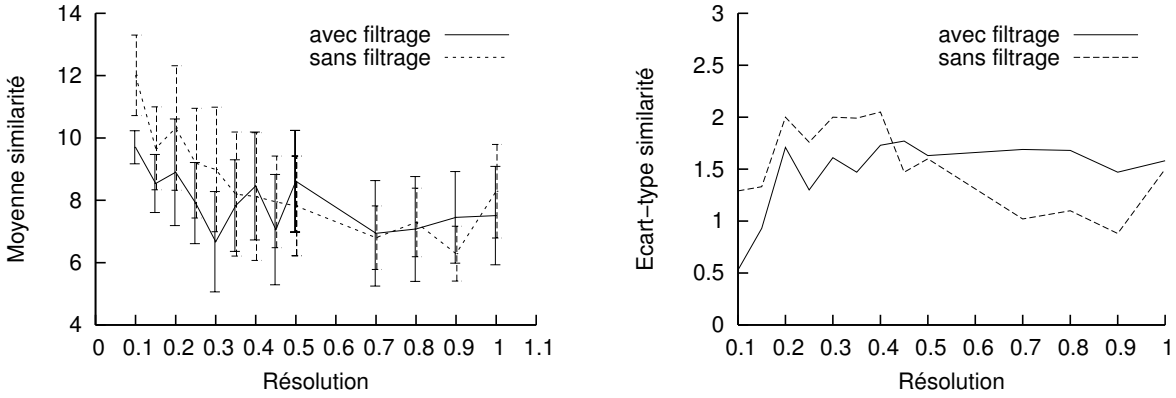
Dans l'optique de tester la robustesse de la méthode, nous avons effectué 50 tests sur le même problème de recalage. Les méthodes d'optimisation locale classiques étant employées sur des basses résolutions, nous avons testé l'impact de la résolution sur l'efficacité de la métaheuristique, notamment afin d'estimer l'intérêt de l'information supplémentaire apportée par la haute résolution. De plus, le filtrage médian supplémentaire est souvent utilisé pour améliorer l'étape d'optimisation. Cependant, comme le temps de calcul est crucial sur ce problème, nous avons testé l'impact du filtrage supplémentaire sur l'optimisation par *HCIAC*.

La figure 5.7 montre que le filtrage supplémentaire est en pratique intéressant pour les hautes résolutions. L'écart-type est une bonne mesure de la robustesse de l'algorithme, puisqu'il est un indicateur du nombre de mauvais recalages : plus l'écart-type est grand, plus l'algorithme échoue souvent à trouver un bon recalage.

La figure 5.8 montre que l'algorithme *NMS* utilisé seul peut trouver un recalage optimal, au prix d'une augmentation de l'écart-type proportionnelle à la résolution, alors que la résolution a peu d'impact sur l'efficacité de *HCIAC*.

5.1.4.3 Cas typiques

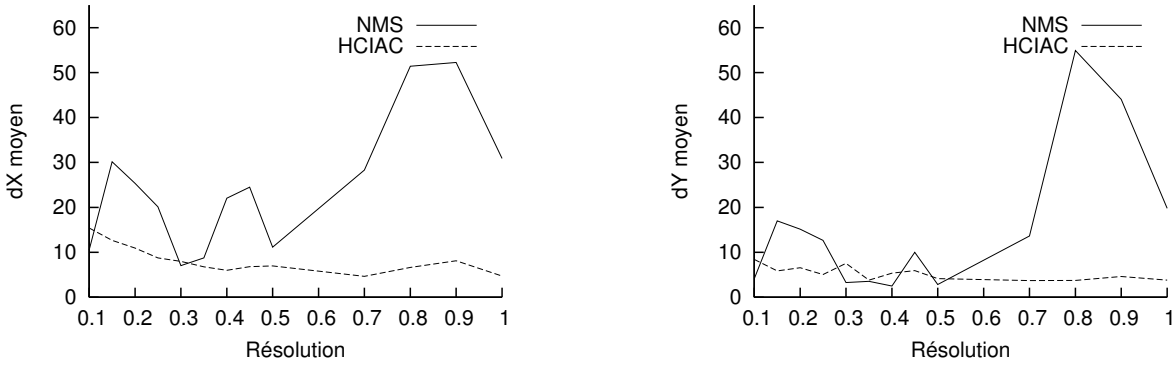
Conformément aux résultats de robustesse, nous avons donc testé les métaheuristiques sur une haute résolution (100% de la taille de l'image originale) sans filtrage médian



(a) moyenne et écart-type

(b) écart-type seul

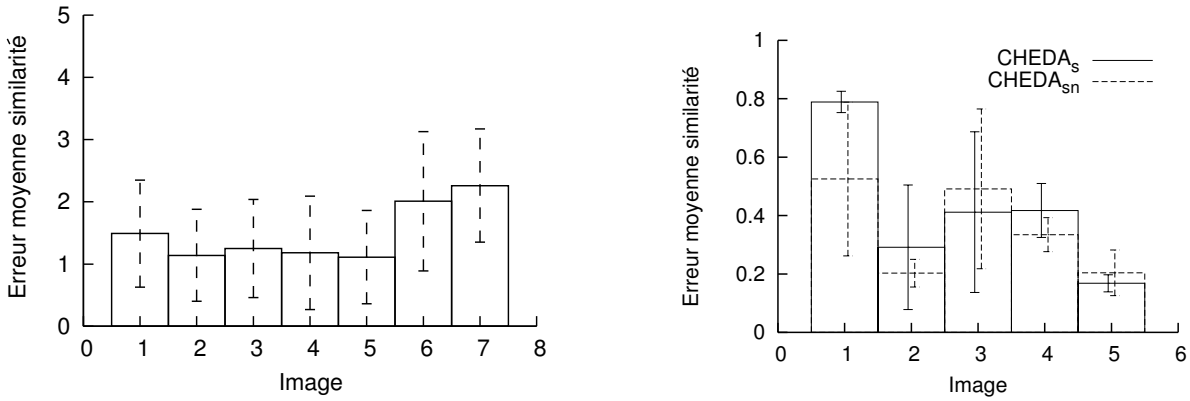
FIG. 5.7 – *Comparaison des moyennes et écarts-types des similarités selon la résolution.*



(a) déplacement horizontal

(b) déplacement vertical

FIG. 5.8 – *Écarts-types selon la résolution au recalage optimal, sans filtrage médian.*



(a) HCIAC (b) $CHEDA$

FIG. 5.9 – *Moyennes et écarts-types des similarités obtenues pour les différentes images de test.*

supplémentaire. Nous avons utilisé 8 couples d'images d'angiographies, effectuant 50 optimisations sur chaque image.

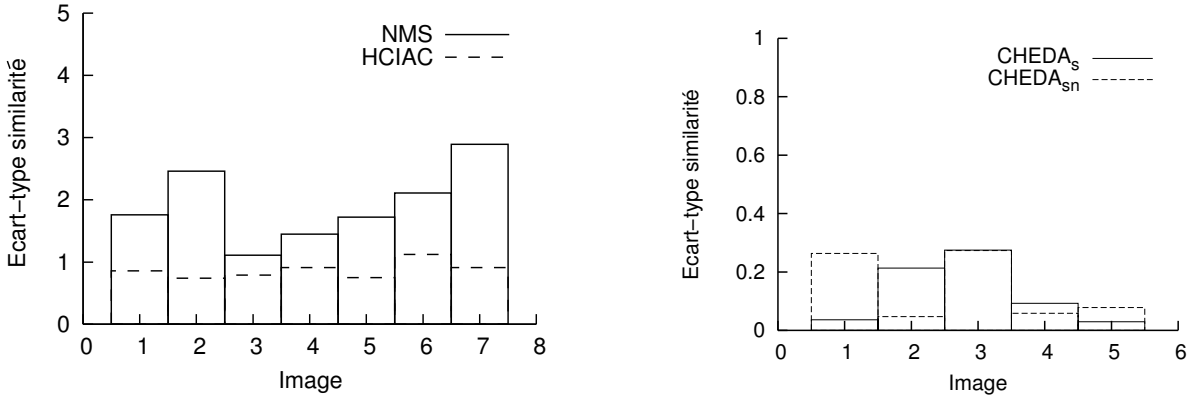
Comme le montre la figure 5.9, la métaheuristique $HCIAC$ ne peut déterminer avec précision l'optimum global pour chaque essai. En effet, si elle trouve toujours une valeur approchée du recalage idéal, elle peut parfois manquer de précision. À l'inverse les deux variantes de $CHEDA$ atteignent des erreurs plus faibles, montrant une plus grande précision. La variante $CHEDA_{sn}$ est soit équivalente à $CHEDA_s$ soit meilleure.

La figure 5.10 montre que la métaheuristique $HCIAC$ peut atteindre une meilleure robustesse que la recherche locale NMS utilisée seule. Les variantes $CHEDA$ montrent en revanche un écart-type bien plus faible.

Il peut cependant rester quelques problèmes dans le recalage précis, où une simple translation n'est pas suffisante pour représenter la véritable transformation. La figure 5.11 montre que bien qu'ayant un bon recalage global, il peut rester des erreurs dans la périphérie de l'image angiographique recalée.

5.1.5 Discussion

Nos résultats démontrent que l'utilisation d'une métaheuristique pour l'étape d'optimisation du recalage d'angiogramme peut être particulièrement utile dans le cas d'images à haute résolution. En effet, le principal avantage des méthodes présentées dans cet article réside dans leur coût constant en terme de temps de calcul. Avec des méthodes classiques comme le flot optique, le coût en temps de calcul augmente considérablement avec la résolution, alors qu'avec les méthodes présentées, il est constant. Cet avantage est dû au fait que les métaheuristiques opèrent par échantillonnage de la fonction ob-



(a) $HCIAC$ et NMS (b) $CHEDA$

FIG. 5.10 – *Écart-type des similarités obtenues pour différentes images, avec* HCIAC, NMS *et* CHEDA.

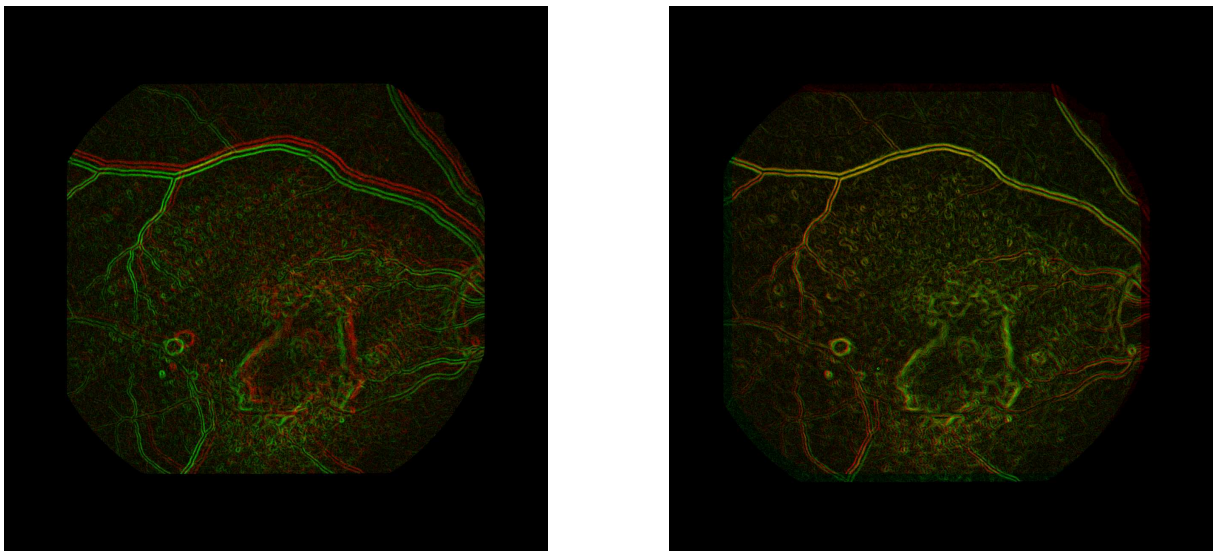
jectif, échantillonnage qui ne dépend pas de la résolution. Le coût en temps est donc principalement fonction du temps de calcul d'une solution.

Cependant, le temps de calcul reste toujours une contrainte critique et nos résultats montrent que, pour les hautes résolutions, utiliser un filtrage médian supplémentaire n'est pas un avantage, car il diminue la robustesse et augmente le temps de calcul, en ajoutant une étape supplémentaire. Ainsi, il devrait être évité d'ajouter trop de filtrages supplémentaires, ceux-ci pouvant éliminer des informations pertinentes utilisables par la métaheuristique, et ainsi réduire sa capacité à trouver l'optimum global.

La recherche de Nelder-Mead montre les même avantages que la métaheuristique du point de vue du temps de calcul. Cependant, nos résultats montrent que la robustesse de cette recherche locale décroît lors de son utilisation sur des problèmes à haute résolution. Ce problème est dû à la présence d'optima locaux qui peuvent piéger la recherche globale. En effet, ce problème est souvent rencontré dans les méthodes employées classiquement pour le recalage, et il est résolu par une approche multi-résolution. Mais de telles approches sont lourdes en temps de calcul, alors que l'utilisation d'une métaheuristique résout le problème des optima locaux, sans trop augmenter le temps de calcul.

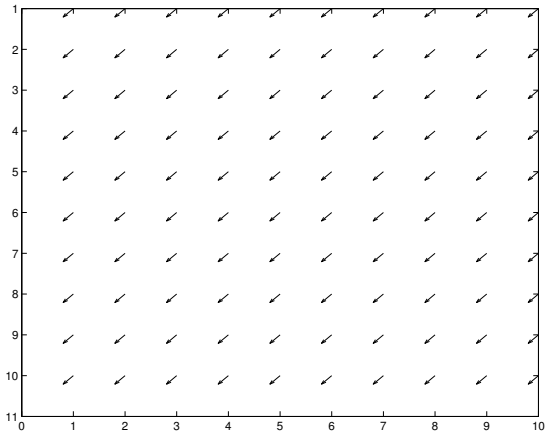
La méthode $CHEDA_{sn}$ semble la plus efficace sur ce problème, suivie de près par $CHEDA_s$, alors que $HCIAC$ a une efficacité relativement mauvaise. Une grande partie de l'efficacité de $CHEDA$ est sans doute due à l'utilisation d'une distribution normale, très proche de la forme générale de la fonction objectif.

Le problème des erreurs périphériques est moins crucial en comparaison de la recherche de la translation principale, qui est plus difficile à obtenir. Il ne peut être résolu via une



(a) non recalé

(b) recalé



(c) translation

FIG. 5.11 – *Exemple de mauvais recalage périphérique.*

approche rigide. Une méthode de recalage élastique [Nunes et al., 2004a] peut être utilisée, pour corriger cette transformation résiduelle.

5.1.6 Conclusion

Le recalage est un sujet important pour plusieurs applications concernant l'analyse d'images médicales, comme la correction de mouvements ou la fusion multimodale.

Du fait du mouvement des yeux durant l'acquisition d'angiographies, une étape de recalage de chaque image est nécessaire pour améliorer l'analyse quantitative des pathologies rétiniennes. Le recalage des images permet une comparaison directe avec l'image précédente, et peut ainsi permettre de juger de la progression de la maladie.

Cependant, les données angiographiques varient considérablement en intensité ; aussi un prétraitement est-il nécessaire pour pallier ce problème, ainsi qu'à la présence de bruit. Le filtrage de Wiener, ainsi que le calcul du gradient, sont indispensables, alors que le filtrage médian n'est pas nécessaire.

Les techniques d'optimisation présentées sont particulièrement adaptées pour les hautes résolutions, où des techniques plus classiques ne peuvent être utilisées, du fait d'un temps de calcul prohibitif. La métaheuristique *HCIAC* permet une meilleure robustesse que la recherche locale de Nelder-Mead. La méthode *CHEDA* présente cependant les meilleurs résultats, effectuant des recalages de bonne qualité.

Une piste d'amélioration possible serait de supprimer les informations de covariances de l'algorithme $CHEDA$; en effet, les déplacements verticaux et horizontaux de l'oeil sont décorrélés, une somme de distribution univariante pourrait dès lors obtenir des résultats similaires, avec un gain de rapidité.

Remerciements

Nous tenons à remercier tout particulièrement Jean-Claude Nunes, chercheur post-doctorat au LERISS, qui a proposé cette application et travaillé sur la partie concernant le traitement des images. Nous remercions également le Centre Hospitalier Intercommunal de Créteil (CHIC) pour avoir fourni les images d'angiographies rétiniennes.

Conclusion et perspectives

Dans ce travail de thèse, nous avons présenté un travail d'adaptation des algorithmes de colonies de fourmis pour l'optimisation continue. Les fourmis en colonie présentent, en effet, des comportements auto-organisés, où des interactions simples au niveau local permettent l'émergence d'un comportement global complexe. Les premiers travaux dans le domaine de l'utilisation de ces caractéristiques en recherche opérationnelle ont donné lieu à des métaheuristiques d'optimisation pour des problèmes combinatoires. Nous avons proposé deux approches pour étendre ces algorithmes au cas continu.

La première consiste à créer un système multi-agent où la communication joue un rôle central, en tant que processus permettant l'émergence d'un comportement global cohérent du système. Nous avons proposé, dans cette optique, trois algorithmes de *colonies de fourmis interagissantes* (*CIAC*) mettant l'accent sur les concepts de canaux de communication et de comportement individuel des fourmis. Ces méthodes ont la particularité d'être distribuées et flexibles, mais demandent un grand nombre d'évaluations et leur réglage est relativement délicat. Afin de pallier ce dernier problème, nous avons proposé une méthode de "métaréglage", où le réglage de la métaheuristique est considéré comme un problème d'optimisation biobjectif et optimisé en conséquence. Cette méthode permet de définir un indice unique, gérant la mise en place de tous les paramètres, depuis un comportement d'intensification jusqu'à un comportement de diversification.

La deuxième approche décrit les algorithmes de colonies de fourmis comme des méthodes manipulant un échantillonnage d'une distribution de probabilité. Nous avons proposé un algorithme à estimation de distribution (*CHEDA*) sur cette base. La distribution choisie pour s'adapter aux problèmes continus est une distribution normale multi-variante. La métaheuristique ainsi conçue est facilement manipulable : le nombre de paramètres est réduit et a un impact sur l'optimisation aisément compréhensible. Ses performances sont, par ailleurs, compétitives, bien que le grand nombre d'évaluations nécessaires soit toujours présent.

Afin d'améliorer les processus d'intensification, nos algorithmes ont fait l'objet d'une hybridation avec une recherche locale de Nelder-Mead. Dans les deux approches étudiées

(*HCIAC* et *CHEDA*), elle permet une amélioration significative de la précision des résultats.

Les algorithmes de colonies de fourmis du type "multi-agent" présentent des caractéristiques de flexibilité particulièrement intéressantes sur des problèmes combinatoires dynamiques. Nous avons donc adapté notre méthode hybride de colonies de fourmis interagissantes à des problèmes continus dynamiques (algorithme *DHCIAC*), pour lesquels nous avons également proposé un nouveau jeu de test cohérent.

Nos algorithmes ont enfin été appliqués dans le cadre d'un problème biomédical concernant les pathologies du vieillissement oculaire. Nous avons utilisé les métaheuristiques décrites dans cette thèse pour effectuer cette étape d'optimisation et montré qu'elles étaient plus particulièrement compétitives sur des images en haute résolution, où les méthodes classiques sont inutilisables.

Les perspectives de ce travail sont relativement nombreuses, tant du point de vue théorique que pratique. En premier lieu, il est possible d'approfondir le principe de métaréglage, qui peut s'avérer être un outil intéressant, non seulement pour régler les paramètres d'un algorithme, mais également pour comprendre son fonctionnement.

Une autre approche qui reste à approfondir est l'évaluation des algorithmes pour l'optimisation dynamique. De nombreuses fonctions de test existent, mais restent des initiatives ponctuelles, sans volonté de constituer un jeu de test cohérent. Une systématisation des principes des problèmes dynamiques pourrait étendre le jeu de test que nous proposons, pour couvrir un ensemble complet de problèmes types.

Une perspective importante peut être une formalisation plus poussée des relations entre les algorithmes à estimation de distribution et la programmation à mémoire adaptative. En effet, cette approche permet la conception de métaheuristiques réellement simples à manipuler et à comprendre, tout en demeurant puissantes. De plus, en suivant une telle structure, et en considérant, pour les processus de diversification et d'intensification, des "briques de base" empruntées aux autres métaheuristiques, une nouvelle manière de concevoir des métaheuristiques peut apparaître.

Enfin, une perspective intéressante reste les applications à des problèmes d'optimisation dans le domaine biomédical. De ce point de vue, le recalage d'images promet d'autres pistes de recherche, comme le recalage affine (prise en compte d'autres déformations que la translation, comme le zoom et la rotation), le recalage multi-échelle (optimisations itératives à différentes résolutions), ou encore sur des images en trois dimensions.

Annexe A

Annexes

A.1 Algorithmes concurrents exploités pour confrontation avec nos algorithmes

TAB. A.1 – *Algorithmes d'optimisation continue utilisés pour comparaison avec nos algorithmes.*

Abbr.	Nom complet	Type	Référence
$ECTS$	Enhanced Continuous Tabu Search	Recherche avec tabous	[Chelouah and Siarry, 2000b]
CGA	Continuous Genetic Algorithm	Algorithme évolutionnaire	[Chelouah and Siarry, 2000a]
DE	Differential Evolution	Algorithme évolutionnaire	[Storn and Price, 1995]
$CACO$	Continuous Ant Colony Algorithm	Colonie de fourmis	[Bilchev and Parmee, 1995]
API	(de la fourmi *Pachycondyla apicalis*)	Colonie de fourmis	[Monmarché et al., 2000b]
$UMDA_c$	Univariate Marginal Distribution Algorithm for Continuous domains	Estimation de distribution	[Larrañaga et al., 2000]
$MIMIC_c$	Mutual Information Maximizing Input Clustering for Continuous domains	Estimation de distribution	[Larrañaga et al., 2000]
$EGNA_{BIC}$	Estimation of Gaussian Network Algorithm (Bayesian Information Criterion)	Estimation de distribution	[Larrañaga et al., 1999]
$EGNA_{BGe}$	Estimation of Gaussian Network Algorithm (BGe metric)	Estimation de distribution	[Larrañaga et al., 1999]
$EGNA_{ee}$	Estimation of Gaussian Network Algorithm (edge exclusion)	Estimation de distribution	[Larrañaga et al., 1999]
$EMNA_{global}$	Estimation of Multivariate Normal Algorithm	Estimation de distribution	[Larrañaga et al., 2001]
$EMNA_a$	Estimation of Multivariate Normal Algorithm (adaptive)	Estimation de distribution	[Larrañaga et al., 2001]

A.2 Fonctions de test

La liste présentée ci-dessous énumère les fonctions tests utilisées dans ce travail. Nous utilisons les notations suivantes :

— *Dim.* le nombre de dimensions de la fonction,
— $\overrightarrow{x}$ le vecteur de variables,
— $\overrightarrow{x}^*$ l'optimum global,
— $x_i \in [\dots]$ le domaine de recherche, pour chaque variable,
— *Glob. Min.* ou *Glob. Max.* la valeur et (si connue) la position de l'optimum,
— *Nb. Loc. Min.* ou *Nb. Loc. Max.* le nombre d'optima locaux.

A.2.1 (D_1) Dreo1

— *Dim.* : 1
— $D_1(\overrightarrow{x}) = \left(4\cos\left(\frac{1}{2}x^2+1\right)+\frac{1}{9}x^2+\frac{1}{3}x\right)-\left(4\cos(3)-\frac{2}{9}\right)$
— $x \in [-7, 15]$
— *Glob. Min.* : $D_1(-2) = 0$

A.2.2 (B_2) B2

— *Dim.* : 2
— $B_2(\overrightarrow{x}) = x_1^2 + 2x_2^2 - 0.3 \cdot \cos(3\pi x_1) - 0.4 \cdot \cos(4\pi x_2) + 0.7$
— $x_i \in [-100, 100]$
— *Glob. Min.* : $B_2(0, 0) = 0$

A.2.3 (MG) Martin & Gaddy

— *Dim.* : 2
— $MG(\overrightarrow{x}) = (x_1 - x_2)^2 + \left(\frac{x_1+x_2-10}{3}\right)^2$
— $x_i \in [-20, 20]$
— *Glob. Min.* : $MG(5, 5) = 0$

A.2.4 (GP) Goldstein & Price

— *Dim.* : 2
— $GP(\overrightarrow{x}) = \Big(1+(x_1+x_2+1)^2 \cdot \big(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2\big)\Big) \cdot$
$\Big(30+(2x_1-3x_2)^2 \cdot \big(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2\big)\Big)$
— $x_i \in [-2, 2]$
— *Glob. Min.* : $GP(0, -1) = 3$

A.2.5 $(S_{4,n})$ Shekel

— *Dim.* : 4
— $S_{4,n}(\overrightarrow{x}) = -\sum_{i=1}^{n}\left(\frac{1}{(x-a_i)^T(x-a_i)+c_i}\right)$
avec $x = (x_1, x_2, x_3, x_4)^T$; $a_i = (a_i^1, a_i^2, a_i^3, a_i^4)^T$ et $n \in \{1, 2, 3\}$
— $x_i \in [0, 10]$
— *Glob. Min.* : $S_{4,5}\left(\overrightarrow{x^*}\right) = -10.1532$; $S_{4,5}\left(\overrightarrow{x^*}\right) = -10.40294$; $S_{4,5}\left(\overrightarrow{x^*}\right) = -10.53641$
— *Nb. Loc. Min.* : $(n-1)$

i	a_i^T				c_i
1	4	4	4	4	0.1
2	1	1	1	1	0.2
3	8	8	8	8	0.2
4	6	6	6	6	0.4
5	3	7	3	7	0.4
6	2	9	2	9	0.6
7	5	5	3	3	0.3
8	8	1	8	1	0.7
9	6	2	6	2	0.5
10	7	3.6	7	3.6	0.5

A.2.6 (St) Step

— *Dim.* : 5
— $St(\overrightarrow{x}) = 25 - \sum_{i=1}^{5} \lfloor x_i \rfloor^*$ avec $\lfloor x_i \rfloor^*$ plus grand entier $\leq x$
— $x_i \in [-5.12, 5.12]$
— *Glob. Max.* : $St\left(\overrightarrow{x^*}\right) = 55$

A.2.7 (SM) Sphere Model

— *Dim.* : 6
— $SM(\overrightarrow{x}) = \sum_{i=1}^{6} x_i^2$
— $x_i \in [-5.12, 5.12]$
— *Glob. Min.* : $SM(0, \dots) = 0$

A.2.8 (B_{f_1}) Baluja f1

— *Dim.* : 100
— $B_{f_1}(\overrightarrow{x}) = \left(\epsilon + \left||y_1| + \sum_{i=1}^{100} |y_i|\right|\right)^{-1}$ avec $y_1 = x_1, \quad y_i = x_i + y_{i-1}$
— $x_i \in [-2.56, 2.56]$

— *Glob. Max.* : $B_{f_1}(0,\dots) = 100000$

A.2.9 (B_{f_2}) Baluja f2

— *Dim.* : 100
— $B_{f_2}(\overrightarrow{x}) = \left(\epsilon + \left||y_1| + \sum_{i=1}^{100} |y_i|\right|\right)^{-1}$ avec $y_1 = x_1, \quad y_i = x_i + \sin(y_{i-1})$
— $x_i \in [-2.56, 2.56]$
— *Glob. Max.* : $B_{f_2}(0,\dots) = 100000$

A.2.10 (B_{f_3}) Baluja f3

— *Dim.* : 100
— $B_{f_3}(\overrightarrow{x}) = \left(\epsilon + \sum_{i=1}^{100} |0.024 \cdot (i+1) - x_i|\right)^{-1}$
— $x_i \in [-2.56, 2.56]$
— *Glob. Max.* : $B_{f_3}(0,\dots) = 100000$

A.2.11 (Gr_n) Griewangk

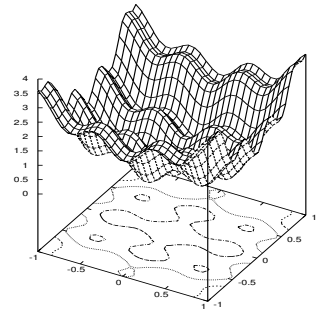
— *Dim.* : n
— $Gr(\overrightarrow{x}) = 1 + \left(\sum_{i=1}^{n} \left(\frac{x_i^2}{4000}\right) - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right)\right)$
— $x_i \in [-512, 512]$
— *Glob. Min.* : $Gr(0,\dots) = 0$

A.2.12 (R_n) Rosenbrock

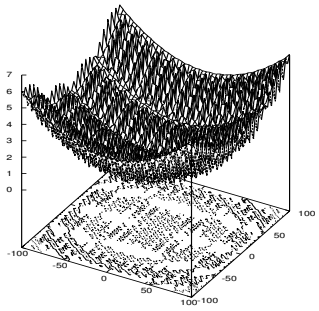
— *Dim.* : n
— $R_n(\overrightarrow{x}) = \sum_{i=1}^{n-1} \left(100 \cdot \left(x_i^2 - x_{i+1}\right)^2 + (x_i - 1)^2\right)$
— $x_i \in [-5, 10]$
— *Glob. Min.* : $R_n(1,\dots) = 0$

A.3 Représentations graphiques de fonctions de test

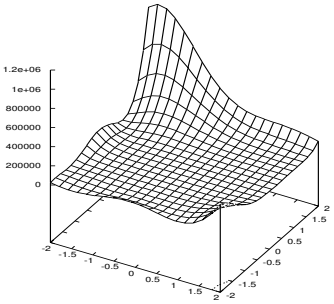
Nous avons représenté graphiquement certaines fonctions de test utilisées afin d'illustrer certains types de problèmes d'optimisation. Les fonctions avec plus de deux variables ne sont représentées que pour les deux premières dimensions. La taille de l'espace de recherche présentée peut être différente de celle utilisée, pour des raisons de lisibilité. Enfin, certains optima locaux peuvent ne pas apparaître à cause de l'échelle.



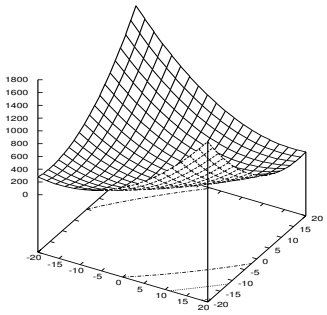
(a) B_2



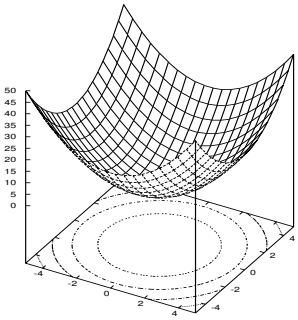
(b) Gr



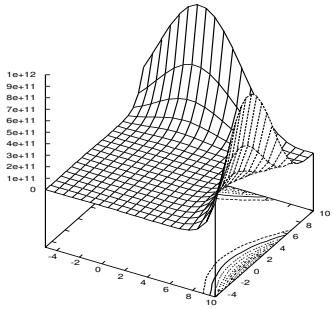
(c) GP



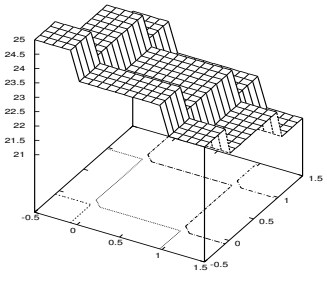
(d) MG



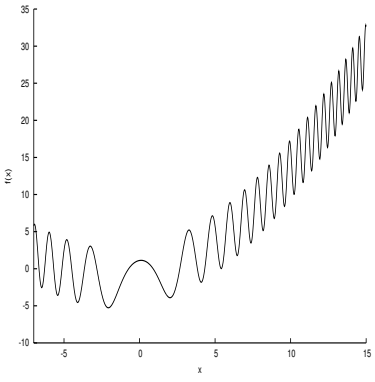
(e) SM



(f) R_2

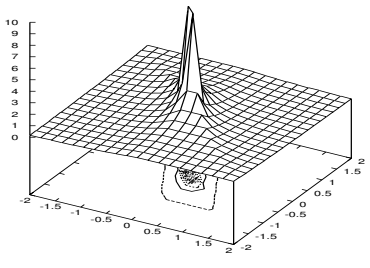


(g) St

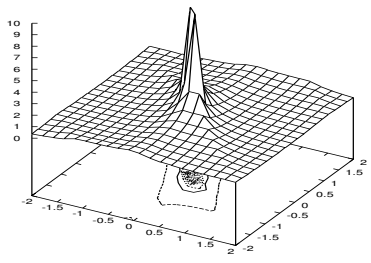


(h) D_1

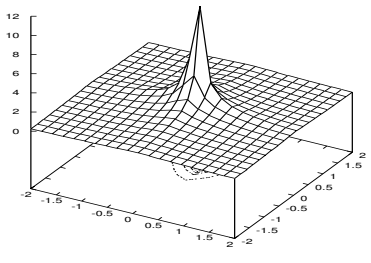
FIG. A.1 – *Représentation de 8 fonctions de test à minimiser.*



(a) B_{f_1}



(b) B_{f_2}



(c) B_{f_3}

FIG. A.2 – *Représentation de 3 fonctions de test à maximiser.*

A.4 Jeu de test dynamique

Pour tous les problèmes listés ci-après, t est un entier représentant le temps écoulé.

A.4.1 OPL : Optimum Position Linéaire

— Fonction de temps *linéaire* : $T(t) = t$,
— *Easom* comme fonction de structure :

$$D\left(x_1, x_2, t\right) = -\cos\left(x_1\right) \cdot \cos\left(x_2\right) \cdot e^{-(x_1 - T(t))^2 - (x_2 - T(t))^2}$$

avec $-50 < x_1, x_2 < 50$,
— Cible : position de l'optimum,
— Caractéristiques : fonction continue, espace de recherche symétrique, mono-modal, bassin d'attraction de l'optimum convexe.

A.4.2 APL : Tout Position Linéaire

— Fonction de temps *linéaire* : $T(t) = t$,
— *B2* comme fonction de structure :

$$\begin{aligned} D\left(x_1, x_2, t\right) = \; & \left(x_1 - T\left(t\right)\right)^2 + 2\left(x_2 - T\left(t\right)\right)^2 \\ & -0.3 \cdot \cos\left(3\pi - \left(x_1 - T\left(t\right)\right)\right) \\ & -0.4 \cdot \cos\left(4\pi - \left(x_2 - T\left(t\right)\right)\right) \\ & +0.7 \end{aligned}$$

avec $-10 < x_1, x_2 < 10$,
— Cible : position de la fonction entière,
— Caractéristiques : fonction continue, convexe, espace de recherche symétrique, fonction symétrique, multi-modale, périodique, plusieurs optima locaux.

A.4.3 OVP : Optimum Valeur Périodique

— Fonction de temps *périodique* : $T\left(t\right) = 8 \cdot \cos\left(t + 0.5\right)$,
— *Morrison* comme fonction de structure :

$$D\left(x_1, x_2, t\right) = \sum_{i=1}^{5} \min\left(-H_i + R_i \cdot \left(\left(x_1 - P_{i,1}\right)^2 + \left(x_2 - P_{i,2}\right)^2\right)\right)$$

avec :
– $-10 < x_1, x_2 < 10$

- $H = \{1, 3, 5, 7 + T(t), 9\}$
- $R = \{1, 1, 1, 1, 1\}$
- $P = \left\{ \begin{array}{ccccc} 2.5 & -2.5 & -5 & 0 & -4 \\ -2.5 & 2.5 & 5 & 0 & -4 \end{array} \right\}$

— Cible : valeur de l'optimum n°4,

— Caractéristiques : fonction continue, espace de recherche symétrique, multi-modale, bassin d'attraction de l'optimum convexe.

A.4.4 AVP : Tout Valeur Périodique

— Fonction de temps *périodique* : $T(t) = \cos(0.2 \cdot t)$

— *Shekel* comme fonction de structure :

$$D(x,t) = T(t) + \sum_{i=1}^{n} \left(\frac{1}{(x - a_i)^T (x - a_i) + c_i} \right)$$

avec :

- $x = (x_1, x_2, x_3, x_4)^T$
- $a_i = (a_i^1, a_i^2, a_i^3, a_i^4)^T$ et

i	a_i^T				c_i
1	4	4	4	4	0.1
2	1	1	1	1	0.2
3	8	8	8	8	0.2
4	6	6	6	6	0.4
5	3	7	3	7	0.4
6	2	9	2	9	0.6
7	5	5	3	3	0.3
8	8	1	8	1	0.7
9	6	2	6	2	0.5
10	7	3.6	7	3.6	0.5

- $n = 5$
- $0 < x_i < 10$

— Cible : valeur de la fonction entière,

— Caractéristiques : fonction continue, multi-modale, à plusieurs optima locaux.

A.4.5 ADL : Tout Dimension Linéaire

— Fonction de temps *linéaire* : $T(t) = t$,

— *Rosenbrock* comme fonction de structure :

$$D(x,t) = \sum_{i=1}^{T(t)} \left(100 \cdot \left(x_i^2 - x_{i+1} \right)^2 + (x_i - 1) \right)$$

avec $-5 < x_i < 10$,

— Cible :nombre de dimensions,

— Caractéristiques : fonction continue, espace de recherche symétrique, quadratique, multi-modale, à plusieurs optima locaux.

A.4.6 O(P+V)P : Optimum Position + Valeur Périodique

— Fonction de temps périodique : $T(t) = 8 \cdot \cos(t + 0.5)$,

— *Morrison* comme fonction de structure :

$$D(x_1, x_2, t) = \sum_{i=1}^{5} \min\left(-H_i + R_i \cdot \left(\left(x_1 - P_{i,1}\right)^2 + \left(x_2 - P_{i,2}\right)^2\right)\right)$$

avec :

- $-10 < x_1, x_2 < 10$
- $H = \{1, 3, 5, (7 + T(t)), 9\}$
- $R = \{1, 1, 1, 1, 1\}$
- $P = \begin{Bmatrix} 2.5 & -2.5 & -5 & T(t) & -4 \\ -2.5 & 2.5 & 5 & T(t) & -4 \end{Bmatrix}$

— Cible : position et valeur de l'optimum n°4,

— Caractéristiques : fonction continue, espace de recherche symétrique, multi-modale, bassin d'attraction de l'optimum convexe.

A.5 Simulation d'une loi normale multi-variante

A.5.1 Loi normale

Pour une seule variable aléatoire, on peut utiliser la méthode de Box-Muller. Si x_1 et x_2 sont des variables aléatoires suivant une distribution uniforme sur $]0, 1[$, alors les variables :

$$y_1 = \sqrt{-2 \ln x_1} \cos 2\pi x_2$$
$$y_2 = \sqrt{-2 \ln x_1} \sin 2\pi x_2$$

suivent une loi normale réduite. On en déduit que les variables $z_1 = m + s \cdot y_1$ et $z_2 = m + s \cdot y_2$ suivent une loi normale d'espérance m et d'écart-type s.

A.5.2 Loi multi-normale

La loi normale multi-variante (ou loi multi-normale) correspond à l'extension de la loi normale pour plusieurs variables $x_1, x_2, \ldots, x_n$, elle est caractérisée par un vecteur de moyennes m et une matrice de variance-covariance V. Chaque élément m_i de m représente l'espérance de x_i. La matrice V est symétrique définie positive. L'élément (diagonal) V_{ii} de V représente la variance σ_i^2 de la variable x_i. L'élément (non diagonal) V_{ij} représente la covariance des variables x_i et x_j.

Pour simuler une loi multi-normale de paramètres m et V, on utilise la méthode suivante :

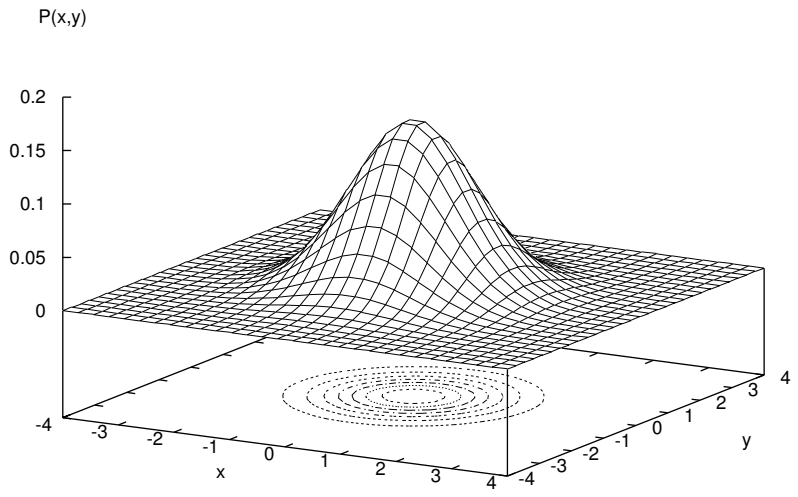


FIG. A.3 – *Distribution de probabilité d'une loi normale bi-variante centrée réduite*

— soit u un vecteur constitué de n nombres distribués selon la loi normale centrée-réduite,
— soit L la matrice résultant de la factorisation de Cholesky de la matrice V,
— le vecteur $y = m + Lu$ suit la loi multi-normale de moyenne m et de variance-covariance V.

A.5.2.1 Factorisation de Cholesky

On cherche la matrice :

$$L = \begin{bmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix}$$

De l'égalité $A = LL^T$ on déduit :

$$a_{ij} = \left(LL^T\right)_{ij} = \sum_{k=1}^{n} l_{ik} l_{jk} = \sum_{k=1}^{\min\{i,j\}} l_{ik} l_{jk},\ 1 \leq i, j \leq n$$

puisque $l_{pq} = 0$ si $1 \leq p < q \leq n$.

La matrice A étant symétrique, il suffit que les relations ci-dessus soient vérifiées pour $i \leq j$, c'est-à-dire que les éléments l_{ij} de la matrice L doivent satisfaire :

$$a_{ij} = \sum_{k=1}^{i} l_{ik} l_{jk},\ 1 \leq i, j \leq n$$

Pour $i = 1$, on détermine la première colonne de L :

$$\begin{array}{llll} (j=1) & a_{11} = l_{11}^2 & \text{d'où} & l_{11} = \sqrt{a_{11}} \\ (j=2) & a_{12} = b_{11} l_{21} & \text{d'où} & l_{21} = \frac{a_{12}}{l_{11}} \\ \ldots & & & \\ (j=n) & a_{1n} = l_{11} l_{n1} & \text{d'où} & l_{n1} = \frac{a_{1n}}{l_{11}} \end{array}$$

On détermine la $i^{\text{ère}}$ colonne de L, après avoir calculé les $(i-1)$ premières colonnes :

$$\begin{array}{llll} (j=1) & a_{ii} = l_{i1} + \ldots + l_{ii} l_{ii} & \text{d'où} & l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2} \\ (j=2) & a_{i,i+1} = l_{i1} l_{i+1} + \ldots + l_{ii} l_{i+1,i} & \text{d'où} & l_{i+1,i} = \frac{a_{i,i+1} - \sum_{k=1}^{i-1} l_{ik} l_{i+1,k}}{l_{ii}} \\ \ldots & & & \\ (j=n) & a_n = l_{i1} l_{n1} + \ldots + l_{ii} l_{ni} & \text{d'où} & l_{ni} = \frac{a_{in} - \sum_{k=1}^{i-1} l_{ik} l_{nk}}{l_{ii}} \end{array}$$

Il résulte du théorème précédent qu'il est possible de choisir tous les éléments b_{ii} en assurant que toutes les quantités

$$a_{11}, \ldots, a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2, \ldots$$

sont positives.

A.5.2.2 Estimation d'une matrice de variance-covariance

Si X est un vecteur de p points, dans n dimensions et de moyenne μ, alors sa matrice de variance-covariance est la matrice de dimension $n \times n$ symétrique définie positive :

$$var\,(X) = E\left((X - \mu)\,(X - \mu)^T\right)$$

Bibliographie commentée

Générale

[Dréo et al., 2003, Dréo et al., 2005] : Un livre qui tente de couvrir les principales métaheuristiques, plus des études de cas.

[Glover and Kochenberger, 2003] : Une revue en anglais d'un grand nombre de métaheuristiques différentes, sous la forme d'une collection de chapitres écrits par différents spécialistes du domaine.

[Teghem and Pirlot, 2002] : Un ouvrage en français abordant le sujet de la recherche opérationnelle.

[Pham and Karaboga, 2000], [Saït and Youssef, 1999], [Reeves, 1995] : Des livres en anglais traitant des métaheuristiques et de leurs applications.

Métaheuristiques

Algorithmes de colonies de fourmis

[Bonabeau et al., 1999, Dorigo and Stützle, 2004] : Ces ouvrages traitent des algorithmes de colonies de fourmis comme des systèmes faisant preuve d'intelligence en essaim. Le premier livre s'articule autour de concepts biologiques et algorithmiques, notamment autour des métaheuristiques de colonies de fourmis. Le second est centré sur l'optimisation uniquement.

[Dorigo and Stützle, 2003] : Un chapitre spécifiquement dédié aux algorithmes des colonies de fourmis, dans un livre généraliste sur les métaheuristiques. Moins riche que le précédent, mais plus récent.

[Dorigo et al., 2002, Dorigo et al., 2004] : Les actes des derniers congrès *ANTS* sur les "algorithmes de fourmis", une vision rapide des recherches les plus récentes dans le domaine. Le congrès se tient tous les deux ans depuis l'année 1998.

Autres métaheuristiques

[Larrañaga and Lozano, 2002] : Un ouvrage de référence sur les algorithmes à estimation de distribution, il couvre les fondements théoriques, les différents algorithmes existants et plusieurs applications.

[Siarry and Dreyfus, 1989] : Un ouvrage sur le recuit simulé, un peu ancien mais qui présente l'essentiel en français.

[Goldberg, 1995] : Un livre d'un des fondateurs des algorithmes évolutionnaires, traduit en français.

[Eberhart et al., 2001] : Une revue sur l'intelligence en essaim, qui aborde entre autres sujets l'optimisation par essaims particulaires.

[Glover and Laguna, 1997] : Un ouvrage de référence sur la recherche avec tabous, par des spécialistes du domaine.

[Clerc, 2004] : Un ouvrage en français traitant de l'optimisation par essaim particulaire, depuis la théorie jusqu'à l'implémentation pratique.

Optimisation difficile

[Collette and Siarry, 2002] : Ce livre aborde les différents aspects de l'optimisation multiobjectif, depuis la conception d'algorithmes à leurs applications, en passant par leur évaluation.

Biologie

[Hölldobler and Wilson, 1990] : Ce livre recense une somme impressionnante de connaissances sur la biologie des fourmis. Une bible en la matière, qui a reçu le prix Pulitzer en 1991.

[Camazine et al., 2000] : On trouvera ici une description très complète des principes de l'auto-organisation dans les systèmes biologiques, accompagnée de nombreux exemples. Des descriptions de modèles permettent de comprendre les fondements théoriques de l'auto-organisation.

Références bibliographiques

[Aarts and Van Laarhoven, 1985] Aarts, E. H. L. and Van Laarhoven, P. J. M. (1985). Statistical cooling : a general approach to combinatorial optimisation problems. *Philips J. of Research*, 40 :193–226.

[Aupetit et al., 2003] Aupetit, S., Monmarché, N., Slimane, M., Guinot, C., and Venturini, G. (2003). Clustering and Dynamic Data Visualization with Artificial Flying Insect. In *Genetic and Evolutionary Computation Conference (GECCO)*.

[Baeck et al., 2000a] Baeck, T., Fogel, D. B., and Michalewicz, Z. (2000a). *Evolutionary Computation 1 : Basic Algorithms and Operators*. Institute of Physics Publishing.

[Baeck et al., 2000b] Baeck, T., Fogel, D. B., and Michalewicz, Z. (2000b). *Evolutionary Computation 2 : Advanced Algorithms and Operators*. Institute of Physics Publishing.

[Baluja, 1994] Baluja, S. (1994). Population-based Incremental Learning : A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning. Technical Report CMU-CS-94-163, Carnegie Mellon University.

[Baluja, 1995] Baluja, S. (1995). An empirical comparison of seven iterative and evolutionary function optimization heuristics. Technical Report CMU-CS-95-193, Carnegie Mellon University.

[Baluja, 1997] Baluja, S. (1997). Genetic Algorithms and Explicit Search Statistics. *Advances in Neural Information Processing Systems*, 9 :319–325.

[Baluja and Caruana, 1995] Baluja, S. and Caruana, R. (1995). Removing the Genetics from the Standard Genetic Algorithm. In Prieditis, A. and Russel, S., editors, *International Conference on Machine Learning*, pages 38–46, Lake Tahoe, California. Morgan Kaufmann.

[Baluja and Davies, 1998] Baluja, S. and Davies, S. (1998). Fast probabilistic modeling for combinatorial optimization. In *Fifteenth National Conference on Artificial Intelligence, Tenth Conference on Innovative Applications of Artificial Intelligence*, Madison, Wisconsin.

[Bangham et al., 1996] Bangham, J. A., Harvey, R., and Ling, P. D. (1996). Morphological scale-space preserving transforms in many dimensions. *J. Electronic Imaging*, 5 :283–299.

[Beckers et al., 1992] Beckers, R., Deneubourg, J. L., and Goss, S. (1992). Trails and U-Turns in the Selection of a Path by the Ant Lasius Niger. *J. Theor. Biol.*, 159 :397–415.

[Bengoetxea et al., 2002] Bengoetxea, E., Miquélez, T., Larrañaga, P., and Lozano, J. A. (2002). *Estimation of Distribution Algorithms, A New Tool for Evolutionary Computation*, chapter Experimental Results in Function Optimization with EDAs in Continuous Domain, pages 181–194. Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers.

[Berger et al., 1999] Berger, J. W., Leventon, M. E., Hata, N., Wells, W., and Kinikis, R. (1999). Design considerations for a computer-vision-enabled ophtalmic augmented reality environment. *Lectures Notes in Computer Science*, 1205 :399–410.

[Berro, 2001] Berro, A. (2001). *Optimisation multiobjectif et stratégie d'évolution en environnement dynamique*. PhD thesis, Université Paul Sabatier, Toulouse.

[Bieszczad and White, 1999] Bieszczad, A. and White, T. (1999). *The Fundamentals of Network Management*, chapter Mobile Agents for Network Management. Plenum books.

[Bilchev and Parmee, 1995] Bilchev, G. and Parmee, I. (1995). The Ant Colony Metaphor for Searching Continuous Design Spaces. *Lecture Notes in Computer Science*, 993 :25–39.

[Bonabeau et al., 1999] Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). *Swarm Intelligence, From Natural to Artificial Systems*. Oxford University Press.

[Bonabeau et al., 1996] Bonabeau, E., Theraulaz, G., and Deneubourg, J.-L. (1996). Quantitative Study of the Fixed Threshold Model for the Regulation of Division of Labour in Insect Societies. In *Proceedings Roy. Soc. London B*, volume 263.

[Bonabeau et al., 1998] Bonabeau, E., Theraulaz, G., and Deneubourg, J.-L. (1998). Fixed Response Thresholds and the Regulation of Division of Labor in Insect Societies. *Bulletin of Mathematical Biology*, (60) :753–807.

[Bosman and Thierens, 2000a] Bosman, P. and Thierens, D. (2000a). Continuous iterated density estimation evolutionary algorithms within the IDEA framework. In Muehlenbein, M. and Rodriguez, A., editors, *Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM Workshop at the Genetic and Evolutionary Computation Conference GECCO-2000*, pages 197–200, San Francisco, California. Morgan Kauffmann.

[Bosman and Thierens, 2000b] Bosman, P. and Thierens, D. (2000b). IDEAs based on the normal kernels probability density function. Technical Report UU-CS-2000-11, Utrecht University.

[Bosman and Thierens, 1999] Bosman, P. A. N. and Thierens, D. (1999). An algorithmic framework for density estimation based evolutionnary algorithm. Technical Report UU-CS-1999-46, Utrech University.

[Branke, 2001] Branke, J. (2001). *Evolutionary Optimization in Dynamic Environments*, volume 3 of *Genetic Algorithms and Evolutionary Computation*. Kluwer Academic Publishers.

[Brown, 1992] Brown, L. G. (1992). A survey of image registration techniques. *ACM Comput. Surveys*, 24 :325–376.

[Camazine et al., 2000] Camazine, S., Deneubourg, J., Franks, N., Sneyd, J., Theraulaz, G., and Bonabeau, E. (2000). *Self-Organization in Biological Systems*. Princeton University Press.

[Campos et al., 2000] Campos, M., Bonabeau, E., Theraulaz, G., and Deneubourg, J.-L. (2000). Dynamic Scheduling and Division of Labor in Social Insects. In *Adaptive Behavior*, pages 83–96.

[Can and Stewart, 1999] Can, A. and Stewart, C. V. (1999). Robust hierarchical algorithm for constructing a mosaic from images of the curved human retina. In *IEEE Conf. on Computer Vision and Pattern Recognition*, volume 22.

[Carlisle, 2002] Carlisle, A. (2002). Tracking Changing Extrema with Adaptive Particle Swarm Optimizer. In *ISSCI 2002, World Automation Congress*, Orlando, USA.

[Cerny, 1985] Cerny, V. (1985). Thermodynamical approach to the traveling salesman problem : an efficient simulation algorithm. *J. of Optimization Theory and Applications*, 45(1) :41–51.

[Chelouah and Siarry, 2000a] Chelouah, R. and Siarry, P. (2000a). A Continuous Genetic Algorithm Designed for the Global Optimization of Multimodal Functions. *Journal of Heuristics*, 6 :191–213.

[Chelouah and Siarry, 2000b] Chelouah, R. and Siarry, P. (2000b). Tabu Search Applied to Global Optimization. *European Journal of Operational Research*, 123 :256–270.

[Choo, 2000] Choo, S.-Y. (2000). *Genetic Algorithms and Genetic Programming at Stanford 2000*, chapter Emergence of a Division of Labour in a Bee Colony, pages 98–107. Stanford Bookstore, Stanford, California.

[Cicirello and Smith, 2001] Cicirello, V. and Smith, S. (2001). Wasp-like Agents for distributed Factory Coordination. Technical Report CMU-RI-TR-01-39, Robotics Institute, Carnegie Mellon University, Pittsburgh.

[Cideciyan et al., 1992] Cideciyan, A. V., Jacobson, S. G., Kemp, C. M., Knighton, R. W., and Nagel, J. H. (1992). Registration of high resolution images of the retina. In *SPIE : Medical Imaging VI : Image Processing*, volume 1652, pages 310–322.

[Clerc, 2002] Clerc, M. (2002). L'optimisation par essaim particulaire : principes, modèles et usages. *Technique et Science Informatiques*, 21 :941–964.

[Clerc, 2004] Clerc, M. (2004). *Principes de l'optimisation par essaims particulaires paramétrique et adaptative*. Hermès, à paraître edition.

[Clerc and Kennedy, 2002] Clerc, M. and Kennedy, J. (2002). The particle swarm : explosion, stability and convergence in a multi-dimensional complex space. *IEEE Transactions on Evolutionnary Computation*, 6 :58–73.

[Cobb and Grefenstette, 1993] Cobb, H. and Grefenstette, J. (1993). Genetic Algorithms for Tracking Changing Environments. In *Fifth International Conference on Genetic Algorithms*.

[Collette, 2002] Collette, Y. (2002). *Contribution à l'évaluation et au perfectionnement des méthodes d'optimisation multiobjectif. Application à l'optimisation des plans de rechargement de combustible nucléaire*. PhD thesis, Université de Paris XII Val-de-Marne.

[Collette, 2004] Collette, Y. (2004). Les nouvelles variantes du recuit simulé. In *Journée META-SCDD*, Créteil, France.

[Collette and Siarry, 2002] Collette, Y. and Siarry, P. (2002). *Optimisation multiobjectif*. Eyrolles edition.

[Colorni et al., 1992] Colorni, A., Dorigo, M., and Maniezzo, V. (1992). Distributed Optimization by Ant Colonies. In Varela, F. and Bourgine, P., editors, *Proceedings of ECAL'91 - First European Conference on Artificial Life*, pages 134–142, Paris, France. Elsevier Publishing.

[Creutz, 1983] Creutz, M. (1983). Microcanonical Monte Carlo simulation. *Physical Review Letters*, 50(19) :1411–1414.

[Dasgupta, 1999] Dasgupta, D. (1999). *Artificial Immune Systems and their applications*. Springer Verlag.

[Dasgupta and Attoh-Okine, 1997] Dasgupta, D. and Attoh-Okine, N. (1997). Immune-based systems : A survey. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 1, pages 369–374, Orlando. IEEE Press.

[De Castro and Von Zuben, 1999] De Castro, L. and Von Zuben, F. (1999). Artificial Immune Systems : Part I : Basic Theory and Applications. Technical Report TR-DCA 01/99, Department of Computer Engineering and Industrial Automation, School of Electrical and Computer Engineering, State University of Campinas, Brazil.

[De Castro and Von Zuben, 2000] De Castro, L. and Von Zuben, F. (2000). Artificial Immune Systems : Part II - A Survey of Applications. Technical Report DCA-RT 02/00, Department of Computer Engineering and Industrial Automation, School of Electrical and Computer Engineering, State University of Campinas, Brazil.

[De Wolf et al., 2002] De Wolf, T., Liesbeth, J., Holvoet, T., and Steegmans, E. (2002). A Nested Layered Threshold Model for Dynamic Task Allocation. In Dorigo, M., Di Caro, G., and Sampels, M., editors, *Proceedings of the Third International Workshop on Ant Algorithms (ANTS'2002)*, volume 2463 of *Lecture Notes in Computer Science*, pages 290–291, Brussels, Belgium. Springer Verlag.

[Di Caro and Dorigo, 1997] Di Caro, G. and Dorigo, M. (1997). AntNet : A mobile agents approach to adaptive routing. Technical Report IRIDIA/97-12, IRIDIA, Université Libre de Bruxelles, Belgium.

[Di Caro and Dorigo, 1998a] Di Caro, G. and Dorigo, M. (1998a). Ant colonies for adaptive routing in packet-switched communications networks. In Eiben, A., Bäck, T., Schoenauer, M., and Schwefel, H.-P., editors, *Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498 of *Lecture Notes in Computer Science*, pages 673–682, Berlin, Germany. Springer Verlag.

[Di Caro and Dorigo, 1998b] Di Caro, G. and Dorigo, M. (1998b). AntNet : Distributed stigmergic control for communications networks. *Journal of Artificial Intelligence Research*, 9 :317–365.

[Dorigo et al., 2002] Dorigo, M., Di Caro, G., and Sampels, M., editors (2002). *Proceedings of the Third International Workshop on Ant Algorithms (ANTS'2002)*, volume 2463 of *Lecture Notes in Computer Science*, Brussels, Belgium. Springer Verlag.

[Dorigo et al., 2004] Dorigo, M., Di Caro, G., and Sampels, M., editors (2004). *Proceedings of the Fourth International Workshop on Ant Algorithms (ANTS'2004)*, volume 2463 of *Lecture Notes in Computer Science*, Brussels, Belgium. Springer Verlag.

[Dorigo and Gambardella, 1997a] Dorigo, M. and Gambardella, L. M. (1997a). Ant Colonies for the Traveling Salesman Problem. *BioSystems*, 43 :73–81.

[Dorigo and Gambardella, 1997b] Dorigo, M. and Gambardella, L. M. (1997b). Ant Colony System : A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Trans. Evol. Comp.*, 1 :53–66.

[Dorigo et al., 1996] Dorigo, M., Maniezzo, V., and Colorni, A. (1996). The Ant System : Optimization by a Colony of Cooperating Agents. *IEEE Trans. Syst. Man Cybern*, B(26) :29–41.

[Dorigo and Stützle, 2003] Dorigo, M. and Stützle, T. (2003). *Handbook of Metaheuristics*, volume 57 of *International series in operations research and management science*,

chapter The Ant Colony Optimization Metaheuristics : Algorithms, Applications and Advances. Kluwer Academic Publishers, Boston Hardbound.

[Dorigo and Stützle, 2004] Dorigo, M. and Stützle, T. (2004). *Ant Colony Optimization*. MIT Press.

[Dréo, 2001] Dréo, J. (2001). Modélisation de la mobilisation chez les fourmis. Mémoire de DEA, Université Paris7 & Université Libre de Bruxelles.

[Dréo et al., 2003] Dréo, J., Pétrowski, A., Siarry, P., and Taillard, E. D. (2003). *Métaheuristiques pour l'optimisation difficile*. Eyrolles.

[Dréo et al., 2005] Dréo, J., Pétrowski, A., Siarry, P., and Taillard, E. D. (2005). *Metaheuristics for difficult optimization*. Springer, à paraître edition.

[Dréo and Siarry, 2001] Dréo, J. and Siarry, P. (2001). A New Ant Colony Algorithm Using Direct Interindividual Communication Aimed At Continuous Optimization. In *First Joint Seminar on Metaheuristics organized by the UK Local Search Group and EU/ME, the EUropean chapter on MEtaheuristics*, City University, London.

[Dréo and Siarry, 2002a] Dréo, J. and Siarry, P. (2002a). A New Ant Colony Algorithm Using the Heterarchical Concept Aimed at Optimization of Multiminima Continuous Functions. In Dorigo, M., Di Caro, G., and Sampels, M., editors, *Proceedings of the Third International Workshop on Ant Algorithms (ANTS'2002)*, volume 2463 of *Lecture Notes in Computer Science*, pages 216–221, Brussels, Belgium. Springer Verlag.

[Dréo and Siarry, 2002b] Dréo, J. and Siarry, P. (2002b). Un nouvel algorithme de colonie de fourmis exploitant la communication directe entre individus , application à l'optimisation en variables continues. In *4ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2002)*.

[Dréo and Siarry, 2002c] Dréo, J. and Siarry, P. (2002c). Un nouvel algorithme de colonie de fourmis exploitant le concept d'hétérarchie pour l'optimisation en variables continues. In *11èmes Journées Neurosciences et Sciences pour l'Igénieur (NSI 2002)*, La Londe Les Maures. CDROM.

[Dréo and Siarry, 2003a] Dréo, J. and Siarry, P. (2003a). Colonies de fourmis et optimisation continue. De l'utilité de l'optimisation en général et des colonies de fourmis en particulier, quand l'éthologie et l'informatique se croisent. In *Séminaire de recherche de l'ISBS-Paris*, Créteil. Université Paris 12.

[Dréo and Siarry, 2003b] Dréo, J. and Siarry, P. (2003b). Diverses techniques d'optimisation inspirées de la théorie de l'auto-organisation dans les systèmes biologiques. In *Journée optimisation par essaim particulaire (OEP'2003)*.

[Dréo and Siarry, 2003c] Dréo, J. and Siarry, P. (2003c). Diverses techniques d'optimisation inspirées de la théorie de l'auto-organisation dans les systèmes biologiques. *soumis à Technique et Science Informatiques (revue Hermès).*

[Dréo and Siarry, 2003d] Dréo, J. and Siarry, P. (2003d). Un algorithme de colonie de fourmis en variables continues hybridé avec un algorithme de recherche locale. In *5ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2003)*, Avignon, France.

[Dréo and Siarry, 2004a] Dréo, J. and Siarry, P. (2004a). Algorithmes à estimation de distribution et colonies de fourmis. In *11ème journée évolutionnaire (JET11).*

[Dréo and Siarry, 2004b] Dréo, J. and Siarry, P. (2004b). An Ant Colony Algorithm aimed at Dynamic Continuous Optimization. *soumis à Computers & Industrial Engineering.*

[Dréo and Siarry, 2004c] Dréo, J. and Siarry, P. (2004c). Continuous Interacting Ant Colony Algorithm Based on Dense Heterarchy. *Future Generation Computer Systems*, 20(5) :841–856.

[Dréo and Siarry, 2004d] Dréo, J. and Siarry, P. (2004d). Hybrid Continuous Interacting Ant Colony aimed at enhanced global optimization. *soumis au Journal of Mathematical Modelling and Algorithms, special issue on Hybrid Metaheuristics.*

[Eberhart et al., 2001] Eberhart, R., Kennedy, J., and Shi, Y. (2001). *Swarm Intelligence.* Evolutionary Computation. Morgan Kaufmann.

[Ferber, 1997] Ferber, J. (1997). *Les systèmes multi-agents. Vers une intelligence collective.* InterEditions.

[Fogel et al., 1966] Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artifical Intelligence through Simulated Evolution.* Wiley.

[Fonseca and Fleming, 1993] Fonseca, C. M. and Fleming, P. J. (1993). Genetic Algorithms for Multiobjective Optimization : Formulation, Discussion and Generalization. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423, San Mateo, California.

[Fraser, 1957] Fraser, A. S. (1957). Simulation of genetic systems by automatic digital computers. *Australian Journal of Biological Sciences*, 10 :484–491.

[Gambardella and Dorigo, 2000] Gambardella, L. and Dorigo, M. (2000). Ant Colony System hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, 12(3) :237–255.

[Gambardella et al., 1999] Gambardella, L., Taillard, E., and Agazzi, G. (1999). *New Ideas in Optimization*, chapter MACS-VPTW : A multiple ant colony system for vehicle routing problems with time windows, pages 63–67. McGraw Hill, London, UK.

[Gambardella and Dorigo, 1995] Gambardella, L. M. and Dorigo, M. (1995). Ant-Q : A Reinforcement Learning Approach to the Travelling Salesman Problem. In *Proceedings Twelfth International Conference on Machine Learning*, volume ML-95, pages 252–260, Palo Alto. Morgan Kaufmann.

[Gaspar and Collard, 1999] Gaspar, A. and Collard, P. (1999). From GAs to artificial immune systems : improving adaptation in time dependent optimization. In Angeline, P., Michalewicz, Z., Schoenauer, M., Yao, X., and Zalzala, A., editors, *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 1859–1866, Washington D.C.

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

[Glover and Kochenberger, 2003] Glover, F. W. and Kochenberger, G. A. (2003). *Handbook of Metaheuristics*, volume 57 of *International series in operations research and management science*. Kluwer Academic Publishers, Boston Hardbound.

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine learning*. Addison-Wesley.

[Goldberg, 1994] Goldberg, D. E. (1994). *Algorithmes génétiques. Exploration, optimisation et apprentissage automatique*. Addison-Wesley France.

[Goldberg, 1995] Goldberg, D. E. (1995). *Algorithmes génétiques, exploration, optimisation et apprentissage automatique.* Paris. Addison Wesley.

[Goss et al., 1989] Goss, S., Aron, S., Deneubourg, J. L., and Pasteels, J. M. (1989). Self-Organized Shortcuts in the Argentine Ant. *Naturwissenchaften*, 76 :579–581.

[Grassé, 1959] Grassé, P.-P. (1959). La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp. La théorie de la stigmergie : essai d'interprétation du comportement des termites constructeurs. *Insectes sociaux*, 6 :41–83.

[Grefenstette, 1986] Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. *IEEE transactions on Systems, MAN, and Cybernetics*, 16(1) :122–128.

[Gutjahr, 2000] Gutjahr, W. J. (2000). A graph-based Ant System and its convergence. *Future Generation Computer Systems*, 16(8) :873–888.

[Gutjahr, 2002] Gutjahr, W. J. (2002). ACO algorithms with guaranted convergence to the optimal solution. *Information Processing Letters*, 82(3) :873–888.

[Hampson and Pesquet, 2000] Hampson, F. J. and Pesquet, J. C. (2000). Motion estimation in the presence of illumination variations. *Signal Processing : Image Communication*, 16 :373–381.

[Harik, 1999] Harik, G. (1999). Linkage learning in via probabilistic modeling in the EcGA. Technical Report 99010, IlliGAL.

[Harik et al., 1998] Harik, G., Lobo, F. G., and Goldberg, D. E. (1998). The compact genetic algorithm. In *IEEE Conference on Evolutionnary Computation*, pages 523–528.

[Hart and Goldbaum, 1994] Hart, W. E. and Goldbaum, M. H. (1994). Registering retinal images using automatically selected control point pairs. In *IEEE International Conference on Image Processing*.

[Hastings, 1970] Hastings, W. K. (1970). Monte Carlo sampling method using Markov chains and their applications. *Biometrika*, 57.

[Hewitt, 1977] Hewitt, C. (1977). Viewing Control Structures as Patterns of Passing Messages. *Journal of Artificial Intelligence*, 8(3) :323–364.

[Hill et al., 2001] Hill, D. L. G., Batchelor, P. G., Holden, M., and Hawkes, D. J. (2001). Medical Image Registration. *Physics in Medicine and Biology*, 46 :1–45.

[Holland, 1962] Holland, J. H. (1962). Outline for logical theory of adaptive systems. *J. Assoc. Comput. Mach.*, 3 :297–314.

[Hölldobler and Wilson, 1990] Hölldobler, B. and Wilson, E. (1990). *The Ants*. Springer Verlag.

[Hukushima and Nemoto, 1996] Hukushima, K. and Nemoto, K. (1996). Exchange Monte Carlo method and application to spin glass simulations. *J. Phys. Soc. Jpn.*, 65 :1604–1608.

[Iba, 2003] Iba, Y. (2003). Population Annealing : An approach to finite-temperature calculation. In *Joint Workshop of Hayashibara Foundation and SMAPIP*. Hayashibara Forum.

[Irani et al., 1994] Irani, M., Rousso, B., and Peleg, S. (1994). Computing occluding and transparent motion. *IJCV*, 12 :5–16.

[Jenkinson and Smith, 2001] Jenkinson, M. and Smith, S. (2001). A global optimisation method for robust affine registration of brain images. *Medical Image Analysis*, 5 :143–156.

[Jennings, 1996] Jennings, N. R. (1996). Coordination Techniques for Distributed Artificial Intelligence. In G. M. P. O'Hare and N. R. Jennings, editor, *Foundations of Distributed Artificial Intelligence*, pages 187–210. John Wiley & Sons.

[Kennedy, 2000] Kennedy, J. (2000). Stereotyping : Improving particle Swarm performances with cluster analysis. In *Proceedings of the 2000 Congress on Evolutionnary Computation*, pages 1507–1512, Piscataway. IEEE Service Center.

[Kennedy and Eberhart, 1995] Kennedy, J. and Eberhart, R. C. (1995). Particle swarm optimization. In *Proc. IEEE Int. Conf. on Neural Networks*, volume IV, pages 1942–1948, Piscataway, NJ : IEEE Service Center.

[Kim et al., 2001] Kim, M., Jeon, J. C., Kwak, J. S., Lee, M. H., and Ahn, C. (2001). Moving object segmentation in video sequences by user interaction and automatic object tracking. *Image and Vision Computing*, 19 :245–260.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C., and Vecchi, M. (1983). Optimization by simulated annealing. *Science*, 220(4598) :671–680.

[Krauth, 1998] Krauth, W. (1998). *Advances in Computer Simulation*, chapter Introduction To Monte Carlo Algorithms. Springer-Verlag.

[Larrañaga et al., 1999] Larrañaga, P., Etxeberria, R., Lozano, J. A., and Peña, J. M. (1999). Optimization by learning and simulation of Bayesian and Gaussian networks. Technical Report KZZA-IK-4-99, Departement of Computer Science and Artificial Intelligence, University of the Basque Country.

[Larrañaga et al., 2000] Larrañaga, P., Etxeberria, R., Lozano, J. A., and Peña, J. M. (2000). Optimization by learning and simulation of Bayesian and Gaussian networks. In Wu, S. A., editor, *Proceedings of the 200 Genetic and Evolutionary Computation Conference Workshop Program*, pages 201–204.

[Larrañaga and Lozano, 2002] Larrañaga, P. and Lozano, J. (2002). *Estimation of Distribution Algorithms, A New Tool for Evolutionary Computation.* Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers.

[Larrañaga et al., 2001] Larrañaga, P., Lozano, J. A., and Bengoetxea, E. (2001). Estimation of Distribution Algorithms based on multivariate normal and Gaussian networks. Technical Report KZZA-IK-1-01, Departement of Computer Science and Artificial Intelligence, University of the Basque Country.

[Liang and Wong, 2000] Liang, F. and Wong, W. H. (2000). Evolutinnary Monte Carlo : Application to C_p Model Sampling and Change Point Theorem. *Statistica Sinica*, 10.

[Ling et al., 2002] Ling, C., Jie, S., Ling, Q., and Hongjian, C. (2002). A Method for Solving Optimization Problems in Continuous Space Using Ant Colony Algorithm. In Dorigo, M., Di Caro, G., and Sampels, M., editors, *Proceedings of the Third International Workshop on Ant Algorithms (ANTS'2002)*, volume 2463 of *Lecture Notes in Computer Science*, pages 288–289, Brussels, Belgium. Springer Verlag.

[Maintz and Viergerver, 1998] Maintz, J. B. A. and Viergerver, M. A. (1998). A survey of medical image registration. *Med. Image Anal.*, 2 :1–36.

[Mathur et al., 2000] Mathur, M., Karale, S. B., Priye, S., Jyaraman, V. K., and Kulkarni, B. D. (2000). Ant Colony Approach to Continuous Function Optimization. *Ind. Eng. Chem. Res.*, 39 :3814–3822.

[Meinhardt, 1982] Meinhardt, H. (1982). *Models of Biological Pattern Formation*. Academic Press, London.

[Merkle et al., 2000] Merkle, D., Middendorf, M., and Schmek, H. (2000). Ant Colony Optimization for resource-constrained project scheduling. In *Proceedings of the Genetic and Evolutionnary Computation Conference (GECCO-2000)*, pages 839–900, San Francisco, CA. Morgan Kaufmann Publishers.

[Metropolis et al., 1953] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21 :1087–1092.

[Monmarché et al., 1999] Monmarché, N., Ramat, E., Dromel, G., Slimane, M., and Venturini, G. (1999). On the similarities between AS, BSC and PBIL : toward the birth of a new meta-heuristics. E3i 215, Université de Tours.

[Monmarché et al., 2000a] Monmarché, N., Ramat, N., Desbarat, L., and Venturini, G. (2000a). Probabilistic search with genetic algorithms and ant colonies. In Wu, A., editor, *Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop*, pages 209–211.

[Monmarché et al., 2000b] Monmarché, N., Venturini, G., and Slimane, M. (2000b). On how Pachycondyla apicalis ants suggest a new search algorithm. *Future Generation Computer Systems*, 16 :937–946.

[Morrison and De Jong, 1999] Morrison, R. and De Jong, K. (1999). A Test Problem Generator for Non-Stationary Environments. In *Congress on Evolutionary Computing*.

[Mühlenbein and Paaβ, 1996] Mühlenbein, H. and Paaβ, G. (1996). From recombination of genes to the estimation of distributions I. Binary parameters. *Lecture Notes in Computer Science 1411 : Parallel Problem Solving from Nature*, PPSN IV :178–187.

[Mukhopadhyay and Chanda, 2001] Mukhopadhyay, S. and Chanda, B. (2001). Fusion of 2D grayscale images using multiscale morphology. *Pattern Recognition*, 34 :606–619.

[Nelder and Mead, 1965] Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, 7 :308–313.

[Nicolis and Prigogine, 1977] Nicolis, G. and Prigogine, I. (1977). *Self-organization in Non-equilibrium Systems*. New York.

[Nouyan, 2002] Nouyan, S. (2002). Agent-Based Approach to Dynamic task Allocation. In Dorigo, M., Di Caro, G., and Sampels, M., editors, *Proceedings of the Third Inter-*

national Workshop on Ant Algorithms (ANTS'2002), volume 2463 of *Lecture Notes in Computer Science*, pages 28–39, Brussels, Belgium. Springer Verlag.

[Nunes, 2003] Nunes, J.-C. (2003). *Analyse multiéchelle d'image. Application à l'angiographie rétinienne et à la DMLA.* PhD thesis, Université Paris 12.

[Nunes et al., 2004a] Nunes, J. C., Bouaoune, Y., Deléchelle, E., and Bunel, P. (2004a). A multiscale elastic registration scheme for retinal angiograms. *Computer Vision and Images Understanding.*

[Nunes et al., 2004b] Nunes, J.-C., Dréo, J., and Siarry, P. (2004b). Rigid registration of retinal angiograms through Nelder-Mead optimization. In *International Workshop on Electronics and System Analysis, IWESA'04.*

[Odobez and Bouthemy, 1994] Odobez, J. M. and Bouthemy, P. (1994). Robust multiresolution estimation of parametric motion models applied to complex scenes. Technical Report 788, IRISA.

[Okamoto and Hansmann, 1995] Okamoto, Y. and Hansmann, U. H. E. (1995). Thermodynamics of helix-coil transitions studied by multicanonical algorithms. *J. Phys. Chem.*, 99 :11276–11287.

[Ourique et al., 2002] Ourique, C., Biscaia, E. J., and Pinto, J. (2002). The use of particle swarm optimization for dynamical analysis in chemical processes. *Computers & Chemical Engineering*, 26(12) :1783–1793.

[Panta, 2002] Panta, L. (2002). *Modeling Transportation Problems Using Concepts of Swarm Intelligence and Soft Computing.* PhD thesis, Virginia Tech, Blacksburg, Virginia.

[Pardalos et al., 1996] Pardalos, P., Xue, G., and Panagiotopoulos, P. D. (1996). Parallel Algorithms for Global Optimization Problems. In *Solving Combinatorial Optimization Problems in Parallel*, pages 232–247.

[Pham and Karaboga, 2000] Pham, D. and Karaboga, D. (2000). *Intelligent optimisation techniques. Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks.* Springer.

[Pinz et al., 1998] Pinz, A., Bernögger, S., Datlinger, P., and Kruger, A. (1998). Mapping the human retina. *IEEE Trans. on Med. Imag.*, 17 :606–619.

[Pourtakdoust and Nobahari, 2004] Pourtakdoust, S. H. and Nobahari, H. (2004). An Extension of Ant Colony System to Continuous Problems. In Dorigo, M., Birattari, M., Blum, C., Gambardella, L. M., Mondada, F., and Stützle, T., editors, *Ant Colony Optimization and Swarm Intelligence*, volume 3172 of *Lecture Notes in Computer Science*, pages 294–301. Springer.

[Prigogine and Glandsdorf, 1971] Prigogine, I. and Glandsdorf, P. (1971). *Thermodynamic Theory and Structure, Stability and Fluctuations*. Wiley and Sons, New York.

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

[Reeves, 1995] Reeves, C. (1995). *Modern Heuristic Techniques for Combinatorial Problems*. Advances topics in computer science. Mc Graw-Hill.

[Resende, 2000] Resende, M. (2000). Greedy randomized adaptive search procedures (GRASP). Technical Report TR 98.41.1, AT&T Labs-Research.

[Richard et al., 1998] Richard, G., Soubrane, G., and Yannuzzi, L. A. (1998). Fluorescein and ICG angiography. *Thieme*.

[Ritter et al., 1999] Ritter, N., Owens, R., Cooper, J., Eikelboom, R. H., and Saarloos, P. P. V. (1999). Registration of stereo and temporal images of the retina. *IEEE Trans. On Medical Imaging*, 18 :404–418.

[Roche et al., 1999] Roche, A., Malandain, G., Ayache, N., and Prima, S. (1999). Towards a better comprehension of similarity measures used in medical image registration. In *MICCAI*.

[Rudolph, 1992] Rudolph, G. (1992). *Parallel approaches to stochastic global optimization*, pages 256–267. IOS Press, Amsterdam.

[Saït and Youssef, 1999] Saït, S. and Youssef, H. (1999). Iterative computer algorithms with applications in engineering. *IEEE Computer Society Press*.

[Schoonderwoerd et al., 1996] Schoonderwoerd, R., Holland, O., Brutent, J., and Rothkrantz, L. (1996). Ant-based load balancing in telecommunication networks. *Adaptive Behavior*, 5(2) :169–207.

[Sebag and Ducoulombier, 1998] Sebag, M. and Ducoulombier, A. (1998). Extending population-based incremental learning to continuous search spaces. In *Parallel Problem Solving from Nature - PPSN V*, pages 418–427. Springer-Verlag.

[Seeley, 1989] Seeley, T. D. (1989). The honey bee colony as a superorganism. *American Scientist*, 77 :546–553.

[Shi and Eberhart, 1998] Shi, Y. and Eberhart, R. (1998). A modified particle swarm optimizer. In *Proceedings of the IEEE International Conference on Evolutionnary Computation*, pages 69–73, Piscataway. IEEE Press.

[Shi and Eberhart, 1999] Shi, Y. and Eberhart, R. (1999). Empirical study of particle swarm optimization. In *Proceedings of the 1999 Congress on Evolutionnary Computation*, pages 1945–1950, Piscataway. IEEE Service Center.

[Siarry and Dreyfus, 1989] Siarry, P. and Dreyfus, G. (1989). *La méthode du recuit simulé : théorie et applications*. ESPCI – IDSET, 10 rue Vauquelin, 75005 Paris.

[Simo and de Ves, 2001] Simo, A. and de Ves, E. (2001). Segmentation of macular fluorescein angiographies, a statistical approach. *Pattern Recognition*, 34 :795–809.

[Socha, 2004] Socha, K. (2004). ACO for Continuous and Mixed-Variable Optimization. In Dorigo, M., Birattari, M., Blum, C., Gambardella, L. M., Mondada, F., and Stützle, T., editors, *Ant Colony Optimization and Swarm Intelligence*, volume 3172 of *Lecture Notes in Computer Science*, pages 25–36. Springer.

[Storn and Price, 1995] Storn, R. and Price, K. (1995). Differential Evolution – A simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR95 -012, International Computer Science Institute, Berkeley, California.

[Stützle and Hoos, 1997] Stützle, T. and Hoos, H. (1997). Improvements on the Ant System : Introducing MAX-MIN Ant System. In *Proceedings International Conference on Artificial Neural Networks and Genetic Algorithms*, Vienna. Springer-Verlag.

[Stützle and Hoos, 2000] Stützle, T. and Hoos, H. (2000). MAX-MIN Ant System. *Future Generation Computer System*, 16 :889–914.

[Syswerda, 1993] Syswerda, G. (1993). Simulated Crossover in Genetic Algorithms. In Whitley, L. D., editor, *Second workshop on Foundations of Genetic Algorithms*, pages 239–255, San Mateo, California. Morgan Kaufmann.

[Taillard, 1998] Taillard, E. D. (1998). *Programmation à mémoire adaptative et algorithmes pseudo-gloutons : nouvelles perspectives pour les méta-heuristiques*. Thèse d'habilitation à diriger les recherches, Université de Versailles Saint Quentin en Yvelines, France.

[Taillard et al., 1998] Taillard, E. D., Gambardella, L. M., Gendreau, M., and Potvin, J.-Y. (1998). Adaptive Memory Programming : A Unified View of Meta-Heuristics. *European Journal of Operational Research*, 135(1) :1–16.

[Talbi, 2002] Talbi, E.-G. (2002). A Taxonomy of Hybrid Metaheuristics. *Journal of Heuristics*, 8(5) :541–564.

[Teghem and Pirlot, 2002] Teghem, J. and Pirlot, M. (2002). *Optimisation approchée en recherche opérationnelle. Recherches locales, réseaux neuronaux et satisfaction de contraintes*. Hermès.

[Theraulaz and Bonabeau, 1995] Theraulaz, G. and Bonabeau, E. (1995). Coordination in distributed building. *Science*, 269 :686–688.

[Theraulaz et al., 1998] Theraulaz, G., Bonabeau, E., and Deneubourg, J.-L. (1998). Response Threshold Reinforcement and Division of Labour in Insect Societies. In *Proceedings of the Royal Society of London B*, volume 265, pages 327–335.

[Triki et al., 2004] Triki, E., Collette, Y., and Siarry, P. (2004). A theoretical study on the behavior of simulated annealing leading to a new cooling schedule. *European Journal of Operational Research*, to appear.

[Wendt and König, 1997] Wendt, O. and König, W. (1997). Cooperative Simulated Annealing : How Much Cooperation is Enough ? Technical Report 97-19, Institute of Information Systems, Goethe University, Frankfurt.

[White et al., 1998] White, T., Pagurek, B., and Oppacher, F. (1998). Connection Management using Adaptive Agents. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, pages 802–809.

[Wilson, 1984] Wilson, E. (1984). The relation between Caste Ratios and Division of Labour in the Ant Genus Pheidole (Hymenoptera : Formicidae). *Behav. Ecol. Sociobiol.*, 16 :89–98.

[Wilson and Hölldobler, 1988] Wilson, E. and Hölldobler, B. (1988). Dense Heterarchy and mass communication as the basis of organization in ant colonies. *Trend in Ecology and Evolution*, 3 :65–68.

[Wodrich and Bilchev, 1997] Wodrich, M. and Bilchev, G. (1997). Cooperative distributed search : the ant's way. *Control and Cybernetics*, 26(3).

[Xiong and Jutan, 2003] Xiong, Q. and Jutan, A. (2003). Continuous optimization using a dynamic simplex method. *Chemical Engineering Science*, 58(16) :3817–3828.

[Yip and Marlin, 2003a] Yip, W. and Marlin, T. (2003a). Designing plant experiments for real-time optimization systems. *Control Engineering Practice*, 11(8) :837–845.

[Yip and Marlin, 2003b] Yip, W. and Marlin, T. (2003b). The effect of model fidelity on real-time optimization performance. *Computers & Chemical Engineering*, 28(1–2) :267–280.

[Yu et al., 1989] Yu, J. J.-H., Hung, B.-N., and Liou, C.-L. (1989). Fast algorithm for digital retinal image alignment. In *IEEE Ann. Int. Conf. Engineering Medicine Biology Society, Images Twenty-First Century*, volume 2, pages 374–375.

[Zana and Klein, 1999] Zana, F. and Klein, J. C. (1999). A multimodal registration algorithm of eye fundus images using vessels detection and Hough transform. *IEEE Trans. on Medical Imaging*, 18 :419–458.

[Zhang and Blum, 2001] Zhang, Z. and Blum, R. S. (2001). A hybrid image registration technique for a digital camera image fusion application. *Information fusion*, 2 :135–149.