Conception de métaheuristiques pour l'optimisation dynamique : application à l'analyse de séquences d'images IRM

Julien Lepagnot

UNIVERSITE PARIS-EST

ÉCOLE DOCTORALE MATHEMATIQUES ET STIC (MSTIC, E.D. 532)

THESE DE DOCTORAT EN INFORMATIQUE

par

Julien LEPAGNOT

Sujet de la thèse :

Conception de métaheuristiques pour l'optimisation dynamique. Application à l'analyse de séquences d'images IRM.

Thèse dirigée par le Professeur Patrick SIARRY

Soutenue le 1er décembre 2011

Jury :

Xavier GANDIBLEUX	Professeur des Universités	Université de Nantes	Président
Christian PRINS	Professeur des Universités	Université de Technologie de Troyes	Rapporteur
Su RUAN	Professeur des Universités	Université de Rouen	Rapporteur
Damien TRENTESAUX	Professeur des Universités	Université de Valenciennes et du Hainaut-Cambrésis	Examinateur
Amir NAKIB	Maître de Conférences	Université Paris-Est Créteil	Examinateur
Hamouche OULHADJ	Maître de Conférences	Université Paris-Est Créteil	Examinateur
Patrick SIARRY	Professeur des Universités	Université Paris-Est Créteil	Directeur de thèse

Remerciements

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

Je tiens à exprimer ma gratitude à Christian Prins et à Su Ruan pour avoir accepté d'être les rapporteurs de cette thèse. Je voudrais également remercier Xavier Gandibleux pour avoir accepté de présider mon jury. Je remercie aussi Damien Trentesaux pour avoir accepté d'examiner mes travaux. J'adresse un grand merci à tous les membres de mon jury, pour avoir ainsi marqué leur intérêt pour mon travail, et pour les remarques qu'ils ont apportées durant la relecture et la soutenance de ma thèse.

Je souhaite également exprimer ma reconnaissance envers tous les membres du LiSSi, pour m'avoir accueilli chaleureusement, et pour toutes les conversations scientifiques ou non que l'on a pu avoir. Un grand merci à Patricia Jamin, Sandrine David et Frédéric Dumont, pour m'avoir aidé à surmonter toutes sortes de problèmes. Je n'oublie pas non plus Michel Noum et Audray Carotine, pour leur accueil et leurs conseils en début de thèse. Je remercie également Brigitte David, de l'ancienne Ecole Doctorale SIMME, et Sylvie Cach, de la nouvelle ED MSTIC.

Je remercie tous mes collègues doctorants, pour la bonne ambiance et leur amitié. Merci en particulier à Abbas El Dor, Ilhem Boussaïd et Mustapha Dakkak. Un grand merci à vous trois, pour tous les bons moments que nous avons eus au sein et à l'extérieur du labo.

Il y a aussi toutes les rencontres faites au détour d'une conférence ou d'une formation. Un grand merci à Nathan Krislock pour son aide, sa grande gentillesse et son amitié. Merci également à Patrick Cansell pour sa patience et ses conseils avisés.

J'ai eu la chance de faire un monitorat pendant ma thèse à l'Université d'Evry-Val-d'Essonne. J'ai beaucoup aimé enseigner à l'université, et je souhaite remercier toute l'équipe enseignante

avec qui j'ai eu l'honneur de travailler. Merci notamment à Guillaume Hutzler, Jean-Marc Delosme et Franck Delaplace, que j'ai tout d'abord connus lorsque je suivais leurs cours en tant qu'étudiant. Je remercie aussi Sylvain Sené, Mohamed Elati, Jean-Yves Didier et Serenella Cerrito. Merci également à Romain Campigotto, qui a su me recevoir avec courtoisie lorsque je faisais irruption dans son bureau pour lui demander un service.

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

Ce travail n'aurait pas pu être réalisé sans le soutien de ma famille, que je remercie tout particulièrement. Un grand merci à mes parents et à mon frère, toujours présents lorsque j'ai besoin d'eux.

Et enfin, comme il y a sans doute des trous dans mon énumération, je conclurai par ces quelques mots, merci à tous !

Résumé

Dans la pratique, beaucoup de problèmes d'optimisation sont dynamiques : leur fonction objectif (ou fonction de coût) évolue au cours du temps. L'approche principalement adoptée dans la littérature consiste à adapter des algorithmes d'optimisation statique à l'optimisation dynamique, en compensant leurs défauts intrinsèques. Plutôt que d'emprunter cette voie, déjà largement explorée, l'objectif principal de cette thèse est d'élaborer un algorithme entièrement pensé pour l'optimisation dynamique. La première partie de cette thèse est ainsi consacrée à la mise au point d'un algorithme, qui doit non seulement se démarquer des algorithmes concurrents par son originalité, mais également être plus performant. Dans ce contexte, il s'agit de développer une métaheuristique d'optimisation dynamique.

Deux algorithmes à base d'agents, MADO (*MultiAgent algorithm for Dynamic Optimization*) et MLSDO (*Multiple Local Search algorithm for Dynamic Optimization*), sont proposés et validés sur les deux principaux jeux de tests existant dans la littérature en optimisation dynamique : MPB (*Moving Peaks Benchmark*) et GDBG (*Generalized Dynamic Benchmark Generator*).

Les résultats obtenus sur ces jeux de tests montrent l'efficacité des stratégies mises en œuvre par ces algorithmes, en particulier :

- MLSDO est classé premier sur sept algorithmes évalués sur GDBG ;
- MLSDO est classé deuxième sur seize algorithmes évalués sur MPB.

Ensuite, ces algorithmes sont appliqués à des problèmes pratiques en traitement de séquences d'images médicales (segmentation et recalage de séquences ciné-IRM cérébrales). A notre connaissance, ce travail est innovant, en ce sens que l'approche de l'optimisation dynamique n'avait jamais été explorée pour ces problèmes. Les gains de performance obtenus montrent l'intérêt d'utiliser les algorithmes d'optimisation dynamique proposés pour ce type d'applications.

Mots clés : *optimisation, optimisation continue, optimisation dynamique, métaheuristiques, multi-agent, recherche locale, segmentation, seuillage, recalage, séquences d'images, IRM.*

Abstract

Many real-world problems are dynamic, i.e. their objective function (or cost function) changes over time. The main approach used in the literature is to adapt static optimization algorithms to dynamic optimization, compensating for their intrinsic defects. Rather than adopting this approach, already widely investigated, the main goal of this thesis is to develop an algorithm completely designed for dynamic optimization. The first part of this thesis is then devoted to the design of an algorithm, that should not only stand out from competing algorithms for its originality, but also perform better. In this context, our goal is to develop a dynamic optimization metaheuristic.

Two agent-based algorithms, MADO (*MultiAgent algorithm for Dynamic Optimization*) and MLSDO (*Multiple Local Search algorithm for Dynamic Optimization*), are proposed and validated using the two main benchmarks available in dynamic environments : MPB (*Moving Peaks Benchmark*) and GDBG (*Generalized Dynamic Benchmark Generator*).

The benchmark results obtained show the efficiency of the proposed algorithms, particularly :

- MLSDO is ranked at the first place among seven algorithms tested using GDBG ;
- MLSDO is ranked at the second place among sixteen algorithms tested using MPB.

Then, these algorithms are applied to real-world problems in medical image sequence processing (segmentation and registration of brain cine-MRI sequences). To our knowledge, this work is innovative in that the dynamic optimization approach had never been investigated for these problems. The performance gains obtained show the relevance of using the proposed dynamic optimization algorithms for this kind of applications.

Keywords : *optimization, continuous optimization, dynamic optimization, metaheuristics, multi-agent, local search, segmentation, thresholding, registration, image sequences, MRI.*

Table des matières

Introduction générale

Les problèmes d'optimisation occupent actuellement une place grandissante dans la communauté scientifique. Ces problèmes peuvent être combinatoires (discrets) ou à variables continues, avec un seul ou plusieurs objectifs (optimisation mono ou multi-objectif), statiques ou dynamiques, avec ou sans contraintes. Cette liste n'est pas exhaustive et un problème peut être, par exemple, à la fois continu et dynamique.

Un problème d'optimisation est défini par un ensemble de variables, une fonction objectif (ou fonction de coût) et un ensemble de contraintes. L'espace de recherche est l'ensemble des solutions possibles du problème. Il possède une dimension pour chaque variable. Pour des raisons pratiques et de temps de calcul, l'espace de recherche des méthodes de résolution est en général fini. Cette dernière limitation n'est pas gênante, puisqu'en général le décideur précise exactement le domaine de définition de chaque variable. La fonction objectif définit le but à atteindre, on cherche à minimiser ou à maximiser celle-ci. L'ensemble des contraintes est en général un ensemble d'égalités et d'inégalités que les variables doivent satisfaire. Ces contraintes limitent l'espace de recherche.

Les méthodes d'optimisation recherchent une solution, ou un ensemble de solutions, dans l'espace de recherche, qui satisfont l'ensemble des contraintes et qui minimisent, ou maximisent, la fonction objectif. Parmi ces méthodes, les *métaheuristiques* sont des algorithmes génériques d'optimisation : leur but est de permettre la résolution d'une large gamme de problèmes différents, sans nécessiter de changements profonds dans l'algorithme. Elles forment une famille d'algorithmes visant à résoudre des problèmes d'optimisation difficile, pour lesquels on ne connaît pas de méthode classique plus efficace. Les métaheuristiques s'inspirent généralement d'analogies avec la physique (recuit simulé), avec la biologie (algorithmes évolutionnaires) ou encore l'éthologie (colonies de fourmis, essaims particulaires). Toutes sortes d'extensions ont été proposées pour ces algorithmes, notamment en optimisation dynamique.

L'optimisation dynamique s'efforce de minimiser ou maximiser une fonction objectif qui varie en fonction du temps. En pratique, l'optimisation dynamique peut être appliquée, par exemple, pour déterminer de bonnes manœuvres dans le domaine aéronautique ; pour le contrôle de robots, de réactions chimiques ; pour le routage dans les réseaux, etc.

Par rapport à l'optimisation statique, des difficultés supplémentaires apparaissent. Par exemple, les informations que l'algorithme a pu accumuler sur le problème, au cours de son exécution, peuvent être « périmées » à l'issue d'un changement dans la fonction objectif. Un moyen simple pour résoudre un problème dynamique consiste à redémarrer un algorithme d'optimisation statique, à chaque fois qu'un changement se produit dans la fonction objectif (sous réserve, entre autres, de pouvoir déterminer l'instant du changement). En pratique, procéder ainsi n'est pas toujours possible et peut s'avérer inefficace. Des algorithmes dédiés à l'optimisation dynamique ont alors été proposés dans la littérature.

La plupart des métaheuristiques d'optimisation dynamique proposées dans la littérature sont bioinspirées. Elles appartiennent principalement à la classe des algorithmes évolutionnaires et à celle des essaims particulaires. Néanmoins, ce type d'algorithmes, conçus initialement pour l'optimisation statique, ne peut pas être employé directement en optimisation dynamique. En effet, dans le cadre de l'optimisation dynamique, ces algorithmes présentent plusieurs défauts intrinsèques. Il est alors nécessaire d'utiliser des techniques particulières, afin de les adapter aux problèmes dynamiques.

Parmi les principales techniques proposées dans la littérature, certaines consistent à introduire ou à maintenir la diversité des solutions testées à chaque itération de l'algorithme (pour les algorithmes faisant évoluer une population de solutions). Il ne s'agit toutefois pas de maintenir l'algorithme dans une phase de diversification continuelle (i.e. l'algorithme doit rester capable de converger avec précision). Dans le même ordre d'idées, des techniques visant à diviser la population de solutions en plusieurs sous-populations, réparties dans l'espace de recherche, permettent de suivre plusieurs optima à la fois et d'augmenter la probabilité d'en trouver de nouveaux. D'autres techniques sont basées sur l'utilisation d'informations sur les états passés de la fonction objectif, dans le but d'accélérer la convergence de l'algorithme au fil des changements du problème. Cependant, ces techniques ne sont utiles que si les changements ne sont pas drastiques (i.e. la fonction objectif ne doit pas être complètement transformée après un changement). Enfin, des techniques cherchant à prédire les futurs changements dans la fonction objectif ont vu le jour récemment. Elles nécessitent toutefois que les changements suivent un certain schéma, pouvant être appris.

Plutôt que de recourir à ces adaptations, notre contribution va s'attacher à proposer une nouvelle métaheuristique, entièrement pensée pour l'optimisation dynamique. Afin de proposer des stratégies innovantes et bien adaptées aux problèmes d'optimisation dynamique, il est toutefois possible de s'inspirer des techniques de base. L'objectif est d'élaborer un algorithme original, dans lequel chaque composante doit constituer un atout important pour la résolution

de problèmes dynamiques. Ainsi, l'algorithme proposé doit non seulement se démarquer des algorithmes concurrents par son originalité, mais également être plus performant. En outre, ce travail vient enrichir les connaissances dans le domaine de l'optimisation dynamique, afin de contribuer à l'amélioration continuelle des algorithmes existants, et à la proposition de nouveaux algorithmes et stratégies.

A l'issue de ce travail, deux algorithmes ont été mis au point, le second constituant un nouvel algorithme plus performant, basé sur le premier. Le principe de ces algorithmes est d'utiliser une population d'agents effectuant des recherches locales en permanence, pour explorer l'espace de recherche. Ces agents sont coordonnés par un module dédié (le coordinateur), permettant d'explorer au mieux l'espace de recherche et de garantir la diversification des solutions. Les meilleurs optima locaux trouvés sont stockés en mémoire, afin d'accélérer la convergence de l'algorithme et de suivre les optima chaque fois qu'un changement est détecté dans la fonction objectif. La structure générale de ces algorithmes est illustrée à la figure 1, où les agents de recherche locale sont représentés par des disques noirs numérotés dans l'espace de recherche S, et le voisinage du $i^{\text{ème}}$ agent est noté N_i.

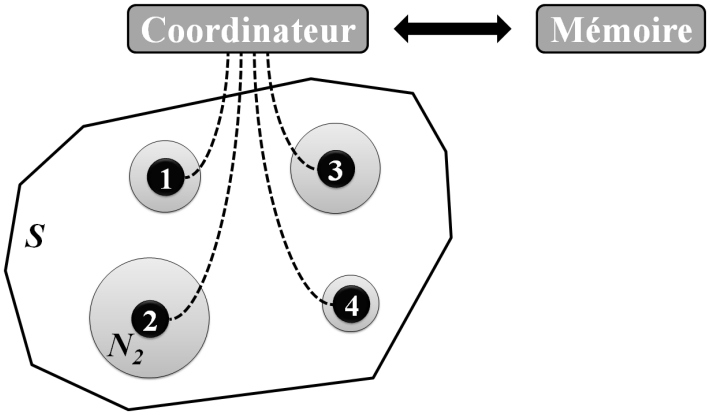


FIGURE 1 – Schéma général du fonctionnement des algorithmes d'optimisation dynamique élaborés dans cette thèse.

Afin d'élaborer ces algorithmes, il nous a fallu en premier lieu identifier les principaux jeux de tests en optimisation dynamique, utilisés dans la littérature. Le rôle de ces jeux de tests est de permettre l'analyse et la comparaison des algorithmes à mettre au point. Les performances des algorithmes proposés dans cette thèse sont donc tout d'abord évaluées sur ces jeux de tests, puis sur des problèmes réels de segmentation et de recalage de séquences d'images ciné-IRM cérébrales. Ces derniers sont abordés comme des problèmes d'optimisation dynamique, ce qui constitue une approche originale dans ce domaine. Dans le cadre de ces applications, notre contribution essentielle est d'accélérer significativement la résolution de ces problèmes, via l'utilisation de l'algorithme proposé.

Cette thèse, financée par une bourse ministérielle (MENRT), a été préparée au sein de l'Université Paris-Est Créteil (UPEC), dans le *Laboratoire Images, Signaux et Systèmes Intelligents* (LiSSi, E.A. 3956). Elle a été dirigée par le Professeur P. Siarry, Directeur de l'équipe *Traitement de l'Image et du Signal*, et co-encadrée par Messieurs A. Nakib et H. Oulhadj, maîtres de conférences au LiSSi et membres de cette équipe. Cette dernière est notamment spécialisée dans les métaheuristiques et dans leurs applications en génie biologique et médical.

Les principaux apports de ce travail de thèse sont :

- l'élaboration de nouveaux algorithmes performants, MADO et MLSDO, adaptés aux problèmes continus dynamiques ;
- leurs applications en imagerie médicale, plus spécifiquement en segmentation et en recalage de séquences d'images ciné-IRM cérébrales.

Le plan de ce manuscrit est le suivant.

Dans le premier chapitre, nous présentons un état de l'art des méthodes d'optimisation dynamique. L'accent est mis sur l'optimisation dynamique en variables continues.

Dans le deuxième chapitre, nous présentons l'algorithme d'optimisation dynamique MADO (*MultiAgent Dynamic Optimization*), que nous avons élaboré durant cette thèse. Il s'agit d'un algorithme à base d'agents effectuant des recherches locales de manière coordonnée. Nous commençons par le décrire en détail, puis nous en présentons une analyse expérimentale. Une comparaison de ses performances avec celles des autres algorithmes d'optimisation dynamique proposés dans la littérature est également présentée. Ensuite, deux variantes améliorées de MADO, nommées CMADO et PMADO, sont décrites et analysées. Nous terminons en soulignant les défauts de MADO, qui nous ont amenés à mettre au point un autre algorithme, MLSDO.

Dans le troisième chapitre, nous présentons un nouvel algorithme d'optimisation dynamique, nommé MLSDO (*Multiple Local Search algorithm for Dynamic Optimization*). Pour élaborer cet algorithme, nous sommes partis de l'architecture de base de MADO et l'avons fait évoluer, en nous appuyant sur les forces et les faiblesses identifiées dans MADO. Comme précédemment, nous décrivons en détail l'algorithme, en présentons une analyse expérimentale, ainsi qu'une comparaison de ses performances avec celles des autres algorithmes d'optimisation dynamique proposés dans la littérature.

Le quatrième chapitre est consacré à l'application de CMADO à la segmentation de séquences d'images médicales. Les séquences utilisées sont issues d'une nouvelle technique d'ac-

quisition d'images cérébrales ciné-IRM mise en œuvre par le Professeur P. Decq (neurochirurgien) et le Docteur J. Hodel (neuroradiologue) au centre hospitalier Henri Mondor, dans le cadre de l'Antenne Analyse et Restauration du Mouvement (ParisTech-ENSAM CNRS 8005). Le problème de la segmentation est abordé dans le contexte d'un problème de quantification du mouvement d'un ventricule cérébral, dans une séquence d'images ciné-IRM. Tout d'abord, nous décrivons le problème et la méthode proposée pour quantifier ces mouvements. Nous nous focalisons ensuite sur le problème de segmentation propre à cette méthode, basée sur une approche par seuillage d'images. Une analyse expérimentale de la méthode, dans laquelle nous montrons l'intérêt d'utiliser CMADO pour accélérer la segmentation des images, est présentée enfin.

Le cinquième chapitre porte sur l'application de MLSDO au recalage de séquences d'images médicales. Dans un premier temps, nous présentons une procédure de recalage utilisant MLSDO dans le contexte de la méthode de quantification de mouvements décrite au chapitre quatre. Dans un deuxième temps, nous proposons une variante de cette procédure, permettant d'en améliorer la précision. Pour chacune de ces deux procédures, une analyse expérimentale est présentée. Les gains de performance obtenus, par rapport aux algorithmes d'optimisation statique les plus compétitifs, montrent l'intérêt d'utiliser MLSDO sur ce type de problèmes.

Enfin, dans la conclusion générale du manuscrit, nous récapitulons nos contributions et proposons des perspectives, sur la base des travaux effectués.

Chapitre Un

Etat de l'art en optimisation continue dynamique

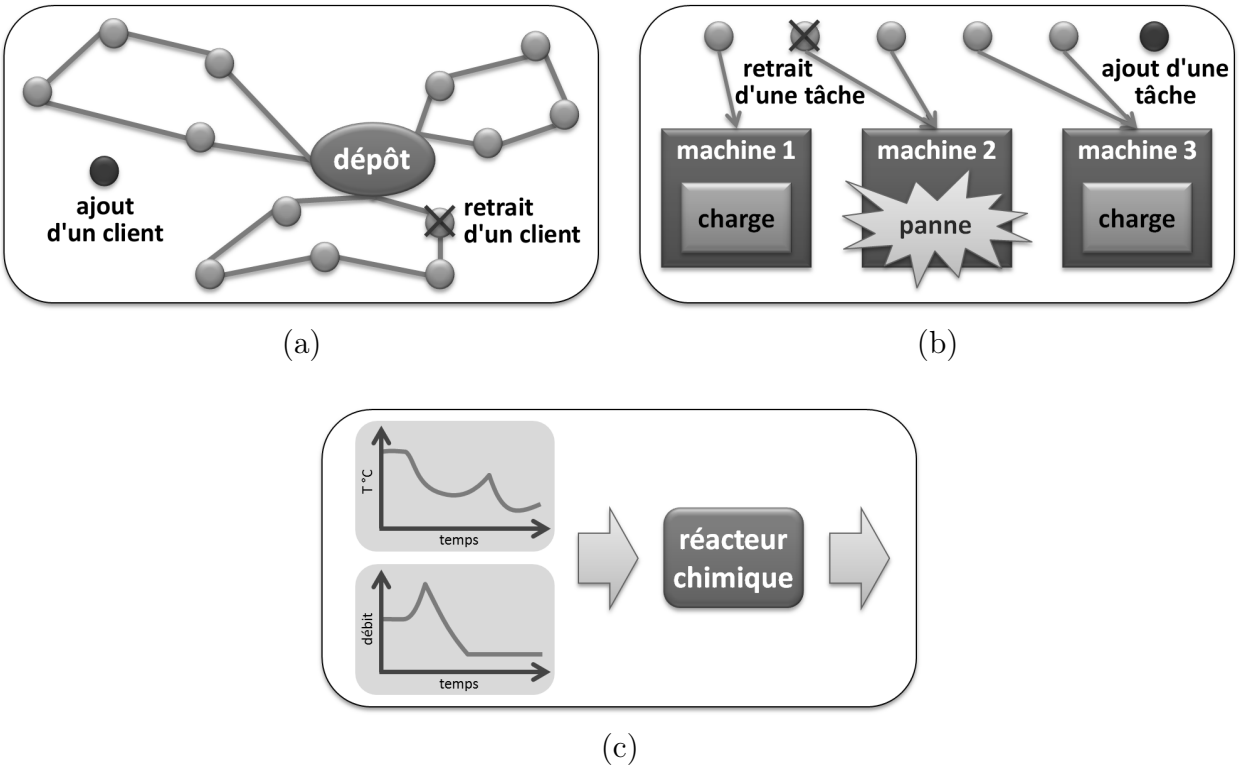
1.1 Introduction

Beaucoup de problèmes réels d'optimisation sont variables dans le temps ou *dynamiques*. Par exemple, un changement peut être dû à des machines qui tombent en panne, à une qualité variable de matières premières, ou à l'apparition de nouvelles tâches devant être planifiées. De ce fait, un intérêt croissant a été porté à ce type de problèmes, et donc aux algorithmes destinés à les résoudre. Un problème d'optimisation dynamique est caractérisé par une fonction objectif qui change en fonction du temps. C'est le cas, par exemple, de problèmes classiques tels que le problème de tournées de véhicules (VRP) ou le problème d'affectation de tâches. On peut également citer le contrôle de réactions chimiques, tributaires, entre autres, des conditions de température et de pression.

Le VRP compte parmi les problèmes les plus étudiés. Il consiste à déterminer les tournées d'une flotte de véhicules afin de livrer un ensemble de clients, tout en minimisant le coût de livraison des biens [Prins, 2009]. Une variante dynamique de ce problème est illustrée à la figure 1.1(a), où les tournées de différents véhicules, depuis un dépôt central, sont représentées par des cycles dont les sommets correspondent à des clients. La nature dynamique de cette variante vient du fait que des clients peuvent être ajoutés ou supprimés inopinément (problème connu dans la littérature sous l'acronyme DVRP).

Une variante dynamique du problème d'affectation de tâches est également illustrée à la figure 1.1(b). Il s'agit de répartir un ensemble de tâches sur différentes machines, de manière à minimiser le temps nécessaire pour toutes les traiter. La nature dynamique de ce problème vient du fait que des tâches peuvent apparaître ou disparaître à tout moment, et que des machines peuvent tomber en panne.

Enfin, le contrôle de réactions chimiques consiste à déterminer un ensemble de paramètres (par exemple, un débit d'alimentation ou l'évolution d'une température, comme illustré à la figure 1.1(c)), permettant de maximiser les performances d'un réacteur chimique [Srinivasan et al., 2003]. Traditionnellement, ces paramètres étaient calculés préalablement à la réaction,



(a) (b)

(c)

FIGURE 1.1 – Exemples de problèmes réels d'optimisation dynamique connus : (a) le problème de tournées de véhicules, (b) le problème d'affectation de tâches, (a) le problème du contrôle de réactions chimiques.

sur la base de quantités constantes de réactifs. Cependant, du fait de la nature changeante des processus chimiques, il est nécessaire d'ajuster ces paramètres au cours du temps, afin d'obtenir leurs valeurs optimales. De ce fait, avec l'augmentation de la puissance de calcul et l'apparition de nouveaux capteurs, un intérêt croissant a été porté à l'optimisation dynamique de réactions chimiques.

Une formulation mathématique du problème d'optimisation dynamique est donnée en (1.1.1), où $f(\vec{x}, t)$ désigne la fonction objectif d'un problème de minimisation, $h_j(\vec{x}, t)$ désigne la $j^{\text{ème}}$ contrainte d'égalité et $g_k(\vec{x}, t)$ la $k^{\text{ème}}$ contrainte d'inégalité. Toutes ces fonctions peuvent varier au cours du temps. Le but n'est pas alors seulement de trouver l'optimum global, mais de le suivre aussi fidèlement que possible dans le temps.

$$\begin{aligned} \min \quad & f(\vec{x}, t) \\ \text{s.c.} \quad & h_j(\vec{x}, t) = 0 \text{ for } j = 1, 2, ..., u \\ & g_k(\vec{x}, t) \leq 0 \text{ for } k = 1, 2, ..., v \end{aligned} \tag{1.1.1}$$

Le moyen le plus simple pour résoudre un problème dynamique est de redémarrer un algorithme d'optimisation statique (c'est-à-dire un algorithme conçu pour des problèmes dont la

fonction objectif n'évolue pas au cours du temps), à chaque fois qu'un changement se produit dans la fonction objectif. Cela nécessite, toutefois, de pouvoir déterminer l'instant du changement et d'avoir suffisamment de temps entre chaque changement pour trouver une solution acceptable.

Une accélération de la convergence peut naturellement être envisagée en utilisant des informations issues des exécutions précédentes de l'algorithme. Par exemple, notons $\vec{o}$ et $\vec{o'}$ les positions de l'optimum global, respectivement avant et après un changement dans la fonction objectif. Si nous supposons que $\vec{o'}$ est « proche » de $\vec{o}$, il peut alors être avantageux de redémarrer la recherche à partir de $\vec{o}$, ou de la restreindre au voisinage de $\vec{o}$. Cependant, l'utilisation d'informations sur les états précédents de la fonction objectif ne peut être bénéfique que si l'intensité des changements n'est pas trop importante. Si les changements sont drastiques, et la fonction objectif complètement transformée après un changement, alors un simple redémarrage de l'algorithme reste la meilleure solution. Néanmoins, pour la plupart des problèmes réels, la fonction objectif ne subit pas de transformations importantes lorsque survient un changement. La question est alors de savoir quelles informations exploiter depuis l'historique de la fonction, et comment les utiliser pour accélérer la convergence. On souligne en outre que redémarrer un algorithme d'optimisation statique dès qu'un changement a lieu dans la fonction objectif n'est pas toujours possible et peut s'avérer inefficace.

Des algorithmes spécialement conçus pour l'optimisation dynamique ont alors été proposés dans la littérature. Dans ce chapitre, nous présentons dans le paragraphe 1.2 les principales techniques utilisées par ces algorithmes pour opérer sur des problèmes dynamiques. Ensuite, nous décrivons dans le paragraphe 1.3 les principales familles d'algorithmes utilisés en optimisation dynamique. Dans le paragraphe 1.4, nous passons en revue les principales mesures de performance et les principaux jeux de tests proposés dans la littérature. Enfin, nous concluons ce chapitre au paragraphe 1.5.

1.2 Les techniques utilisées en optimisation dynamique

Le plus souvent, les algorithmes utilisés sont des métaheuristiques classiques d'optimisation statique ayant été adaptées au cas dynamique, au moyen de différentes techniques. En effet, les algorithmes d'optimisation statique ne peuvent pas en général être utilisés tels quels sur des problèmes dynamiques, principalement pour deux raisons :

1. Après un changement dans la fonction objectif, les informations que l'algorithme a pu accumuler sur celle-ci peuvent être « périmées ». C'est le cas notamment des valeurs de la fonction objectif des différentes solutions archivées en mémoire. Il peut alors être

nécessaire de mettre en place des traitements spécifiques, afin que le processus d'optimisation n'en soit pas perturbé. Il est possible, par exemple, de réévaluer tout ou partie des solutions mémorisées par l'algorithme.

2. Après un changement dans la fonction objectif, il est probable que l'optimum global ait changé de position dans l'espace de recherche. Il est alors nécessaire de converger rapidement vers sa nouvelle position, afin de pouvoir le suivre au fil des changements de la fonction. Si la nouvelle position de l'optimum est très éloignée de la précédente, l'algorithme doit assurer une diversification suffisante pour pouvoir la trouver. Or, lorsqu'un algorithme d'optimisation statique converge vers un optimum, il peut lui être difficile de s'en écarter rapidement, afin d'explorer des zones plus prometteuses de l'espace de recherche.

Les principales techniques proposées pour permettre le suivi et la détection d'optima peuvent être regroupées dans cinq classes [Jin & Branke, 2005] :

1. **Générer de la diversité après un changement :** lorsqu'un changement dans la fonction objectif est détecté, des traitements sont effectués pour augmenter la diversité des solutions et faciliter ainsi la recherche du nouvel optimum.
2. **Maintenir la diversité tout au long de la recherche :** la diversité des solutions est préservée au cours du temps, en partant du principe qu'une population largement répartie dans l'espace de recherche peut s'adapter plus efficacement aux changements.
3. **Utiliser une mémoire :** l'algorithme dispose d'une mémoire pour sauvegarder des informations sur le passé de la fonction objectif. En pratique, les bonnes solutions trouvées sont stockées, en vue d'être réutilisées lorsqu'un changement est détecté. Des techniques plus sophistiquées ont également été proposées dans la littérature [Yang & Yao, 2008].
4. **Utiliser plusieurs populations :** diviser la population en plusieurs sous-populations, distribuées sur différents optima locaux, permet de suivre plusieurs optima à la fois et d'augmenter la probabilité d'en trouver de nouveaux. Historiquement, des algorithmes de référence en optimisation dynamique sont basés sur cette approche [Branke et al., 2000; Parrott & Li, 2004].
5. **Prédire les futurs changements :** récemment, une attention particulière s'est portée sur des techniques visant à prédire les changements. Cette approche repose sur le fait que, pour des problèmes réels, les changements dans la fonction objectif sont susceptibles de suivre un certain schéma, qui peut être appris.

La plupart des algorithmes d'optimisation dynamique doivent ainsi détecter les changements qui surviennent dans la fonction objectif. Le moyen le plus simple et le plus répandu pour y parvenir consiste à réévaluer une solution : si la valeur de cette solution a changé, on considère alors qu'un changement a eu lieu dans la fonction objectif. Des techniques de détection de changement plus élaborées ont également été proposées [Richter, 2009]. Néanmoins, la détection de changement par réévaluation d'une ou plusieurs solutions reste souvent la méthode la plus efficace.

1.3 Les différentes approches en optimisation dynamique

La plupart des algorithmes d'optimisation dynamique proposés dans la littérature sont des métaheuristiques, généralement bio-inspirées, utilisant une ou plusieurs populations de solutions. La majorité d'entre eux appartiennent à la classe des *algorithmes évolutionnaires* (s'inspirant de la théorie de l'évolution de Darwin) ou à celle de l'*optimisation par essaim particulaire*. Néanmoins, d'autres algorithmes, tels que l'*optimisation par colonies de fourmis*, les *systèmes immunitaires artificiels* et des approches hybrides, ont également été proposés. Pour une description détaillée de ces classes de métaheuristiques, on peut se référer à [Dréo et al., 2003].

1.3.1 Les algorithmes évolutionnaires (EAs)

1.3.1.1 Principe

Les EAs, élaborés au cours des années 1950 [Fraser, 1957], forment une famille d'algorithmes d'optimisation inspirés de l'évolution biologique des espèces. Ils s'inspirent de la théorie Darwinienne de la sélection naturelle des espèces : les individus qui ont hérité de caractères bien adaptés à leur milieu ont tendance à vivre assez longtemps pour se reproduire, alors que les plus faibles ont tendance à disparaître. Au cours des années 1960 et 1970, avec l'avènement des calculateurs, de nombreuses tentatives de modélisation de l'évolution ont été entreprises. Plusieurs approches ont ainsi émergé :

- Les *algorithmes génétiques* [Holland, 1975], initialement conçus pour résoudre des problèmes d'optimisation à variables discrètes, en modélisant l'évolution génétique.
- La *programmation génétique* [Koza, 1989, 1990], basée sur les algorithmes génétiques, mais où les individus (ou chromosomes) sont des programmes informatiques, représentés en utilisant une structure d'arbre.
- La *programmation évolutionnaire* [Fogel, 1962; Fogel et al., 1966], historiquement conçue pour des problèmes d'apprentissage à partir d'automates à états finis. Elle travaille directement sur le *phénotype* (notamment, le comportement des automates finis), en utilisant

une succession de sélections et de mutations.

- Les *statégies d'évolution* [Rechenberg, 1965; Beyer, 2001], initialement conçues pour résoudre des problèmes à variables continues. Elles sont axées sur la modélisation des paramètres stratégiques qui contrôlent la variation dans l'évolution, autrement dit « l'évolution de l'évolution ».
- L'*évolution différentielle* [Storn & Price, 1997; Price et al., 2005], initialement conçue pour résoudre des problèmes à variables continues. Sa stratégie consiste à biaiser un opérateur de mutation, appliqué à un individu, en fonction des différences calculées entre d'autres individus sélectionnés aléatoirement.

Les approches évolutionnaires s'appuient sur un modèle commun présenté par l'algorithme 1.1. Les individus soumis à l'évolution sont des solutions possibles du problème posé. L'ensemble de ces individus constitue une *population*. Cette population évolue durant une succession d'itérations, appelées *générations*. Au cours de chaque génération, une série d'opérateurs est appliquée aux individus, pour créer la population de la génération suivante. Chaque opérateur utilise un ou plusieurs individus, appelés *parents*, pour engendrer de nouveaux individus, appelés *enfants*. A la fin de chaque génération, une sélection d'enfants créés durant la génération remplace un sous-ensemble d'individus de la population.

modèleCommun

```
1 Initialisation de la population de μ individus
2 Evaluation des μ individus
3 tant que le critère d'arrêt n'est pas satisfait faire
4 |   Sélection de ρ individus en vue de la phase de reproduction
5 |   Croisement des ρ individus sélectionnés
6 |   Mutation des λ enfants obtenus
7 |   Evaluation des λ enfants obtenus
8 |   Sélection pour le remplacement
9 fin
```

ALGORITHME 1.1 – Algorithme évolutionnaire générique.

Un algorithme évolutionnaire dispose de trois opérateurs principaux :

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

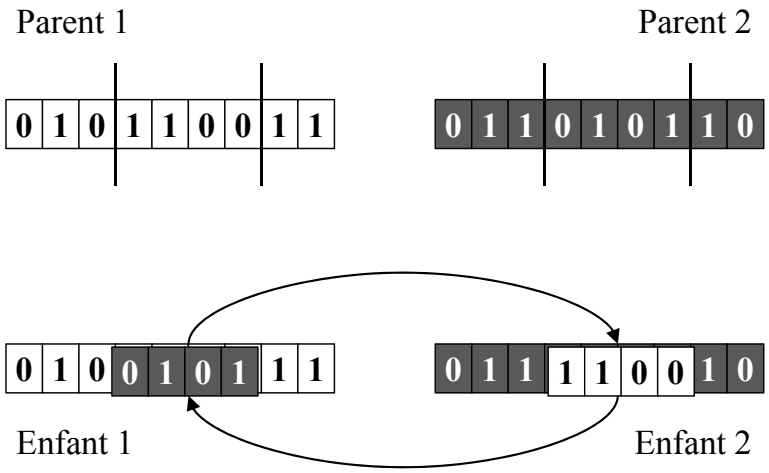


FIGURE 1.2 – Exemple d'opérateur de croisement en représentation binaire.

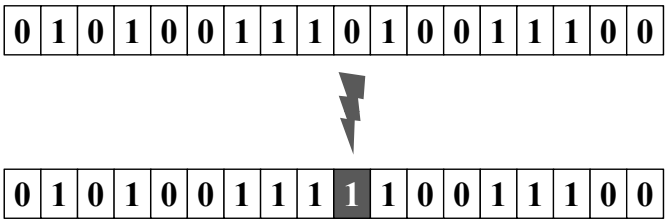


FIGURE 1.3 – Exemple d'opérateur de mutation en représentation binaire.

croisement simple, pour des individus codés en représentation binaire, est présenté à la figure 1.2 ;

3. un opérateur de mutation, qui consiste à tirer aléatoirement une composante de l'individu parent et à la remplacer par une valeur aléatoire. L'apport d'un caractère aléatoire à la création de la descendance permet ainsi de maintenir une certaine diversité dans la population. La figure 1.3 montre un exemple de mutation, pour un individu codé en représentation binaire.

Une liste des méthodes existantes pour définir ces opérateurs est disponible dans [Eiben & Smith, 2008].

1.3.1.2 Les EAs en optimisation dynamique

De nombreux algorithmes d'optimisation dynamique sont basés sur les principes des EAs. En effet, les algorithmes de cette classe peuvent s'avérer adaptés à des environnements changeants, du fait qu'ils s'inspirent de la théorie de l'évolution des espèces. Les EAs à plusieurs populations peuvent notamment obtenir de bons résultats pour des problèmes d'optimisation dynamique multimodaux. En effet, des résultats intéressants ont été obtenus par l'algorithme *self-organizing scouts* proposé dans [Branke et al., 2000]. Cet algorithme utilise une population

mère pour explorer l'espace de recherche, pendant que des populations filles suivent les optima locaux, au fil des changements dans la fonction objectif. Dans [Mendes & Mohais, 2005], un algorithme à *évolution différentielle* (DE) utilisant plusieurs populations est présenté. Cet algorithme est basé sur des techniques visant à maintenir une bonne diversité entre les individus et entre les populations. Une variante basée sur l'évolution de plusieurs populations est présentée dans [Brest et al., 2009]. La stratégie utilisée par cet algorithme est la suivante : chaque individu prend de l'âge au fil des générations, et les vieux individus qui stagnent sont réinitialisés. Si un individu qui stagne est le meilleur de sa population, alors cette population est réinitialisée. De plus, si deux populations se chevauchent, alors la plus « mauvaise » d'entre elles (en comparant le meilleur individu d'une population avec celui de l'autre) est réinitialisée. Par ailleurs, les meilleurs individus sont archivés au cours de l'exécution de l'algorithme, et l'archive qui en résulte est utilisée lors de la réinitialisation d'individus. [Yu & Suganthan, 2009] présentent un algorithme à *programmation évolutionnaire*, qui s'adapte aux changements dans la fonction objectif, en générant de la diversité après un changement, en maintenant une bonne diversité tout au long de l'exécution, et en sauvegardant des individus dans deux archives (l'une représentant la mémoire à court terme, et l'autre, la mémoire à long terme).

Dans [Rossi et al., 2008], l'algorithme proposé utilise une technique de prédiction de mouvement basée sur un filtrage de Kalman, afin d'accélérer le suivi des optima locaux. Ce type de prédiction peut s'avérer efficace pour de nombreux problèmes, mais il augmente la complexité et le nombre de paramètres de l'algorithme.

1.3.1.3 Points forts et lacunes

Selon la théorie de l'évolution Darwinienne, la sélection naturelle permet aux espèces de s'adapter aux changements dans leur environnement. De ce fait, les EAs se présentent comme de bons candidats pour résoudre des problèmes d'optimisation dynamique. En effet, il est également nécessaire, en optimisation dynamique, de s'adapter aux changements qui ont lieu dans la fonction objectif, afin de retrouver rapidement l'optimum global.

Cependant, les EAs sont des métaheuristiques dont l'objectif est de trouver l'optimum global, en éludant volontairement les optima locaux. Or, il est important de ne pas négliger les optima locaux d'un problème d'optimisation dynamique, car l'un d'eux peut évoluer et devenir le nouvel optimum global après un changement. De plus, la capacité des EAs à éviter les optima locaux peut occasionner, en contrepartie, une convergence plus lente. Or, il est plus important pour un algorithme d'optimisation dynamique de converger rapidement vers une bonne solution, que de converger plus lentement vers l'optimum global.

Pour pouvoir trouver et suivre les optima locaux au cours du temps, des EAs utilisant plu-

sieurs populations ont été proposés. Néanmoins, la convergence de ces algorithmes reste à être améliorée, d'autant que l'utilisation d'un grand nombre de populations à la fois peut ralentir davantage leur convergence. De plus, l'adaptation des EAs à l'optimisation dynamique complexifie significativement les algorithmes. L'usage de plusieurs populations accroît notamment à la fois la complexité des algorithmes et le nombre de paramètres que l'utilisateur doit régler.

1.3.2 L'optimisation par essaim particulaire (OEP)

1.3.2.1 Principe

L'OEP a été proposée par Kennedy et Eberhart en 1995 [Kennedy & Eberhart, 1995]. Elle s'inspire du comportement social des animaux évoluant en essaim, tels que les nuées d'oiseaux et les bancs de poissons. Un individu de l'essaim ne dispose que d'une connaissance locale de sa situation dans l'essaim. Il utilise cette information locale, ainsi que sa propre mémoire, pour décider de son déplacement. Des règles simples, telles que « aller dans une même direction » ou « rester proche de ses voisins », suffisent à maintenir la cohésion de l'essaim, et permettent la mise en œuvre de comportements collectifs complexes et adaptatifs.

Le principe de l'OEP s'est éloigné du comportement (trop complexe) des animaux, pour ne conserver qu'une modélisation basées sur des agents simples, appelés *particules*. Un essaim de particules, qui sont des solutions potentielles au problème d'optimisation, « survole » l'espace de recherche, à la recherche de l'optimum global. Le déplacement d'une particule est influencé par trois composantes :

- une composante *d'inertie* : la particule tend à suivre sa direction courante de déplacement ;
- une composante *cognitive* : la particule tend à se fier à sa propre expérience et, ainsi, à se diriger vers le meilleur site par lequel elle est déjà passée ;
- une composante *sociale* : la particule tend à se fier à l'expérience de ses congénères et, ainsi, à se diriger vers le meilleur site déjà atteint collectivement par l'essaim.

La figure 1.4 illustre la stratégie de déplacement d'une particule :

Dans un espace de recherche de dimension d, la particule i de l'essaim est modélisée par son vecteur position $\vec{x_i} = (x_{i1}\ x_{i2}\ ...\ x_{id})^{\mathrm{T}}$ et par son vecteur vitesse $\vec{v_i} = (v_{i1}\ v_{i2}\ ...\ v_{id})^{\mathrm{T}}$, en désignant par A^{T} la transposée d'une matrice A. La qualité de sa position est déterminée par la valeur de la fonction objectif en ce point. Cette particule garde en mémoire la meilleure position par laquelle elle est déjà passée, que l'on note $\vec{p_i} = (p_{i1}\ p_{i2}\ ...\ p_{id})^{\mathrm{T}}$. La meilleure position atteinte par ses particules voisines est notée $\vec{g_i} = (g_{i1}\ g_{i2}\ ...\ g_{id})^{\mathrm{T}}$. Au temps t, le vecteur vitesse est calculé à partir de (1.3.1).

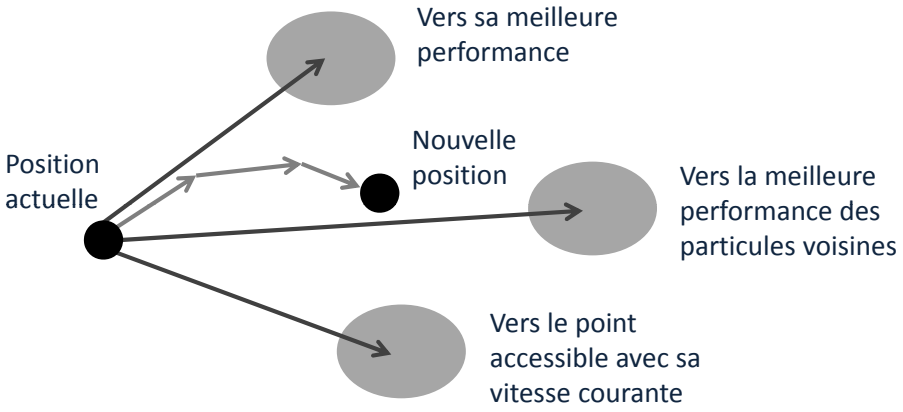


FIGURE 1.4 – Déplacement d'une particule.

$$v_{ij}(t+1) = w\ v_{ij}(t) + c_1\ r_1\,(p_{ij}(t) - x_{ij}(t)) + c_2\ r_2\,(g_{ij}(t) - x_{ij}(t))\,,\ j \in \{1, 2, ..., d\} \quad (1.3.1)$$

où w est en général une constante, appelée *coefficient d'inertie* ; c_1 et c_2 sont deux constantes, appelées *coefficients d'accélération* ; r_1 et r_2 sont deux nombres aléatoires tirés uniformément dans $[0, 1]$, à chaque itération et pour chaque dimension.

Il est à noter que le terme « vitesse » est ici abusif, car les vecteurs $\vec{v}_i$ ne sont pas homogènes à une vitesse. Cependant, pour respecter l'analogie avec le monde animal, les auteurs ont préféré utiliser ce terme.

Dans l'équation précédente, $v_{ij}(t)$ correspond à la composante *d'inertie* du déplacement. Le paramètre w contrôle l'influence de la direction de déplacement sur le déplacement futur. L'expression $c_1\ r_1\,(p_{ij}(t) - x_{ij}(t))$ correspond à la composante *cognitive* du déplacement. Le paramètre c_1 contrôle le comportement cognitif de la particule. L'expression $c_2\ r_2\,(g_{ij}(t) - x_{ij}(t))$ correspond à la composante *sociale* du déplacement. Le paramètre c_2 contrôle l'aptitude sociale de la particule.

La combinaison des paramètres w, c_1 et c_2 permet de régler l'équilibre entre les phases de diversification et d'intensification du processus de recherche. [Clerc & Kennedy, 2002] ont démontré qu'une bonne convergence peut être obtenue en rendant dépendants ces paramètres. L'utilisation d'un *facteur de constriction* χ permet de prévenir la divergence de l'essaim. L'équation (1.3.1) devient alors :

$$v_{ij}(t+1) = \chi\,(v_{ij}(t) + \phi_1\ r_1\,(p_{ij}(t) - x_{ij}(t)) + \phi_2\ r_2\,(g_{ij}(t) - x_{ij}(t)))\,,\ j \in \{1, 2, ..., d\} \quad (1.3.2)$$

avec :

$$\chi = \frac{2}{\phi - 2 + \sqrt{\phi^2 - 4\phi}},\ \phi = \phi_1 + \phi_2,\ \phi > 4 \quad (1.3.3)$$

La position au temps t de la particule i est alors définie par (1.3.4).

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1),\ j \in \{1, 2, ..., d\} \tag{1.3.4}$$

Les différentes étapes de l'OEP sont présentées dans l'algorithme 1.2.

OEP

Initialiser aléatoirement les positions et vitesses de chaque particule
Evaluer les positions des particules
Pour chaque particule i, $\vec{p}_i = \vec{x}_i$
Calculer les $\vec{g}_i$
tant que le critère d'arrêt n'est pas satisfait **faire**
    Déplacer les particules selon (1.3.2) et (1.3.4)
    Evaluer les positions des particules
    Mettre à jour $\vec{p}_i$ et $\vec{g}_i$
fin

ALGORITHME 1.2 – Algorithme d'optimisation par essaim particulaire.

1.3.2.2 L'OEP en optimisation dynamique

Cette classe d'algorithmes est également très utilisée en optimisation dynamique. Parmi les algorithmes d'optimisation dynamique basés sur l'OEP, un algorithme utilisant plusieurs essaims de particules, ainsi qu'un processus de *spéciation* (les particules spatialement proches forment une espèce particulière), est proposé dans [Li, 2004; Parrott & Li, 2004], où chaque particule a pour voisines les particules de son espèce. Plusieurs variantes améliorées de cet algorithme sont proposées dans [Parrott & Li, 2006; Li et al., 2006; Bird & Li, 2007]. Les algorithmes à plusieurs essaims (ou multi-essaims) sont en effet très utilisés en optimisation dynamique. Un autre algorithme basé sur l'OEP est proposé dans [Du & Li, 2008]. Cet algorithme utilise deux essaims de particules : le premier pour la diversification, et le deuxième pour l'intensification. Dans [Li & Yang, 2009], l'approche multi-essaims proposée utilise des techniques de regroupement (ou *clustering*) pour créer des sous-essaims, et les sous-essaims se chevauchant sont fusionnés. Les meilleures solutions trouvées avant qu'un changement ne survienne dans la fonction objectif sont archivées, afin d'être ajoutées en tant que nouvelles particules, lorsqu'un changement est détecté.

L'utilisation dans l'OEP de concepts issus du domaine de la physique a également été très étudiée. Dans [Blackwell & Branke, 2004], deux algorithmes multi-essaims basés sur un modèle atomique sont présentés. Le premier algorithme utilise des essaims composés de sous-essaims

de particules se repoussant entre elles, et tournant autour d'un autre sous-essaim de particules *neutres* (particules conventionnelles de l'OEP). Le deuxième algorithme est basé sur un modèle quantique de l'atome, où des particules chargées (électrons) ne suivent pas une trajectoire classique de l'OEP, mais sont réinitialisées aléatoirement dans une boule (sphère pleine) centrée sur la meilleure position trouvée par l'essaim. Ces deux algorithmes placent leurs essaims sur chaque optimum local trouvé, afin que chaque essaim suive un optimum local différent. La même approche est utilisée dans [Blackwell & Branke, 2006], [Li et al., 2006] et [Novoa et al., 2009], hybridée avec d'autres techniques visant à augmenter la diversité et à suivre les optima locaux (réinitialisation du plus mauvais essaim ; utilisation d'un processus de spéciation avec réinitialisation des particles appartenant à la plus mauvaise espèce ; contrôle de la trajectoire des particules n'améliorant pas leur *fitness* pendant un certain nombre de déplacements). Un autre concept est utilisé dans [Liu et al., 2008, 2010], où des « particules composites » sont créées dans l'essaim. Ces particules composites, formées de trois particules élémentaires, sont inspirées des particules composites en physique.

1.3.2.3 Points forts et lacunes

Comme les EAs, l'OEP est initialement conçue pour résoudre des problèmes d'optimisation statique, et présente des défauts intrinsèques en optimisation dynamique. On retrouve des inconvénients similaires à ceux des EAs, qu'il est nécessaire de pallier au moyen de différentes techniques, ce qui rend les algorithmes plus complexes.

On remarque que les algorithmes d'optimisation dynamique basés sur l'OEP sont essentiellement multi-essaims. L'utilisation de plusieurs essaims permet en effet, tout comme l'utilisation de plusieurs populations pour les EAs, de suivre différents optima locaux, tout en augmentant la probabilité d'en trouver de nouveaux. Néanmoins, les algorithmes multi-essaims sont plus complexes et comportent plus de paramètres à régler.

1.3.3 L'optimisation par colonies de fourmis (ACO)

1.3.3.1 Principe

Comme l'OEP, ACO (de l'anglais *Ant Colony Optimization*) est une approche utilisant l'intelligence en essaim. Elle résulte de l'observation des insectes sociaux, en particulier des fourmis, qui résolvent naturellement des problèmes complexes. Une telle aptitude s'avère possible en raison de la capacité des fourmis à communiquer entre elles indirectement, par le dépôt sur le sol de substances chimiques, appelées *phéromones*. Ce type de communication indirecte est appelé *stigmergie*.

La figure 1.5 illustre ce phénomène : un obstacle est placé sur le trajet des fourmis qui,

après une étape d'exploration, finiront par emprunter le plus court chemin entre le nid et la source de nourriture [Goss et al., 1989]. Les fourmis qui sont retournées le plus rapidement au nid en passant par la source de nourriture sont celles qui ont emprunté le chemin le plus court. Il en découle que la quantité de phéromones déposées par unité de temps sur ce trajet est plus importante que sur les autres. Par ailleurs, une fourmi est d'autant plus attirée à un certain endroit que le taux de phéromones y est important. De ce fait, le plus court chemin a une probabilité plus importante d'être emprunté par les fourmis que les autres chemins et sera donc, à terme, emprunté par toutes les fourmis.

Le premier algorithme d'optimisation s'inspirant de cette analogie a été proposé par Colorni, Dorigo et Maniezzo [Colorni et al., 1991; Dorigo et al., 1996], afin de résoudre le problème du voyageur de commerce. L'algorithme 1.3 résume l'approche par colonies de fourmis proposée par les auteurs. Si l'on considère un problème de voyageur de commerce à N villes, chaque fourmi k parcourt le graphe et construit un trajet de longueur $n = |N|$. Pour chaque fourmi, le trajet d'une ville i à une ville j dépend de :

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

La règle de déplacement est la suivante :

$$p_{ij}^k(t) = \begin{cases} \frac{(\tau_{ij}(t))^\alpha (\eta_{ij})^\beta}{\sum_{l \in J_i^k} (\tau_{il}(t))^\alpha (\eta_{il})^\beta} & \text{si } j \in J_i^k \\ \\ 0 & \text{sinon} \end{cases} \tag{1.3.5}$$

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

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

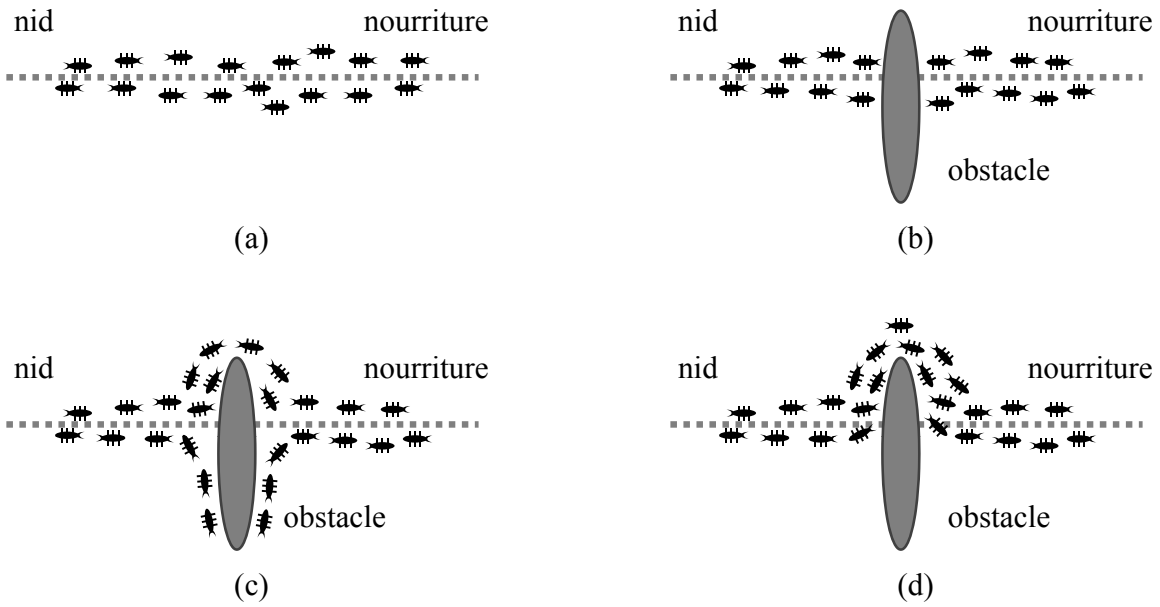


FIGURE 1.5 – Illustration de la capacité des fourmis à chercher de la nourriture en minimisant leur parcours. (a) Recherche sans obstacle, (b) Apparition d'un obstacle, (c) Recherche du chemin optimal, (d) Chemin optimal trouvé.

antSystem

$t \leftarrow 1$
tant que le critère d'arrêt n'est pas satisfait **faire**
  pour $k = 1$ à m **faire**
    Choisir une ville au hasard
    pour chaque ville non visitée i **faire**
      Choisir une ville j dans la liste J_i^k des villes restantes selon (1.3.5)
    fin
    Déposer une piste $\Delta\tau_{ij}^k(t)$ sur le trajet $T^k(t)$ conformément à (1.3.6)
  fin
  Evaporer les pistes selon (1.3.7)
  $t \leftarrow t + 1$
fin

ALGORITHME 1.3 – Algorithme de colonies de fourmis pour le problème du voyageur de commerce.

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \frac{Q}{L^k(t)} & \text{si } (i,j) \in T^k(t) \\ \\ 0 & \text{sinon} \end{cases} \tag{1.3.6}$$

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

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

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t) \tag{1.3.7}$$

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

Depuis, la démarche initiée par cette analogie a été étendue à la résolution d'autres problèmes d'optimisation, discrets et continus [Dorigo & Blum, 2005; Dréo et al., 2003]. ACO présente plusieurs caractéristiques intéressantes, telles que le parallélisme intrinsèque élevé, la robustesse (une colonie peut maintenir une recherche efficace, même si certains de ses individus sont défaillants) ou encore la décentralisation (les fourmis n'obéissent pas à une autorité centralisée).

1.3.3.2 ACO en optimisation dynamique

ACO a également été utilisée en optimisation dynamique. Dréo et al. ont présenté dans [Dréo & Siarry, 2006] une hybridation de ACO avec une recherche locale (*simplexe de Nelder-Mead* [Nelder & Mead, 1965]). Néanmoins, cet algorithme a des difficultés à résoudre des problèmes comportant un grand nombre d'optima locaux. Dans [Tfaili & Siarry, 2008], une charge électrostatique est attribuée aux fourmis, afin de maintenir une bonne diversité. De plus, cela permet d'éviter aux fourmis de converger vers un même optimum local. Ainsi, cet algorithme est plus performant que celui de [Dréo & Siarry, 2006], pour des problèmes présentant beaucoup d'optima locaux, mais il est moins performant pour des problèmes en présentant peu. Dans [Korosec & Silc, 2009], un algorithme simple basé sur ACO pour l'optimisation combinatoire est utilisé pour optimiser un graphe représentant plusieurs chemins possibles qu'une fourmi peut emprunter depuis la solution courante de l'algorithme. La quantité de phéromones est augmentée sur les sommets du graphe correspondant au meilleur chemin trouvé, parmi les chemins empruntés par toutes les fourmis. Lorsque les fourmis stagnent, la solution courante et les quantités de phéromones sont réinitialisées.

1.3.3.3 Points forts et lacunes

Comme nous l'avons vu au paragraphe 1.3.3.1, dans la nature si un obstacle apparaît sur le chemin des fourmis, celles-ci parviendront après une étape de recherche à retrouver le plus court chemin de leur nid à leur source de nourriture. On peut faire ici un parallèle entre l'apparition de l'obstacle et la survenue d'un changement dans un problème d'optimisation dynamique. Ainsi, ACO peut être vue comme une classe d'algorithmes adaptée à l'optimisation dynamique.

Néanmoins, les algorithmes de cette classe présentent les mêmes défauts, en optimisation dynamique, que les EAs ou l'OEP. De plus, les algorithmes basés sur ACO convergent souvent lentement. Il s'agit d'un défaut critique en optimisation dynamique. En outre, ils peuvent également nécessiter une hybridation avec une recherche locale afin d'améliorer la précision des solutions trouvées.

1.3.4 Les systèmes immunitaires artificiels (AIS)

1.3.4.1 Principe

AIS cherche à modéliser et à appliquer des principes d'immunologie pour résoudre des problèmes d'optimisation. Le système immunitaire d'un organisme est un système biologique robuste et adaptatif, qui défend le corps 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* [Dréo et al., 2003]. En effet, les cellules vivantes disposent sur leurs membranes de molécules spécifiques dites *antigènes*. Chaque organisme possède ainsi une identité unique, déterminée par l'ensemble des antigènes présents sur ses cellules. Certaines cellules du système immunitaire, appelées *lymphocytes*, possèdent des *récepteurs* capables de se lier spécifiquement à un antigène donné, permettant ainsi de reconnaître une cellule étrangère à l'organisme. Un lymphocyte ayant reconnu une cellule du non-soi va être incité 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, son clonage permettra à l'organisme de réagir plus vite à une nouvelle exposition à l'antigène (c'est le principe de la vaccination). A noter qu'un mécanisme d'*hyper-mutation* des cellules clonées assure la diversité des récepteurs dans l'ensemble de la population des lymphocytes. Dans le second cas, le combat contre les agressions est possible grâce à la production d'*anticorps*. Ces étapes sont résumées dans la figure 1.6.

AIS s'inspire essentiellement des sélections opérées sur les lymphocytes, et des processus permettant la multiplication et la mémoire du système. En effet, ces caractéristiques sont capitales

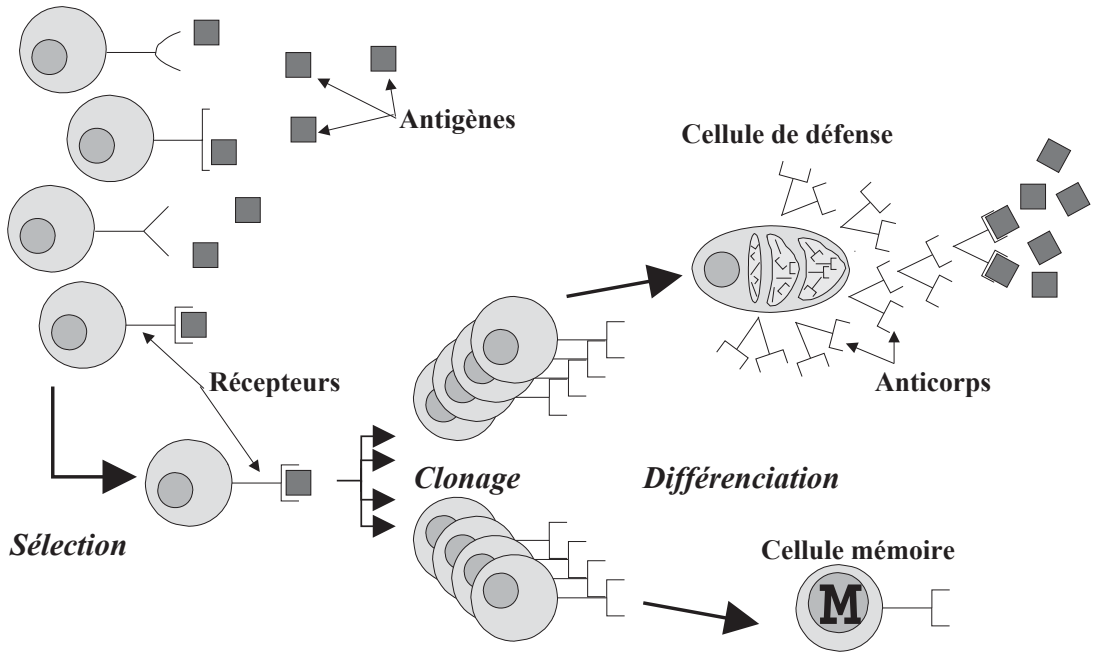


FIGURE 1.6 – La sélection par clonage : des lymphocytes, présentant des récepteurs spécifiques d'un antigène, se différencient en cellule mémoire ou en cellule participant à la défense active de l'organisme par le biais d'anticorps [Dréo et al., 2003].

pour maintenir les propriétés auto-organisées du système.

Cette approche est assez similaire à celle des algorithmes évolutionnaires [Gasper & Collard, 1999]. Elle peut également être comparée à celle des réseaux de neurones [Dasgupta, 1997]. Dans le cadre de l'optimisation, on peut considérer les AIS comme une forme d'algorithme évolutionnaire présentant des opérateurs particuliers. Par exemple, l'opération de sélection est basée 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. Un exemple d'AIS basique est présenté dans l'algorithme 1.4, où t désigne l'itération courante. Plus de détails sur AIS sont donnés dans [de Castro & Timmis, 2002].

1.3.4.2 AIS en optimisation dynamique

L'utilisation de cette classe d'algorithmes en optimisation dynamique est quelque peu marginale. Une étude de plusieurs algorithmes de cette classe est réalisée dans [Trojanowski & Wierzchon, 2009], sans toutefois proposer de majeures modifications de ces derniers en vue de les adapter aux problèmes dynamiques. Dans [de França & Zuben, 2009], une approche par *réseau immunitaire artificiel* est utilisée, où les solutions proches les unes des autres (situées à une distance Euclidienne inférieure à un certain seuil) sont supprimées.

AIS

$t \leftarrow 1$
Initialiser aléatoirement une population $P(t)$ d'individus, composée de m cellules mémoires $P_m(t)$ et de r autres cellules $P_r(t)$: $P(t) = P_m(t) \cup P_r(t)$
Evaluer l'affinité de chaque individu de $P(t)$
tant que le critère d'arrêt n'est pas satisfait **faire**
    Sélectionner un sous-ensemble $S(t)$ d'individus de plus haute affinité dans $P(t)$
    Cloner chaque individu de $S(t)$ proportionnellement à son affinité, pour former une population $C(t)$
    Muter chaque individu de $C(t)$ avec un taux inversement proportionnel à son affinité, pour former une population $C^*(t)$
    Evaluer l'affinité de chaque individu de $C^*(t)$
    Sélectionner les m individus de plus haute affinité dans $C^*(t) \cup P_m(t)$ pour former $P_m(t+1)$
    Remplacer un sous-ensemble d'individus de plus faible affinité dans $P_r(t)$ par des nouveaux individus, tirés aléatoirement, pour former $P_r(t+1)$
    Evaluer l'affinité de ces nouveaux individus
    $t \leftarrow t+1$
fin

ALGORITHME 1.4 – Exemple d'algorithme AIS basique.

1.3.4.3 Points forts et lacunes

Le rôle du système immunitaire est de défendre notre organisme contre les agressions d'agents pathogènes extérieurs. L'apparition dans l'organisme d'un agent pathogène peut être comparée à l'apparition d'un changement dans la fonction objectif d'un problème d'optimisation dynamique. De ce point de vue, il peut être intéressant d'étudier les algorithmes AIS en optimisation dynamique.

Néanmoins, il s'agit d'une classe d'algorithmes assez peu employée en optimisation dynamique. En termes de performance, nous verrons au paragraphe 1.4.2.3 que l'algorithme de [de França & Zuben, 2009] obtient actuellement des résultats inférieurs à ceux des algorithmes des autres classes.

1.3.5 Les approches hybrides

Afin d'améliorer les performances des algorithmes d'optimisation dynamique, certains auteurs ont proposé des algorithmes hybrides. Nous pouvons notamment citer [Lung & Dumitrescu, 2007] et [Lung & Dumitrescu, 2008], qui proposent des algorithmes hybrides OEP/EAs.

Néanmoins, les performances de ces algorithmes ne diffèrent pas significativement de celles des algorithmes basés sur l'OEP ou sur les principes des EAs, décrits ci-dessus. Certains auteurs ont également étudié l'utilisation d'une recherche locale, en tant que composant principal d'un algorithme d'optimisation dynamique. Nous pouvons citer [Moser & Hendtlass, 2007], où les auteurs proposent d'utiliser l'*extremal optimization* (EO) pour déterminer la solution initiale d'une procédure de recherche locale. EO n'utilise pas une population de solutions, mais améliore une solution unique par mutation. Cet algorithme commence par générer un ensemble de solutions candidates échantillonnées régulièrement (à égales distances) le long de chaque dimension de l'espace de recherche. Ensuite, la meilleure solution candidate sert de solution initiale à une recherche locale, afin d'intensifier cette solution. Enfin, la solution est archivée en mémoire et l'algorithme est appliqué de nouveau à partir d'une autre solution générée aléatoirement. Une variante améliorée et généralisée de cet algorithme est proposée dans [Moser & Chiong, 2010], appelée *Hybridised EO*.

1.3.5.1 Points forts et lacunes

L'objectif de l'hybridation de plusieurs algorithmes de classes différentes est de combiner les avantages de chacun au sein d'un même algorithme. Ce dernier risque cependant d'hériter également de leurs faiblesses. De plus, un algorithme résultant de l'hybridation de plusieurs algorithmes peut avoir une complexité importante.

1.4 Validation des algorithmes

De nombreux problèmes réels admettent des fonctions objectifs gourmandes en temps de calcul. De ce fait, l'effort de calcul fourni par un algorithme sur un jeu de tests est souvent exprimé en termes de nombre d'évaluations. C'est le cas pour la plupart des jeux de tests utilisés en optimisation dynamique dans la littérature. Parmi les plus connus, nous pouvons citer le *Moving Peaks Benchmark* (MPB, [Branke, 1999]), le *DF1 Generator* [Morrison & de Jong, 1999], les générateurs de problèmes par combinaison dynamique de fonctions objectifs, proposés dans [Jin & Sendhoff, 2004], le jeu de tests basé sur l'opérateur *ou exclusif* (XOR) proposé dans [Yang, 2003; Yang & Yao, 2005, 2008], ainsi que le *Generalized Dynamic Benchmark Generator* (GDBG, [Li & Yang, 2008; Li et al., 2008]), proposé plus récemment. Nous allons maintenant décrire les principales mesures de performance utilisées, ainsi que les deux jeux de tests les plus souvent utilisés en optimisation continue dynamique : MPB et GDBG.

1.4.1 Mesures de performance

En optimisation dynamique, le but est de trouver rapidement l'optimum global, et de le suivre au cours du temps. En général, il est donc plus important de converger vers l'optimum global rapidement qu'avec précision. Ainsi, les mesures de performance utilisées doivent tenir compte de ce fait. Malgré l'intérêt croissant pour les algorithmes d'optimisation dynamique, il n'y a pas de vision uniforme de la mesure des performances de ces algorithmes. Selon [Morrison, 2003], les mesures de performance proposées dans la littérature, dans le cadre des algorithmes évolutionnaires, peuvent être classées selon qu'elles utilisent :

- la différence entre la valeur de l'optimum global et celle de la meilleure solution trouvée, juste avant qu'un changement ne survienne dans la fonction objectif ;
- la moyenne des valeurs des meilleures solutions trouvées entre deux changements successifs dans la fonction objectif ;
- la moyenne des distances Euclidiennes entre l'optimum global et les individus de chaque génération ;
- la moyenne, sur plusieurs exécutions d'un algorithme sur un même problème, des valeurs des meilleurs individus de chaque génération ;
- une comparaison du meilleur individu d'une génération avec le meilleur et le plus mauvais individu, dans un petit intervalle de générations récentes.

Afin de pouvoir nous comparer aux autres algorithmes proposés dans la littérature, nous allons utiliser les mesures de performance les plus répandues pour les jeux de tests que nous avons choisis pour notre évaluation. Ces mesures sont l'*offline error* pour MPB, ainsi qu'une mesure hybride, proposée spécifiquement pour GDBG. Nous allons voir, dans le paragraphe suivant, comment sont calculées ces mesures de performance au sein de ces deux jeux de tests.

1.4.2 Les principaux jeux de tests

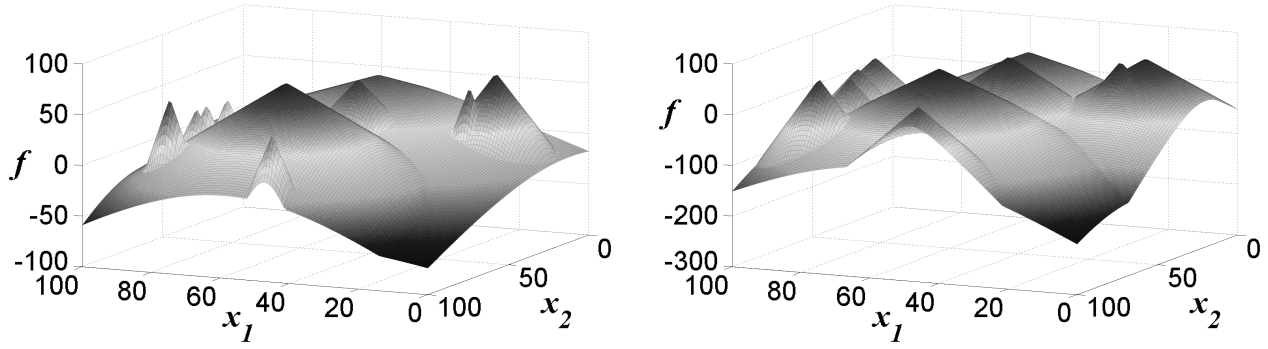
1.4.2.1 *Moving Peaks Benchmark* (MPB)

MPB est le jeu de tests le plus répandu en optimisation dynamique. Il s'agit d'un problème de maximisation, où la fonction objectif est composée d'un ensemble de pics (optima locaux) dont la forme, la position et la hauteur peuvent varier au cours du temps. Lors d'un changement, n'importe lequel de ces optima locaux peut devenir le nouvel optimum global. De plus, les déplacements aléatoires des pics sont autocorrélés par un coefficient λ, $0 \leq \lambda \leq 1$, où 0 correspond à des mouvements décorrélés et 1 correspond à des mouvements très corrélés entre eux. Les changements dans la fonction objectif ont lieu toutes les α évaluations. Une plage d'évaluations durant laquelle la fonction objectif ne change pas est appelée un *time span*. Un

paysage de MPB avant et après un changement (au cours de deux *time spans* consécutifs) est illustré dans la figure 1.7. La fonction utilisée pour générer le paysage de MPB est la suivante :

$$f(\vec{x},t) = \max_{i=1,\ldots,N_p} \left(H_i(t) - W_i(t) \left\| \vec{x} - \vec{X}_i(t) \right\| \right) \tag{1.4.1}$$

où N_p est le nombre de pics et $H_i(t)$, $W_i(t)$ et $\vec{X}_i(t)$ sont la hauteur, la largeur et la position du $i^{\text{ième}}$ pic au temps t, respectivement.



(a) (b)

FIGURE 1.7 – Un paysage de MPB avant et après un changement : (a) le paysage avant le changement, (b) le paysage après le changement. On constate une variation de la hauteur, de la largeur et de la position des pics.

Afin d'évaluer les performances de l'algorithme testé, une mesure appelée *offline error* est utilisée. Il s'agit de la moyenne des écarts entre la valeur de l'optimum global et celle de la meilleure solution trouvée à chaque évaluation. Une erreur de 0 signifie donc un suivi parfait de l'optimum. L'*offline error* (*oe*) est définie par l'équation (1.4.2) :

$$oe = \frac{1}{N_c} \sum_{j=1}^{N_c} \left(\frac{1}{N_e(j)} \sum_{i=1}^{N_e(j)} \left(f_j^* - f_{ji}^* \right) \right) \tag{1.4.2}$$

où N_c est le nombre total de changements ayant lieu dans la fonction objectif au cours d'un test, $N_e(j)$ est le nombre d'évaluations effectuées au cours du $j^{\text{ième}}$ *time span*, f_j^* est la valeur de l'optimum global au cours du $j^{\text{ième}}$ *time span*, et f_{ji}^* est la valeur de la meilleure solution trouvée par l'algorithme testé à la $i^{\text{ième}}$ évaluation du $j^{\text{ième}}$ *time span*.

Nous pouvons constater que cette mesure présente quelques faiblesses : elle est sensible à la hauteur moyenne du paysage, et au nombre de pics. Pour minimiser l'*offline error*, il est important pour l'algorithme testé de converger rapidement vers l'optimum global. Ainsi,

Paramètre	**Valeur**
Nombre de pics N_p	10
Dimension d	5
Hauteur des pics	$[30; 70]$
Largeur des pics	$[1; 12]$
Nombre d'évaluations d'un *time span* α	5000
Sévérité des déplacements des pics s	1
Sévérité des changements de hauteur des pics	7
Sévérité des changements de largeur des pics	1
Coefficient d'autocorrélation λ	0
Nombre de changements N_c	100

TABLEAU 1.1 – Paramétrage de MPB selon le scénario 2.

Paramètre	**Valeur**
Dimension (fixe) d	10
Dimension (variable) d	$[5; 15]$
Nombre d'évaluations d'un *time span* α	$10000 \times d$
Nombre de changements N_c	60

TABLEAU 1.2 – Paramétrage de GDBG lors de la compétition de CEC'2009.

l'approche la plus prometteuse serait basée sur un ensemble de solutions, permettant de garder la trace de chaque pic [Moser & Hendtlass, 2007].

Dans [Branke, 1999], trois jeux de paramètres, appelés scénarios, ont été proposés pour ce jeu de tests. Le plus répandu étant le scénario 2 (voir le tableau 1.1), c'est celui-ci que nous allons utiliser par la suite.

1.4.2.2 *Generalized Dynamic Benchmark Generator* (GDBG)

GDBG a été utilisé lors de la compétition du congrès IEEE CEC'2009, afin de tester des métaheuristiques dédiées à l'optimisation dynamique. Ce jeu de tests est constitué de 49 fonctions dynamiques. L'ensemble de ces fonctions est supposé être représentatif de la plupart des problèmes d'optimisation dynamique réels. GDBG est paramétrable, et comporte certains paramètres similaires à ceux de MPB. Nous utilisons ici le paramétrage de la compétition de CEC'2009 (voir le tableau 1.2). Ce jeu de tests donne globalement un score unique sur 100 à la fin de son exécution.

Les fonctions statiques utilisées comme base pour générer les fonctions dynamiques de GDBG sont celles de la table 1.3. Par rotation, composition et combinaison de ces fonctions statiques, six fonctions de test de différents degrés de difficulté sont obtenues, dont certaines sont illustrées dans la figure 1.8 :

Nom	**Fonction**	**Intervalle**
Sphère	$f(\vec{x}) = \sum_{i=1}^{d} x_i^2$	$[-100; 100]^d$
Rastrigin	$f(\vec{x}) = \sum_{i=1}^{d} (x_i^2 - 10\cos(2\pi x_i) + 10)$	$[-5; 5]^d$
Weierstrass	$f(\vec{x}) = \sum_{i=1}^{d} (\sum_{k=0}^{k_{max}} [a^k \cos(2\pi b^k (x_i + 0{,}5))]) - d \sum_{k=0}^{k_{max}} [a^k \cos(\pi b^k)]$ $a = 0{,}5; b = 3; k_{max} = 20$	$[-0{,}5; 0{,}5]^d$
Griewank	$f(\vec{x}) = \frac{1}{4000} \sum_{i=1}^{d} (x_i)^2 - \prod_{i=1}^{d} \cos(\frac{x_i}{\sqrt{i}}) + 1$	$[-100; 100]^d$
Ackley	$f(\vec{x}) = -20\exp(-0{,}2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}) - \exp(\frac{1}{d}\sum_{i=1}^{d} \cos(2\pi x_i)) + 20 + \exp(1)$	$[-32; 32]^d$

TABLEAU 1.3 – Fonctions statiques utilisées pour générer les fonctions de test de GDBG.

F_1 : fonction à pics tournée (à 10 et 50 pics)

F_2 : composition de la fonction Sphère

F_3 : composition de la fonction Rastrigin

F_4 : composition de la fonction Griewank

F_5 : composition de la fonction Ackley

F_6 : composition hybride de fonctions

Divers types de changements, plus ou moins chaotiques et brutaux, ont lieu après un certain nombre d'évaluations de chaque fonction de test. Au total, sept types de changements de différents degrés de difficulté ont été proposés :

T_1 : un déplacement faible

T_2 : un déplacement important

T_3 : un déplacement aléatoire selon une distribution Gaussienne

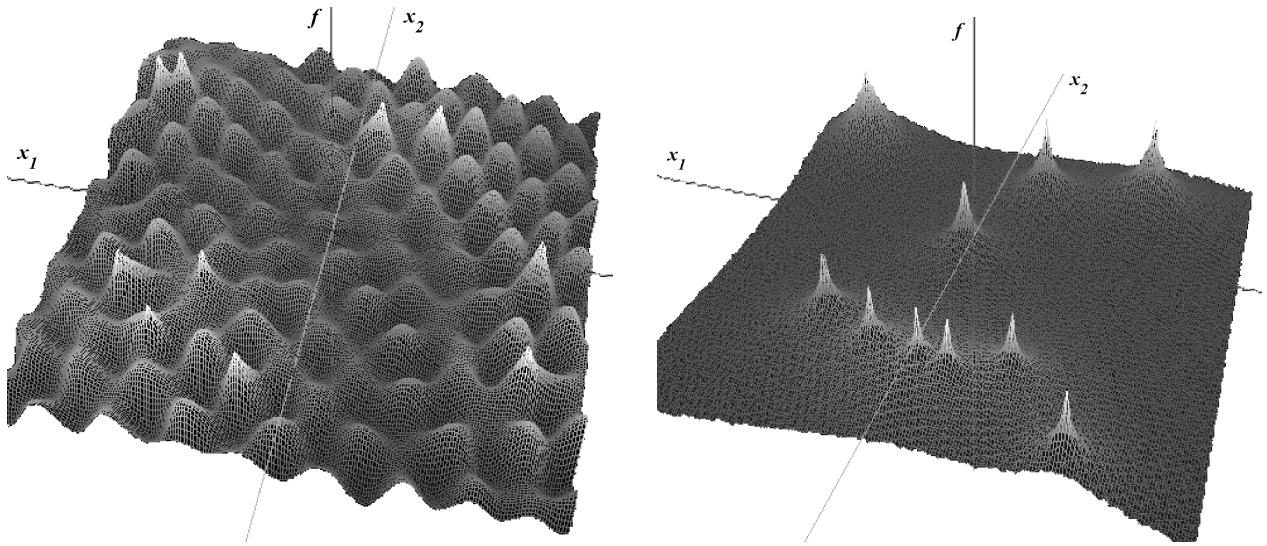
T_4 : un changement *chaotique* (selon une fonction logistique)

T_5 : un changement récurrent (un déplacement périodique)

T_6 : un changement récurrent bruité (comme T_6, mais l'optimum ne retourne jamais exactement au même point)

T_7 : un changement dans la dimension de l'espace de recherche

Ainsi, par combinaison de ces six fonctions de test avec ces sept types de changements (la fonction F_1 étant utilisée deux fois, avec 10 et 50 pics), nous obtenons un ensemble de 49



(a) (b)

FIGURE 1.8 – Illustration du paysage de fonctions de test de GDBG en utilisant 2 dimensions : (a) la fonction F_3, (b) la fonction F_5.

fonctions dynamiques. Pour chacune d'elles, la moyenne des meilleures valeurs (équation (1.4.3), des valeurs moyennes (équation (1.4.4)) et des moins bonnes valeurs (équation (1.4.5) trouvées, ainsi que l'écart-type (équation (1.4.6) de l'erreur absolue sont calculés :

$$Avg_{best} = \sum_{i=1}^{runs} \min_{j=1}^{N_c} \frac{E_{ij}}{runs} \tag{1.4.3}$$

$$Avg_{mean} = \sum_{i=1}^{runs} \sum_{j=1}^{N_c} \frac{E_{ij}}{runs \times N_c} \tag{1.4.4}$$

$$Avg_{worst} = \sum_{i=1}^{runs} \max_{j=1}^{N_c} \frac{E_{ij}}{runs} \tag{1.4.5}$$

$$STD = \sqrt{\frac{\sum_{i=1}^{runs} \sum_{j=1}^{N_c} (E_{ij} - Avg_{mean})^2}{runs \times N_c}} \tag{1.4.6}$$

où $E_{ij} = \left| f_j^* - \tilde{f}_{ji}^* \right|$, f_j^* est la valeur de l'optimum global au cours du $j^{\text{ième}}$ *time span*, $\tilde{f}_{ji}^*$ est la valeur de la meilleure solution trouvée au cours de la $i^{\text{ième}}$ exécution de l'algorithme testé, au $j^{\text{ième}}$ *time span*, et $runs$ est le nombre d'exécutions de l'algorithme testé, fixé à 20 dans nos expérimentations.

Les graphes de convergence, montrant l'évolution de l'erreur relative $r_i(t)$ lors de l'exécution de performance médiane pour chaque fonction dynamique, sont également calculés. Pour le

problème de maximisation F_1, l'expression (1.4.7) est utilisée pour calculer $r_i(t)$, et pour les problèmes de minimisation F_2 à F_6, l'expression (1.4.8) est utilisée.

$$r_i(t) = \frac{f_i(t)}{f_i^*(t)} \tag{1.4.7}$$

$$r_i(t) = \frac{f_i^*(t)}{f_i(t)} \tag{1.4.8}$$

où $f_i(t)$ est la valeur de la meilleure solution trouvée au temps t depuis la dernière occurrence d'un changement dans la fonction dynamique, à la $i^{\text{ième}}$ exécution de l'algorithme testé, et $f_i^*(t)$ est la valeur de l'optimum global au temps t.

Un score est calculé pour chaque fonction dynamique, selon l'expression (1.4.9) :

$$mark_{pct} = mark_{max} \times \sum_{i=1}^{runs} \sum_{j=1}^{N_c} \frac{r_{ij}}{runs \times N_c} \tag{1.4.9}$$

La somme de tous les scores $mark_{pct}$ donne un score total, noté *op*, qui correspond à la performance globale de l'algorithme testé. La valeur maximale de ce score est 100. Dans (1.4.9), r_{ij} est calculé selon l'expression (1.4.10), et $mark_{max}$ est un coefficient qui définit le pourcentage du score de chaque fonction dynamique intervenant dans le score final *op*. Il s'agit également de la valeur maximale du score $mark_{pct}$ que l'algorithme testé peut obtenir pour chaque fonction dynamique.

$$r_{ij} = \frac{r_i(t_j + \alpha)}{1 + \sum_{s=1}^{\frac{\alpha}{s_f}} \frac{1 - r_i(t_j + s_f \times s)}{\frac{\alpha}{s_f}}} \tag{1.4.10}$$

où t_j est l'instant auquel le $j^{\text{ième}}$ changement survient, s_f est la fréquence d'échantillonnage, fixée à 100, et α est le nombre d'évaluations d'un *time span*.

1.4.2.3 Classement sur MPB et GDBG

Nous allons maintenant présenter un classement des différents algorithmes d'optimisation dynamique, proposés dans la littérature, sur MPB (tableau 1.4) et sur GDBG (figure 1.9). Nous y reviendrons plus en détail dans les chapitres suivants, lors de la comparaison de ces algorithmes avec ceux élaborés durant cette thèse.

Les EAs ayant obtenu les meilleurs résultats sur les deux principaux jeux de tests en optimisation dynamique sont basés sur DE : [Mendes & Mohais, 2005] sur MPB (classé en dixième

Algorithme	***Offline error***
Moser & Chiong, 2010	$0{,}25 \pm 0{,}08$
Novoa et al., 2009	$0{,}40 \pm 0{,}04$
Moser & Hendtlass, 2007	$0{,}66 \pm 0{,}20$
Yang & Li, 2010	$1{,}06 \pm 0{,}24$
Liu et al., 2010	$1{,}31 \pm 0{,}06$
Lung & Dumitrescu, 2007	$1{,}38 \pm 0{,}02$
Bird & Li, 2007	$1{,}50 \pm 0{,}08$
Lung & Dumitrescu, 2008	$1{,}53 \pm 0{,}01$
Blackwell & Branke, 2006	$1{,}72 \pm 0{,}06$
Mendes & Mohais, 2005	$1{,}75 \pm 0{,}03$
Li et al., 2006	$1{,}93 \pm 0{,}06$
Blackwell & Branke, 2004	$2{,}16 \pm 0{,}06$
Parrott & Li, 2006	$2{,}51 \pm 0{,}09$
Du & Li, 2008	$4{,}02 \pm 0{,}56$

TABLEAU 1.4 – Classement des algorithmes en compétition sur MPB (non compris nos propres algorithmes).

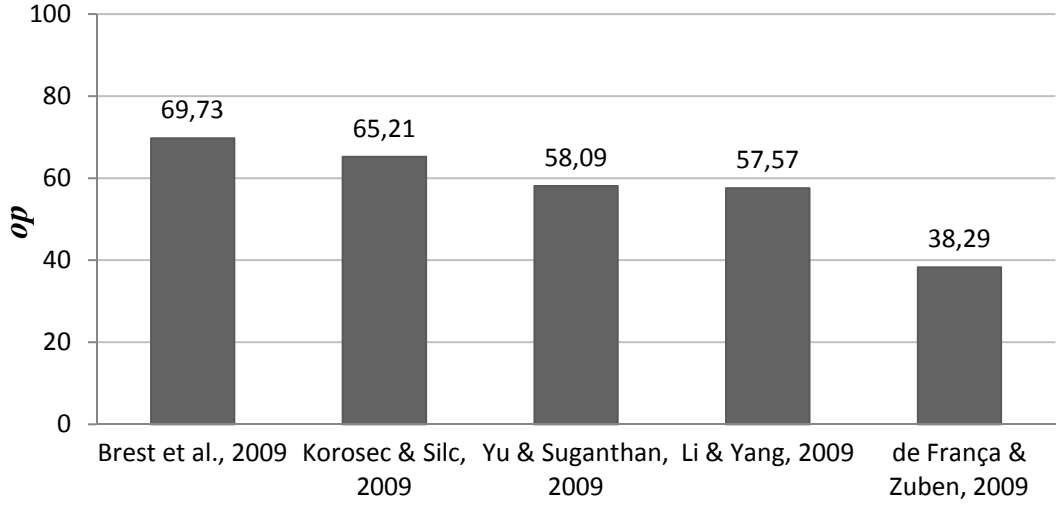


FIGURE 1.9 – Classement des algorithmes en compétition sur GDBG (non compris nos propres algorithmes).

position parmi quatorze algorithmes comparés sur MPB) et [Brest et al., 2009] sur GDBG (en tête du classement, parmi sept algorithmes comparés sur GDBG).

Parmi les algorithmes basés sur l'OEP, l'algorithme multi-essaims de [Li & Yang, 2009] est actuellement le seul à avoir été analysé en utilisant GDBG (classé à la quatrième place). Une version simplifiée de cet algorithme est analysée en utilisant MPB dans [Yang & Li, 2010]. En OEP, l'algorithme le mieux classé sur MPB est une variante de [Blackwell & Branke, 2006], proposée dans [Novoa et al., 2009] (classé à la deuxième place).

Pour les approches ACO et AIS, seul GDBG a été utilisé par les auteurs. Les seuls algo-

rithmes testés sont celui de [Korosec & Silc, 2009] (basé sur ACO, classé à la deuxième place) et celui de [de França & Zuben, 2009] (basé sur AIS, classé en dernière position).

Parmi les approches hybrides, l'algorithme *Hybridised EO*, proposé dans [Moser & Chiong, 2010], arrive en tête du classement sur MPB. Néanmoins, cet algorithme a été spécifiquement élaboré pour MPB, et aucune étude n'a été réalisée pour déterminer s'il peut être plus largement applicable.

1.5 Conclusion

Dans ce chapitre, nous avons défini ce qu'est un problème d'optimisation dynamique, puis nous avons présenté un état de l'art de l'optimisation dynamique. Nous avons décrit les mesures de performance existantes, ainsi que les jeux de tests les plus utilisés en optimisation continue dynamique.

Nous avons vu que l'approche principalement adoptée dans la littérature consiste à adapter des algorithmes d'optimisation statique à l'optimisation dynamique, en compensant leurs défauts intrinsèques. L'adaptation de ces algorithmes aux problèmes dynamiques a pour conséquence d'augmenter leur complexité. De plus, la performance d'un algorithme adapté à l'optimisation dynamique, en palliant ses défauts, peut s'avérer inférieure à celle d'un algorithme initialement conçu pour résoudre des problèmes dynamiques. De ce fait, plutôt que de partir d'un algorithme statique existant, il peut être intéressant d'élaborer un nouvel algorithme d'optimisation dynamique, sur la seule base des propriétés qu'un algorithme d'optimisation dynamique doit présenter (par exemple, converger rapidement, trouver les optima locaux et les suivre, conserver une bonne diversification tout au long de la recherche). De cette façon, il est idéalement possible de proposer un algorithme performant et peu complexe.

Sur cette idée, nous avons conçu, durant cette thèse, des algorithmes d'optimisation dynamique, que nous allons présenter dans les chapitres 2 et 3 suivants. Ces algorithmes sont analysés en utilisant les jeux de tests principalement utilisés dans la littérature, que nous venons de décrire.

Chapitre Deux

Elaboration de notre algorithme d'optimisation dynamique MADO

2.1 Introduction

Comme nous l'avons vu dans le chapitre précédent, il est impératif en optimisation dynamique de converger rapidement vers l'optimum global, afin de pouvoir le suivre au fil des *changements* dans la fonction objectif. Nous avons vu également que n'importe quel optimum local pouvait devenir, après un changement, le nouvel optimum global. De ce fait, il est important pour un algorithme d'optimisation dynamique de sauvegarder des informations sur la position des optima locaux trouvés durant sa recherche. Ces informations peuvent alors être exploitées pour accélérer la détection du nouvel optimum global après un changement. De plus, il est probable qu'un changement plus brutal engendre l'apparition d'un nouvel optimum global. L'algorithme doit alors assurer une diversification suffisante pour pouvoir le trouver.

Nous avons aussi vu que la plupart des métaheuristiques d'optimisation dynamique proposées dans la littérature sont bioinspirées. Elles appartiennent principalement à la classe des algorithmes évolutionnaires et à celle des essaims particulaires. Or, dans le cadre de l'optimisation statique, ces algorithmes ont tendance à ne converger que vers les meilleurs optima locaux (population des meilleurs individus), au détriment des autres. Leur but est en effet de découvrir l'optimum global, sans rester piégés dans des optima locaux. De plus, ils perdent progressivement leur potentiel de diversification au fil des itérations, censées assurer la convergence vers l'optimum global. Afin de les adapter à l'optimisation dynamique, il est alors nécessaire d'utiliser des techniques particulières pour pallier ces défauts.

L'approche principalement adoptée dans la littérature consiste ainsi à adapter ces algorithmes bioinspirés à l'optimisation dynamique, en compensant leurs défauts intrinsèques. Plutôt que d'emprunter cette voie, déjà largement explorée, nous avons préféré concevoir un algorithme entièrement pensé pour l'optimisation dynamique. Notre choix s'est alors porté sur un algorithme à base d'agents de recherche locale coordonnés : MADO (*MultiAgent Dynamic Optimization*) [Lepagnot et al., 2009, 2010a,b,c].

A la différence des métaheuristiques, les algorithmes de recherche locale ne prennent pas le temps d'explorer globalement l'espace de recherche. Leur but est de converger le plus rapidement possible vers l'optimum local le plus proche. De ce fait, ils présentent deux qualités majeures en optimisation dynamique : une convergence rapide et une bonne détection des optima locaux. L'exploration globale de l'espace de recherche, en vue de découvrir l'optimum global, peut ensuite se faire par l'utilisation judicieuse et coordonnée de plusieurs recherches locales en parallèle.

L'algorithme proposé utilise ainsi une population coordonnée d'agents explorant continuellement et parallèlement l'espace de recherche. La coordination permanente des agents garantit une bonne couverture de l'espace de recherche, ce qui confère à l'algorithme un fort potentiel de diversification tout au long de son exécution. L'intensification est quant à elle assurée par l'utilisation de recherches locales, effectuées de façon indépendante par les agents. Ces recherches locales permettent une détection rapide et large des optima locaux.

Dans ce chapitre, nous allons décrire l'algorithme MADO en détail, en présenter une analyse expérimentale, ainsi qu'une comparaison de ses performances avec celles des autres algorithmes d'optimisation dynamique proposés dans la littérature.

2.2 Description de l'algorithme

MADO a été proposé pour résoudre des problèmes d'optimisation continue dynamique sans contraintes. Afin de bien comprendre le fonctionnement de cet algorithme, nous commencerons par décrire sa structure générale, puis nous verrons en détail chacune des stratégies mises en œuvre.

2.2.1 Structure générale

MADO est un algorithme à architecture multi-agent. En conséquence, il fait appel à une population d'agents pour explorer l'espace de recherche. Ces agents sont supposés « myopes » car ils n'ont qu'une vision locale de l'espace de recherche. Plus précisément, ces agents effectuent en permanence des recherches locales, en se déplaçant pas à pas depuis leur solution courante vers une meilleure solution avoisinante, jusqu'à ce qu'ils ne puissent plus améliorer leur solution courante, atteignant ainsi un état de stagnation caractéristique d'un optimum local. Un sous-ensemble des optima locaux ainsi trouvés par les agents est alors sauvegardé afin d'accélérer la convergence de l'algorithme. Le nombre d'agents existants, noté N, évolue en fonction des conditions rencontrées au fil des itérations (initialisation de MADO, changements affectant

la fonction objectif, saturation en agents de l'espace de recherche). La structure générale de MADO est illustrée à la figure 2.1. Elle est composée des deux modules suivants :

1. **Le module de gestion de la mémoire :** dans le cas d'un problème multimodal, un algorithme d'optimisation dynamique doit garder une trace des optima locaux trouvés, car l'un d'eux peut à tout moment devenir le nouvel optimum global. De ce fait, nous proposons de sauvegarder les optima locaux trouvés dans une archive. Cette archive est alors maintenue par un module dédié, le module de gestion de la mémoire.
2. **Le coordinateur :** à la différence des agents, le coordinateur a une vision globale de l'espace de recherche. Il supervise la recherche effectuée par les agents, et il leur évite de chercher dans des zones non prometteuses de l'espace de recherche. Les agents sont ainsi coordonnés pour explorer au mieux l'espace de recherche et garantir la diversité des solutions. Le coordinateur gère également les interactions entre le module de gestion de la mémoire et les agents. Il est notamment informé des optima locaux trouvés, afin de les transmettre au module de gestion de la mémoire pour archivage. Enfin, le coordinateur gère la création, la destruction et le repositionnement des agents.

Nous allons maintenant décrire l'étape d'initialisation indiquée à la figure 2.1, ainsi que la procédure de recherche exécutée par les agents. A cet effet, il est tout d'abord nécessaire de présenter la manière dont les distances sont calculées dans l'espace de recherche. Le critère d'arrêt de la figure 2.1 dépend, quant à lui, du problème d'optimisation traité.

2.2.1.1 Calcul des distances

Nous proposons de définir l'espace de recherche en tant qu'espace Euclidien à d dimensions, car c'est le plus simple et le plus utilisé. Un espace Euclidien est un espace vectoriel V comportant un nombre fini de dimensions, et équipé d'une norme Euclidienne $\|\cdot\|$. Une norme Euclidienne est une fonction $V \to [0; +\infty[$ donnée par $\vec{x} \mapsto \sqrt{\langle \vec{x}, \vec{x} \rangle}$, où $\langle \cdot, \cdot \rangle$ est un produit scalaire.

L'espace Euclidien utilisé pour l'espace de recherche étant $\mathbb{R}^d$, le produit scalaire est donné par l'expression (2.2.1), et la norme Euclidienne est donnée par l'expression (2.2.2).

$$\langle \vec{x}, \vec{y} \rangle = \sum_{i=1}^{d} x_i \, y_i \tag{2.2.1}$$

$$\|\vec{x}\| = \sqrt{\sum_{i=1}^{d} (x_i)^2} \tag{2.2.2}$$

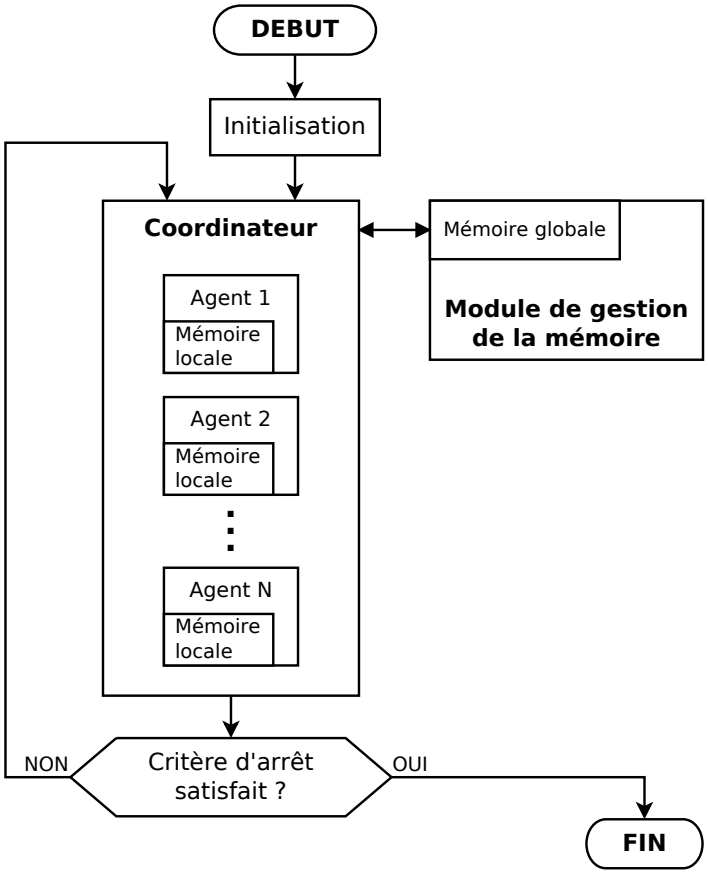


FIGURE 2.1 – Schéma fonctionnel de MADO. Le nombre d'agents existants, variable, est noté N.

Ensuite, comme l'espace de recherche n'a pas nécessairement les mêmes bornes pour chaque dimension, nous proposons d'utiliser une base « normalisée ».

Soit Δ_i la longueur de chaque intervalle définissant l'espace de recherche, où $i \in \{1, ..., d\}$. Les vecteurs unitaires $(\vec{e}_i)$ formant la base canonique de $\mathbb{R}^d$ sont alors utilisés pour former une base normalisée de vecteurs $(\vec{u}_i)$, définis par $\{\vec{u}_1 = \Delta_1 \, \vec{e}_1, \vec{u}_2 = \Delta_2 \, \vec{e}_2, ..., \vec{u}_d = \Delta_d \, \vec{e}_d\}$. Ce changement de base transforme ainsi un espace de recherche hyper-rectangulaire en un espace de recherche hyper-cubique, selon l'équation (2.2.3) :

$$x'_i = \frac{x_i}{\Delta_i} \text{ pour } i = 1, 2, ..., d \tag{2.2.3}$$

où x'_i et x_i sont les $i^{\text{ièmes}}$ coordonnées d'une solution exprimée dans l'espace hyper-cubique, et dans l'espace hyper-rectangulaire, respectivement.

2.2.1.2 Initialisation de MADO

A l'initialisation, le coordinateur commence par créer un ensemble initial d'agents. Le nombre d'agents créés est donné par un paramètre de l'algorithme, noté n_a. Ces agents ne

sont pas disposés aléatoirement dans l'espace de recherche, mais de manière à être espacés le plus possible. Plus précisément, nous maximisons la distance entre les deux plus proches agents. Nous obtenons ainsi une bonne couverture de l'espace de recherche par les agents, afin d'améliorer la diversité des solutions et de potentiellement découvrir plus rapidement l'optimum global.

Pour ce faire, nous proposons d'utiliser une heuristique de répulsion électrostatique issue de [Conway & Sloane, 1998]. Cette heuristique considère un ensemble de points de l'espace comme des particules chargées se repoussant mutuellement. A partir d'un ensemble arbitraire de points, chaque point est repoussé itérativement selon une force en $\frac{1}{r^2}$, en désignant par r la distance entre deux points. L'heuristique est exécutée jusqu'à la satisfaction d'un critère d'arrêt.

Nous utilisons ainsi cette heuristique pour positionner les agents initiaux dans l'espace de recherche, à partir d'un ensemble de n_a points. Ces points sont engendrés aléatoirement dans un hyper-cube, noté T, de dimension d et centré dans l'espace de recherche. Comme illustré à la figure 2.2, T est le résultat d'une homothétie de l'espace de recherche, dilatant les distances par rapport à un point fixe, appelé le centre d'homothétie. Le rapport d'homothétie est la valeur par laquelle les distances sont multipliées. Ici, le centre d'homothétie est le centre de l'espace de recherche, et le rapport d'homothétie, noté δ_e, est donné par l'équation (2.2.4) :

$$\delta_e = \frac{n_a - 1}{n_a} \tag{2.2.4}$$

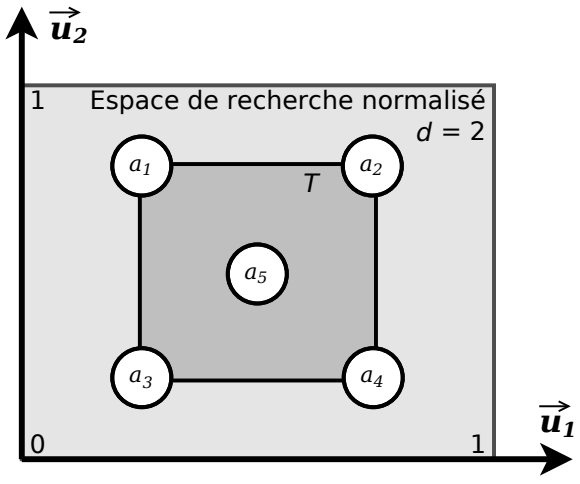


FIGURE 2.2 – Génération d'un ensemble initial de 5 agents dans un espace de recherche à 2 dimensions. Les agents sont représentés par des disques blancs notés a_i, avec $i \in \{1, 2, ..., 5\}$.

L'heuristique est exécutée jusqu'à ce qu'aucun point ne se déplace d'une distance supérieure à ϵ, typiquement égal à 10^{-4}. Le pseudo-code de cette heuristique est présenté dans la

procédure 2.1.

A chaque itération de l'heuristique, la force de répulsion $\vec{F}$ s'appliquant au point $\vec{P_i}$ est calculée à la ligne 8 de ce pseudo-code. Ensuite, le coefficient *pas* est adapté de façon à maximiser le déplacement $(pas \times \vec{F})$ du point $\vec{P_i}$, sous la contrainte d'absence d'accroissement de l'énergie e du système. Pour empêcher le point P_i de se déplacer hors de l'hyper-cube T, nous le maintenons en son sein à la ligne 12. L'ensemble des points résultant de l'exécution de ce pseudo-code nous permet d'obtenir l'ensemble des positions initiales des agents, à l'étape d'initialisation de MADO.

2.2.1.3 La procédure de recherche des agents

Les agents effectuent leurs recherches locales indépendamment les uns des autres. La procédure exécutée en boucle par un agent est illustrée à la figure 2.3. Sur cette figure, les états nommés « SYNCHRONISATION A » et « SYNCHRONISATION B » marquent la fin d'une itération de la procédure : lorsqu'un agent atteint l'un ou l'autre de ces deux états, il interrompt son exécution, jusqu'à ce que tous les autres agents aient également atteint l'un de ces états. Ensuite, les agents reprennent leur exécution de la manière suivante : si un agent s'est interrompu dans l'état SYNCHRONISATION A (respectivement, SYNCHRONISATION B), alors il reprend son exécution à partir de l'état SYNCHRONISATION A (respectivement, SYNCHRONISATION B). De cette façon, les agents peuvent opérer en parallèle.

2.2.2 Les stratégies d'optimisation de MADO

Nous allons maintenant décrire, en détail, les stratégies que nous proposons pour détecter et suivre efficacement les optima locaux d'un problème d'optimisation dynamique. Ces stratégies sont utilisées par les agents, le coordinateur et le module de gestion de la mémoire.

2.2.2.1 La stratégie d'exploration utilisée par les agents

Les agents explorent l'espace de recherche en s'y déplaçant pas à pas, de leur solution courante vers une meilleure solution voisine. Pour bien appréhender le comportement d'un agent, nous allons commencer par décrire le voisinage utilisé par les agents. Comme un agent n'effectue que des recherches locales, il ne teste que des solutions situées dans une zone délimitée de l'espace de recherche, centrée sur sa position courante. En l'absence d'information sur le paysage local de l'espace de recherche, nous supposons que le voisinage est isotrope. Dans ce cas, nous proposons d'utiliser la topologie d'une boule (sphère pleine). Ensuite, pour minimiser les évaluations de la fonction objectif, les solutions de ce voisinage, qui seront testées par l'agent,

répulsion

Entrées : n_a, ϵ
Variables locales : $pas, \delta_{max}, \vec{P}_i, \vec{P}_j, \vec{F}, e, \vec{P}'_i, \vec{P}^*_i$
$P \leftarrow$ un ensemble de n_a points engendrés aléatoirement dans l'hyper-cube T
$pas \leftarrow 1$
répéter
  $\delta_{max} \leftarrow 0$
  pour tout point $\vec{P}_i \in P$ **faire**
    $\vec{F} \leftarrow \sum_{j \neq i} \frac{\vec{P}_i - \vec{P}_j}{\left\|\vec{P}_i - \vec{P}_j\right\|^3}$
    $e \leftarrow \sum_{j \neq i} \frac{1}{\left\|\vec{P}_i - \vec{P}_j\right\|^2}$
    répéter
      $\vec{P}'_i \leftarrow \vec{P}_i + pas \times \vec{F}$
      $\vec{P}^*_i \leftarrow$ le point dans T le plus proche de $\vec{P}'_i$
      si $e < \sum_{j \neq i} \frac{1}{\left\|\vec{P}^*_i - \vec{P}_j\right\|^2}$ **alors**
        $pas \leftarrow \frac{pas}{2}$
      sinon
        $pas \leftarrow pas \times 2$
      fin
    tant que pas peut être adapté et $e < \sum_{j \neq i} \frac{1}{\left\|\vec{P}^*_i - \vec{P}_j\right\|^2}$
    si $\|\vec{P}_i - \vec{P}^*_i\| > \delta_{max}$ **alors**
      $\delta_{max} \leftarrow \|\vec{P}_i - \vec{P}^*_i\|$
    fin
    $P \leftarrow (P - \vec{P}_i) \cup \{\vec{P}^*_i\}$
  fin
tant que $\delta_{max} > \epsilon$
retourner P

PROCÉDURE 2.1 – Heuristique de répulsion électrostatique utilisée pour positionner les agents initiaux.

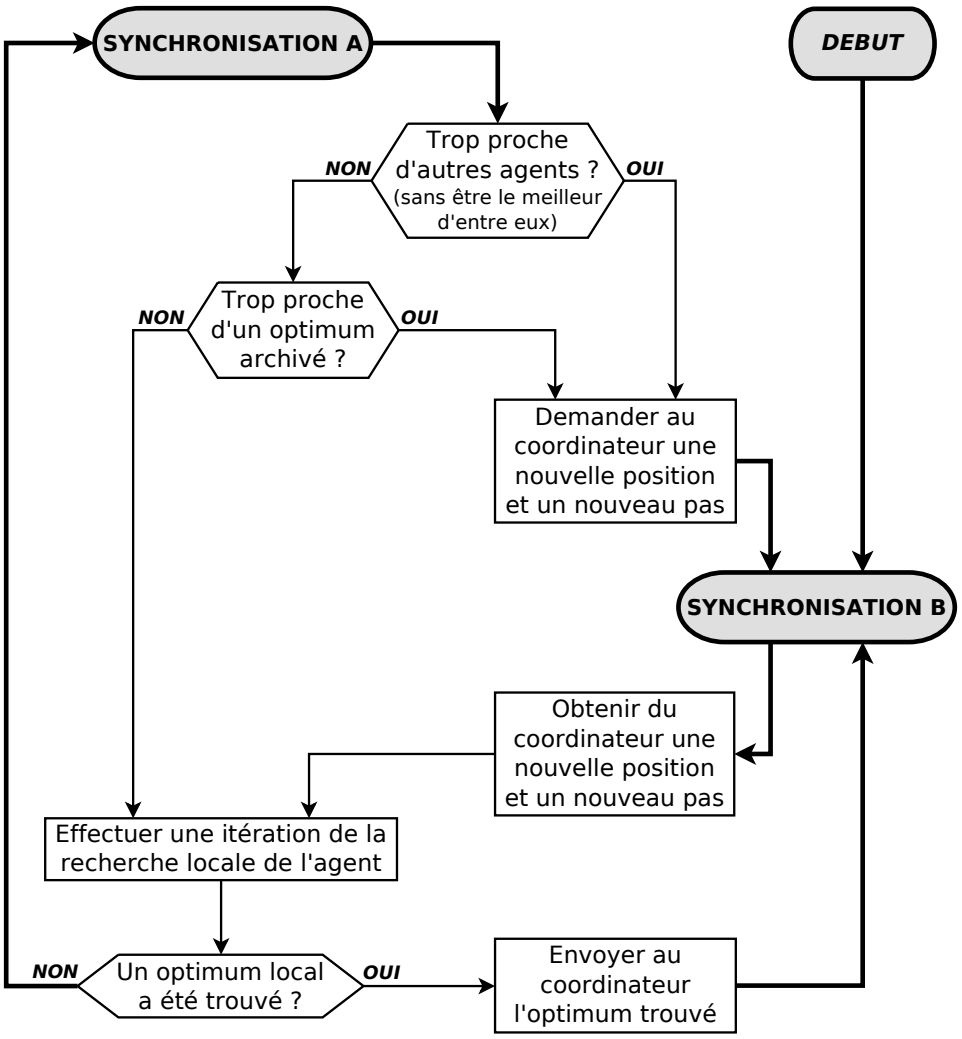


FIGURE 2.3 – Organigramme de la procédure principale d'un agent de MADO.

doivent couvrir au mieux le paysage local de l'agent. Nous proposons alors de maximiser les distances entre ces solutions candidates. Nous avons choisi de les échantillonner à la périphérie d'une boule : une hypersphère centrée sur la solution courante de l'agent. De plus, le voisinage d'un agent doit pouvoir s'adapter au paysage local de l'espace de recherche, pour accroître l'efficacité de la recherche locale et assurer la convergence de l'agent. Cela se fait en adaptant le rayon de l'hypersphère au fur et à mesure des déplacements de l'agent.

Nous définissons donc le voisinage d'un agent comme un ensemble de n_s solutions candidates réparties sur une hypersphère de rayon R et centrée sur la solution courante de l'agent. Le nombre n_s de solutions candidates est un paramètre de MADO. Ce voisinage est illustré à la figure 2.4, pour un espace de recherche à deux dimensions.

Afin de créer cet ensemble de solutions candidates, un ensemble de n_s points est engendré et sauvegardé à l'initialisation de MADO. Ces points sont placés de façon à satisfaire les contraintes

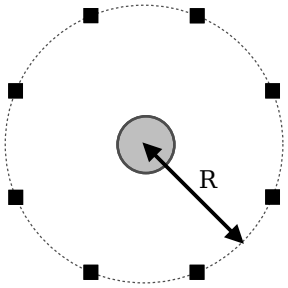


FIGURE 2.4 – Illustration du voisinage d'un agent pour $d = 2$ et $n_s = 8$. La solution courante est représentée par un disque gris, et les solutions candidates sont représentées par des carrés noirs.

suivantes :

- la plus petite distance entre deux points de l'ensemble est maximisée ;
- la distance entre chaque point et l'origine de l'espace (le point $(0, 0, ..., 0)$) est égale à 1 ;
- la dimension de l'espace dans lequel ces points sont placés est égale à celle de l'espace de recherche.

Cet ensemble de points est créé en utilisant l'heuristique de répulsion électrostatique décrite précédemment, au paragraphe 2.2.1.2. A partir de cet ensemble de points, noté S, un agent peut ensuite obtenir l'ensemble des solutions candidates de son voisinage, noté S', au moyen de la procédure 2.2. Dans cette procédure, la solution courante de l'agent est notée $\vec{S}_c$.

voisinage

Entrées : $d, S, R, \vec{S}_c$
Variables locales : $\vec{V}, \vec{P}_i, \vec{P}_r, \vec{P}_{rs}, \vec{P}_{rst}$
$S' \leftarrow \varnothing$
$\vec{V} \leftarrow$ un vecteur unitaire orienté vers un point placé aléatoirement selon une distribution uniforme sur l'hypersphère de rayon 1, de dimension d et centrée sur l'origine
pour tout point $\vec{P}_i \in S$ **faire**
    $\vec{P}_r \leftarrow \vec{P}_i - 2 \times \left\langle \vec{V}, \vec{P}_i \right\rangle \times \vec{V}$
    $\vec{P}_{rs} \leftarrow R \times \vec{P}_r$
    $\vec{P}_{rst} \leftarrow \vec{P}_{rs} + \vec{S}_c$
    si $\vec{P}_{rst}$ ne dépasse pas les bornes de l'espace de recherche **alors**
        $S' \leftarrow S' \cup \{\vec{P}_{rst}\}$
    fin
fin
retourner S'

PROCÉDURE 2.2 – Placement des solutions candidates appartenant au voisinage d'un agent.

A partir de chaque point $\vec{P_i}$ de S, la procédure crée une solution candidate qu'elle ajoute à S'. Elle commence par créer (à la ligne 4) un vecteur unitaire $\vec{V}$ orienté aléatoirement. Ensuite, une transformation de Householder [Householder, 1958] est appliquée (à la ligne 6) à $\vec{P_i}$, en utilisant $\vec{V}$ comme vecteur de Householder. Cette transformation permet ainsi de pivoter aléatoirement les points de S. Pour chaque point $\vec{P_i}$, nous obtenons alors un point transformé $\vec{P_r}$. Puis, à la ligne 7, une homothétie centrée sur l'origine de l'espace et de rapport R est appliquée à $\vec{P_r}$. Nous obtenons alors, pour chaque point $\vec{P_r}$ appartenant à l'hypersphère de rayon 1 et centrée sur l'origine, un point $\vec{P_{rs}}$ appartenant à l'hypersphère de rayon R et centrée sur l'origine. Enfin, à la ligne 8, chaque point $\vec{P_{rs}}$ est translaté pour donner un point $\vec{P_{rst}}$ appartenant à l'hypersphère de centre $\vec{S_c}$ et de rayon R. Les points $\vec{P_{rst}}$ qui ne sont pas en dehors des bornes de l'espace de recherche constituent alors l'ensemble S'.

2.2.2.2 La stratégie d'adaptation du pas d'un agent

L'adaptation du pas R d'un agent est basée sur la trajectoire suivie par l'agent au cours de sa recherche locale. En effet, comme illustré sur la figure 2.5, la nécessité d'adapter le pas d'un agent peut être déduite de l'historique de ses déplacements successifs.

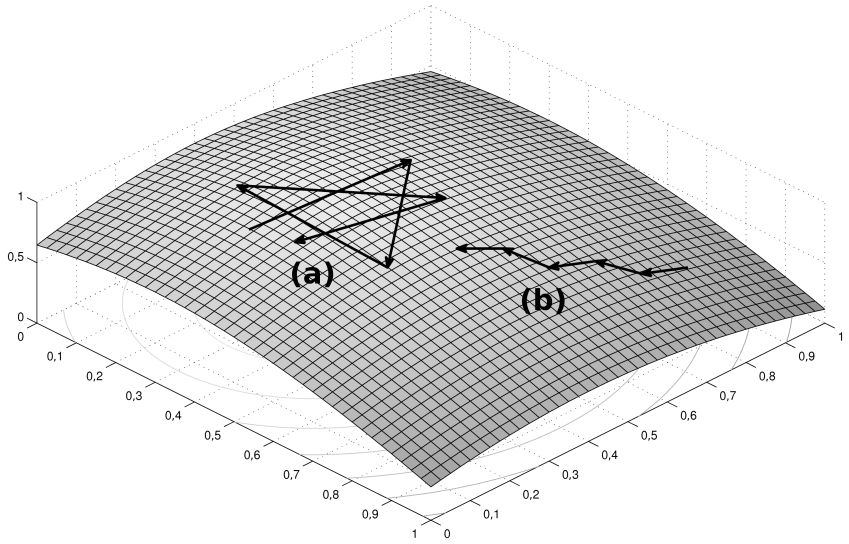


FIGURE 2.5 – Illustration de 2 trajectoires conduisant à une adaptation du pas. (a) Un agent tournant autour d'un optimum local. (b) Un agent remontant une large pente.

Sur cette figure, sont illustrées deux trajectoires pouvant être suivies par un agent :

– La trajectoire (a) correspond aux déplacements d'un agent tournant autour d'un optimum local, sans pouvoir l'atteindre. Ceci est dû au fait que le pas de l'agent est trop grand, ce qui engendre des déplacements avant-arrière, qui s'annulent les uns les autres. Il est donc

nécessaire, lorsqu'un agent suit une trajectoire telle que (a), de réduire le pas de l'agent, afin de lui permettre de converger.

– A l'inverse, les déplacements de la trajectoire (b) sont orientés dans une même direction. Plutôt que d'effectuer plusieurs petits déplacements consécutifs dans une même direction, l'agent pourrait, dans ce cas, accélérer sa recherche, en se déplaçant avec un pas plus grand. En présence d'une trajectoire telle que (b), il est donc nécessaire d'augmenter le pas de l'agent, afin d'économiser des évaluations de la fonction objectif.

Cette idée d'adaptation est inspirée de celle proposée dans [Hansen & Ostermeier, 2001]. Plutôt que de se baser nous aussi sur la distance parcourue par un agent, à l'issue de ses déplacements successifs, pour adapter son pas, nous proposons de l'adapter en fonction des changements d'orientation de ses déplacements. Les calculs que nous proposons pour adapter le pas sont simples, et se prêtent bien à notre algorithme. Ils sont basés sur le « produit scalaire cumulé », noté U_n, des n déplacements successifs de l'agent, depuis l'initialisation de sa recherche locale.

Soit $\vec{D}_n$ le vecteur directeur unitaire du $n^{\text{ième}}$ déplacement de l'agent depuis l'initialisation de sa recherche locale. Soit c_u une constante réelle dans $[0;1]$. Alors U_n est calculé comme suit :

$$U_n = \begin{cases} c_u \times U_{n-1} + \langle \vec{D}_{n-1}, \vec{D}_n \rangle & \text{si } n > 0 \\ \\ 0 & \text{sinon} \end{cases} \tag{2.2.5}$$

On remarque qu'avec $c_u = 0$, le produit scalaire cumulé ne dépend que des deux derniers déplacements de l'agent. En revanche, pour $c_u = 1$, les deux derniers déplacements ont autant de poids que les précédents dans le calcul de U_n. Il s'agit en effet d'un paramètre de MADO permettant d'indiquer l'importance à accorder à l'historique des déplacements de l'agent.

Le choix d'augmenter ou de réduire le pas de l'agent est ensuite effectué en fonction de U_n. En effet, lorsque l'agent suit une trajectoire telle que celle illustrée en (a) à la figure 2.5, les valeurs de U_n sont fortement négatives. En revanche, lorsque l'agent suit une trajectoire telle que celle illustrée en (b), les valeurs de U_n sont fortement positives. Nous proposons donc de réduire le pas de l'agent lorsque U_n devient inférieur à un certain *seuil négatif* $(-s_u)$, et d'augmenter le pas de l'agent lorsque U_n devient supérieur à un certain *seuil positif* $(+s_u)$. Cette adaptation du pas est présentée dans la procédure 2.3, où c_r une constante réelle dans $[0;1]$.

Dans cette procédure, on remarque que plus c_r est petit, et plus le pas est réduit (ligne 4) ou

miseAJourDuPas

1 **Entrées :** U_n, s_u, R, c_r
2 mise à jour de U_n selon l'équation (2.2.5)
3 **si** $U_n < -s_u$ **alors**
4 $\quad R \leftarrow c_r \times R$
5 $\quad U_n \leftarrow 0$
6 **sinon si** $U_n > s_u$ **alors**
7 $\quad R \leftarrow \frac{1}{c_r} \times R$
8 $\quad U_n \leftarrow 0$
9 **fin**
10 **retourner** $\{U_n, R\}$

PROCÉDURE 2.3 – Adaptation du pas d'un agent au moyen du produit scalaire cumulé. Le produit scalaire cumulé est noté U_n, le seuil d'adaptation du pas est noté s_u, le pas est noté R et le coefficient d'adaptation du pas est noté c_r.

augmenté (ligne 7). Ce coefficient d'adaptation du pas est un paramètre de MADO. La valeur de s_u est, quant à elle, calculée comme suit :

$$s_u = cos(\frac{\pi}{3})\ (1 + c_u) \tag{2.2.6}$$

Cette expression, définie de manière empirique, se justifie de la manière suivante : lorsque $c_u = 0$, alors U_n est égal au cosinus de l'angle, noté a, entre les vecteurs directeurs des deux derniers déplacements de l'agent. Comme $a \in [0; \pi]$, nous partitionnons cet intervalle de façon uniforme en trois intervalles, correspondant aux trois cas suivants :

1. L'agent suit une trajectoire comme celle illustrée en (a) à la figure 2.5. Ce cas correspond à des déplacements avant-arrière, et l'angle entre les vecteurs directeurs de deux déplacements consécutifs est donc important. Ce cas, conduisant à une réduction de R, est alors associé aux valeurs de a appartenant à l'intervalle $[\frac{2\ \pi}{3}; \pi]$.
2. L'agent suit une trajectoire ne nécessitant pas d'adaptation de R. Ce cas est associé aux valeurs de a appartenant à l'intervalle $[\frac{\pi}{3}; \frac{2\ \pi}{3}]$.
3. L'agent suit une trajectoire comme celle illustrée en (b) à la figure 2.5. Ce cas correspond à des déplacements orientés dans une même direction, et l'angle entre les vecteurs directeurs de deux déplacements consécutifs est donc faible. Ce cas, conduisant à une augmentation de R, est associé aux valeurs de a appartenant à l'intervalle $[0; \frac{\pi}{3}]$.

Ainsi, lorsque $c_u = 0$, ce partitionnement revient à définir $s_u = cos(\frac{\pi}{3})$. Ensuite, pour que ce seuil soit adapté à toute valeur de $c_u \in [0; 1]$, nous proposons d'utiliser l'expression (2.2.6).

Cette adaptation de R suppose que l'agent soit toujours capable de se déplacer vers une meilleure solution voisine. Cependant, lorsqu'aucune solution voisine n'est meilleure que la solution courante de l'agent, aucun déplacement n'est alors possible et l'agent reste sur sa solution courante. Dans ce cas, nous proposons d'utiliser une autre stratégie d'adaptation du pas, basée sur l'hypothèse suivante : si un agent ne peut plus trouver de meilleure solution voisine, cela signifie qu'il a convergé vers un optimum local, et que la distance entre cet optimum local et la solution courante de l'agent est inférieure à R. Afin que l'agent puisse continuer sa convergence vers cet optimum local, son pas est réduit ($R \leftarrow c_r\ R$).

2.2.2.3 La recherche locale d'un agent

On considère qu'un agent a convergé vers un optimum local lorsque son pas devient inférieur à r_l et lorsqu'aucun de ses δ_t derniers déplacements (de sa solution courante vers une meilleure solution voisine) n'a pu améliorer significativement (de plus de δ_p) sa solution courante. Ainsi, le critère d'arrêt de la recherche locale d'un agent est comme suit :

$$stop = \begin{cases} vrai & \text{si } R < r_l \text{ et } |f_n - f_{n-1}| \leq \delta_p, \text{ pour } n = 1, 2, ..., \delta_t \\ \\ faux & \text{sinon} \end{cases} \tag{2.2.7}$$

où r_l est un paramètre de MADO que nous verrons par la suite, f_n est la valeur de la fonction objectif de la solution courante trouvée par l'agent, et n est la valeur d'un compteur comptabilisant les δ_t derniers déplacements de l'agent. Ici, δ_t et δ_p sont deux paramètres de MADO, fixés par l'utilisateur, δ_t étant un entier strictement positif et δ_p une constante réelle exprimant la précision de la recherche locale. L'utilisation de δ_t et de δ_p constitue un critère de stagnation très utilisé dans les algorithmes de recherche locale, mais elle introduit, malgré tout, deux paramètres devant être adaptés.

Lorsque ce critère d'arrêt est satisfait, l'optimum local ainsi trouvé est transmis au module de gestion de la mémoire via le coordinateur, afin d'être archivé. Le coordinateur fournit ensuite une nouvelle position et un nouveau pas à l'agent, afin qu'il débute une nouvelle recherche locale dans une zone inédite de l'espace de recherche. Nous verrons ultérieurement comment la nouvelle position et le nouveau pas de l'agent sont calculés (paragraphe 2.2.2.5), et comment cette archive d'optima locaux est maintenue (paragraphe 2.2.2.7).

Afin de minimiser les évaluations de la fonction objectif, nous proposons également d'adapter le nombre de solutions voisines évaluées, à chaque déplacement de l'agent, en fonction du « degré de convergence » de l'agent. A la différence du critère d'arrêt, qui n'indique que de façon binaire si un agent a convergé ou non vers un optimum local, ce degré de convergence nous permet de quantifier numériquement la convergence de l'agent. Cela nous permet alors d'effectuer une transition souple entre les phases d'exploration et d'intensification d'un agent. En effet, en phase d'exploration, nous partons du principe qu'un agent doit évaluer toutes ses solutions voisines, afin d'explorer largement l'espace de recherche. En revanche, en phase d'intensification, l'agent peut se contenter d'évaluer des solutions voisines jusqu'à ce qu'il en trouve une qui soit meilleure que sa solution courante. N'évaluer ainsi qu'une partie de ses solutions voisines n'empêche pas l'agent de converger vers un optimum local, et permet d'économiser des évaluations de la fonction objectif.

Nous proposons d'exprimer le degré de convergence d'un agent par le nombre, noté d_t, de ses derniers déplacements successifs n'ayant pas permis d'améliorer significativement sa solution courante (autrement dit, n'ayant pas amélioré la valeur de sa solution courante de plus de δ_p). La recherche d'une meilleure solution voisine d'un agent s'effectue alors selon la procédure 2.4.

sélection

Entrées : $\vec{S}_c, S', R, r_l, \delta_t, d_t$
Variables locales : $\vec{S}_i$
$\vec{S}_{best} \leftarrow \vec{S}_c$
pour toute solution $\vec{S}_i \in S'$ **faire**
  si $\vec{S}_i$ est strictement meilleure que $\vec{S}_{best}$ **alors**
    $\vec{S}_{best} \leftarrow \vec{S}_i$
  fin
  si $\vec{S}_{best}$ est strictement meilleure que $\vec{S}_c$ **alors**
    si $R < r_l$ et $rand(\delta_t) < d_t$ **alors**
      retourner $\vec{S}_{best}$
    fin
  fin
fin
retourner $\vec{S}_{best}$

PROCÉDURE 2.4 – Recherche effectuée par un agent pour trouver une solution voisine, notée $\vec{S}_{best}$, meilleure que sa solution courante $\vec{S}_c$. La fonction $rand(\delta_t)$ permet de tirer un nombre réel aléatoire, selon une distribution uniforme dans $[0; \delta_t]$.

La procédure principale de la recherche locale d'un agent est alors présentée à la procédure 2.5. Cette procédure correspond à l'état indiqué à la figure 2.3 par « effectuer une itération de la recherche locale de l'agent ».

rechercheLocale

Entrées : $d, S, R, \vec{S}_c, r_l, \delta_t, d_t, U_n, s_u, c_r$
Variables locales : $S', stop$
$S' \leftarrow \text{voisinage}\left(d, S, R, \vec{S}_c\right)$
$\vec{S}_c \leftarrow \text{sélection}\left(\vec{S}_c, S', R, r_l, \delta_t, d_t\right)$
$stop \leftarrow$ valeur de l'expression (2.2.7)
$\{U_n, R\} \leftarrow \text{miseAJourDuPas}(U_n, s_u, R, c_r)$
si $stop = vrai$ **alors**
  critère d'arrêt de la recherche locale satisfait : $\vec{S}_c$ est l'optimum local trouvé
fin
retourner $\left\{\vec{S}_c, U_n, R\right\}$

PROCÉDURE 2.5 – Procédure effectuant une itération de la recherche locale d'un agent.

2.2.2.4 La stratégie de maintien de la diversité

Le maintien de la diversité est déjà inhérent à MADO, car la recherche locale d'un agent est réinitialisée, à chaque fois qu'elle est terminée, dans une zone inexplorée de l'espace de recherche. Cependant, pour éviter que plusieurs agents n'explorent une même zone de l'espace de recherche, au risque de converger vers le même optimum local, nous attribuons, à chaque agent, une zone de recherche exclusive. A cet effet, chaque agent dispose d'un rayon d'exclusion, noté r_e. Ainsi, un agent détecte si d'autres agents sont situés à des distances, entre sa position (solution) courante et celles de ces autres agents, inférieures à r_e. Si tel est le cas, seul celui ayant la meilleure solution courante continue sa recherche à cet endroit. Les autres agents en conflit dans cette zone doivent redémarrer leur recherche locale ailleurs dans l'espace de recherche. L'agent ayant détecté d'autres agents à proximité peut lui-même devoir redémarrer sa recherche, s'il n'a pas la meilleure solution courante. Cette stratégie correspond à la condition indiquée à la figure 2.3 par « Trop proche d'autres agents ? ».

De plus, lorsque le pas d'un agent devient inférieur à un certain seuil, noté r_l, la procédure 2.6 est exécutée. Cette procédure sert à éviter de converger vers un optimum local déjà présent dans l'archive des optima locaux trouvés par les agents. Elle permet ainsi d'économiser des

évaluations de la fonction objectif. Le seuil r_l est un paramètre de MADO, utilisé également par la suite dans les paragraphes 2.2.2.6 et 2.2.2.7. Cette procédure correspond à la condition indiquée à la figure 2.3 par « Trop proche d'un optimum archivé ? ».

échappementDesOptimaConnus

1 **Entrées :** $\vec{S_c}, A_m, r_l, r_e$
2 **Variables locales :** $\vec{O_i}$
3 **pour tout** $\vec{O_i} \in A_m$ **faire**
4 **si** $\left\|\vec{S_c} - \vec{O_i}\right\| \leq \sqrt{r_l \times r_e}$ et $\vec{O_i}$ est strictement meilleur que $\vec{S_c}$ **alors**
5 redémarrer la recherche locale de l'agent
6 terminer la procédure
7 **fin**
8 **fin**

PROCÉDURE 2.6 – Procédure permettant d'éviter à un agent de converger vers un optimum local déjà archivé. La solution courante de l'agent est notée $\vec{S_c}$, l'archive des optima locaux est notée A_m, les pas initiaux des agents de suivi et d'exploration sont notés r_l et r_e, respectivement.

Dans cette procédure, nous voyons (ligne 4) que si un optimum local archivé, noté $\vec{O_i}$, est strictement meilleur que la solution courante $\vec{S_c}$ de l'agent, et si $\vec{O_i}$ se trouve à une distance de $\vec{S_c}$ inférieure ou égale à la moyenne géométrique de r_l et de r_e, alors la recherche locale de l'agent est redémarrée (ligne 5). La moyenne géométrique de r_l et de r_e sert ainsi de seuil pour déterminer les optima locaux de l'archive qui sont « proches » de $\vec{S_c}$. Ce seuil est également utilisé par la suite, au paragraphe 2.2.2.7.

Ce redémarrage de la recherche locale d'un agent, qu'il soit dû au rayon d'exclusion ou à la procédure 2.6, se fait de la même façon que pour un agent ayant terminé sa recherche locale : dans une zone inédite de l'espace de recherche, indiquée par le coordinateur. Nous allons voir, au paragraphe suivant, comment le coordinateur calcule cette nouvelle position initiale.

2.2.2.5 Le repositionnement des agents

Afin de réinitialiser les agents dans des zones inexplorées de l'espace de recherche, donc potentiellement prometteuses, la nouvelle position (solution) initiale de la recherche locale d'un agent est calculée comme suit : une position est tirée aléatoirement selon une distribution uniforme dans l'espace de recherche. La distance entre cette position et tout autre agent doit être supérieure ou égale à r_e. La distance entre cette position et tout optimum présent dans l'archive des optima locaux trouvés doit également être supérieure ou égale à r_e. Si ces conditions ne sont

pas satisfaites, une nouvelle position est tirée aléatoirement. Si, après cinq tentatives (nous avons déterminé empiriquement que cinq tentatives étaient suffisantes pour obtenir de bons résultats), la position tirée ne satisfait toujours pas ces conditions, alors l'espace de recherche est considéré comme saturé, et l'agent est *détruit*. Dans le cas contraire, nous obtenons une position initiale, pour cet agent, qui est éloignée de celles des autres agents et des optima locaux trouvés.

Le pas de l'agent est, quant à lui, initialisé à r_e, tout comme celui des agents créés à l'initialisation de MADO.

2.2.2.6 La détection de changements et le suivi des optima locaux

Une autre fonction du coordinateur est de détecter tout éventuel changement dans la fonction objectif. Cette détection est effectuée lorsque tous les agents ont terminé l'exécution d'une itération de leurs recherches locales. Cela correspond au moment où tous les agents se trouvent dans l'état de SYNCHRONISATION décrit au paragraphe 2.2.1.3. Le coordinateur détecte un changement dans la fonction objectif en réévaluant le meilleur optimum de l'archive des optima locaux. Si l'archive est vide, alors c'est la solution courante d'un agent, choisi aléatoirement, qui est réévaluée. Si la valeur de cette solution a changé, cela signifie qu'un changement s'est produit dans la fonction objectif. Dans ce cas, les opérations suivantes sont effectuées : les solutions courantes des agents, ainsi que les optima locaux archivés, sont réévalués. Ensuite, un nouvel agent est créé pour chaque optimum local de l'archive. Ces nouveaux agents ont pour but de suivre les déplacements éventuels, dans l'espace de recherche, des optima archivés. Leurs solutions courantes $\vec{S_c}$ sont donc initialisées avec les optima de l'archive. Comme leur rôle n'est pas d'explorer largement l'espace de recherche, mais essentiellement de suivre les optima de l'archive, leurs pas R ne sont pas initialisés à r_e, mais à une valeur plus petite, notée r_l. Cela évite ainsi aux agents de suivi de s'écarter de l'optimum local à suivre pour aller explorer plus largement l'espace de recherche. Une fois ces agents créés, l'archive des optima locaux est vidée.

A chaque détection d'un changement, des agents de suivi sont ainsi créés. Pour éviter que le nombre d'agents ne devienne trop grand, nous proposons de prendre les mesures suivantes : si le nombre d'agents existants est supérieur à n_a et si un agent doit être repositionné (voir paragraphe 2.2.2.5), alors le coordinateur détruit cet agent. Ainsi, tant que le nombre d'agents est supérieur à n_a, tout agent est détruit dès la fin de sa recherche locale.

Enfin, le pas des agents de suivi, r_l, est un paramètre de MADO. Il est également utilisé dans la gestion de l'archive, comme expliqué au paragraphe suivant.

2.2.2.7 La gestion de l'archive des optima locaux

Le module de gestion de la mémoire maintient l'archive, notée A_m, des optima locaux trouvés par les agents. Pour que la consommation de mémoire et l'effort de calcul restent raisonnables, il est nécessaire de borner cette archive. Nous avons donc limité la capacité de l'archive à n_m solutions.

Lorsqu'un nouvel optimum local, noté $\vec{O}_c$, est trouvé par un agent, le module de gestion de la mémoire décide s'il doit être ajouté ou non à l'archive. Cela se fait en respectant les règles suivantes :

- Si l'archive n'est pas pleine, alors $\vec{O}_c$ y est ajouté.
- Si l'archive est pleine, alors $\vec{O}_c$ n'est sauvegardé que s'il est meilleur que le plus mauvais optimum de l'archive, ou si sa valeur est au moins égale à celle du plus mauvais optimum. Dans ce cas, $\vec{O}_c$ remplace le plus mauvais optimum de l'archive.
- Si une ou plusieurs solutions de l'archive sont « trop proches » de $\vec{O}_c$ (une solution archivée est considérée « trop proche » de $\vec{O}_c$ lorsqu'elle se trouve à une distance de $\vec{O}_c$ inférieure ou égale à la moyenne géométrique de r_l et de r_e), alors toutes ces solutions, proches les unes des autres, sont considérées comme étant « dominées » par la meilleure d'entre elles. Dans ce cas, cet ensemble de solutions est remplacé par la meilleure d'entre elles. Ce remplacement est effectué par la procédure 2.7.

retraitDesOptimaDominés

Entrées : $\vec{O}_c, A_m, r_l, r_e$
Variables locales : $A_{sub}, \vec{O}_{best}, \vec{O}_i$
$A_{sub} \leftarrow \varnothing$
$\vec{O}_{best} \leftarrow \vec{O}_c$
pour tout $\vec{O}_i \in A_m$ **faire**
  si $\left\|\vec{O}_c - \vec{O}_i\right\| \leq \sqrt{r_l \times r_e}$ **alors**
    $A_{sub} \leftarrow A_{sub} \cup \{\vec{O}_i\}$
    si $\vec{O}_i$ est strictement meilleur que $\vec{O}_{best}$ **alors**
      $\vec{O}_{best} \leftarrow \vec{O}_i$
    fin
  fin
fin
$A_m \leftarrow A_m - A_{sub}$
$A_m \leftarrow A_m \cup \{\vec{O}_{best}\}$
retourner A_m

PROCÉDURE 2.7 – Remplacement des optima locaux dominés de l'archive.

2.2.3 Résultats et analyse

Pour tester MADO, nous utilisons le *Moving Peaks Benchmark* (MPB) et le *Generalized Dynamic Benchmark Generator* (GDBG), tels que décrits au paragraphe 1.4.2. Nous commençons par une discussion sur le paramétrage de MADO et par la présentation des valeurs des paramètres utilisées pour ces deux jeux de tests. Ensuite, nous effectuons une analyse de la sensibilité des paramètres de MADO, suivie d'une analyse expérimentale des stratégies utilisées dans l'algorithme. Enfin, nous réalisons une étude de la convergence de l'algorithme, ainsi qu'une comparaison de ses performances avec celles des autres algorithmes de la littérature.

2.2.3.1 Paramétrage

Le tableau 2.1 récapitule la liste des paramètres de MADO et leur réglage pour MPB et GDBG.

Nous avons fixé le nombre d'agents $n_a = 4$ car l'algorithme doit converger rapidement. En effet, avoir trop d'agents qui explorent l'espace de recherche en même temps ralentit la convergence globale de MADO. La précision δ_p et le pas r_l des agents « de suivi » (créés pour suivre les optima archivés, lorsqu'un changement dans la fonction objectif est détecté) doivent être adaptés à la sévérité et à la fréquence des changements de la fonction objectif.

En effet, avec une valeur trop faible de δ_p (c'est-à-dire, inférieure à sa valeur optimale, donnée dans le tableau 2.1 pour MPB et GDBG), l'agent consommera beaucoup d'évaluations pour intensifier sa solution courante, au détriment de la diversification. L'espace de recherche risque alors de ne pas être suffisamment exploré entre deux changements consécutifs de la fonction objectif, et le nombre d'optima locaux trouvés risque d'être trop restreint. De ce fait, plus le nombre d'évaluations entre deux changements dans la fonction objectif est faible, et plus il faut accroître la valeur de δ_p. Cela signifie qu'il existe un compromis entre la précision des optima locaux trouvés (intensification) et l'exploration de l'espace de recherche (diversification), de manière à favoriser la diversification, lorsque les changements dans la fonction objectif sont fréquents. Il en va de même pour n_s, qui doit rester petit (significativement inférieur à la dimension du problème), lorsque la fonction objectif est sujette à des changements fréquents. Ce choix permet de garantir une convergence plus rapide des agents, et par suite une exploration plus efficace de l'espace de recherche. Ce paramètre dépend également de la dimension du problème.

Le pas initial d'un agent de suivi, r_l, doit être inférieur au rayon de la zone d'attraction de l'optimum local qu'il doit suivre, afin que l'agent ne s'écarte pas de cet optimum local pour

Nom	**Type**	**Intervalle**	**MPB**	**GDBG**	**Courte description**
n_a	entier	$[1;\ 10]$	4	4	nombre initial d'agents
n_m	entier	$[0;\ +\infty[$	10	10	capacité de l'archive des optima locaux
n_s	entier	$[1;\ 2\,d]$	$round(0{,}28\,d+2)$	$round(0{,}5\,d+2)$	nombre de solutions candidates du voisinage d'un agent
c_u	réel	$[0;\ 1]$	0,5	0,5	coefficient du produit scalaire cumulé
c_r	réel	$]0;\ 1[$	0,8	0,8	coefficient d'adaptation du pas d'un agent
r_e	réel	$]0;\ 1]$	0,2	0,2	rayon d'exclusion des agents, et pas initial d'un agent « d'exploration »
r_l	réel	$]0;\ r_e[$	7E−3	7E−3	pas initial d'un agent « de suivi » (créé pour suivre un optimum de l'archive)
δ_t	entier	$[0;\ 10]$	2	2	nombre de déplacements qu'un agent peut faire sans améliorer sa solution courante de plus de δ_p
δ_p	réel	$[0;\ +\infty[$	1E−3	6E−5	la précision du critère de stagnation de la recherche locale d'un agent

TABLEAU 2.1 – Valeurs des paramètres de MADO pour MPB et GDBG. La fonction *round* arrondit un réel, donné en argument, à l'entier le plus proche. La dimension du problème est notée d.

aller explorer plus largement l'espace de recherche. Le rayon d'exclusion r_e doit correspondre au rayon de la zone d'attration d'un optimum local. Par conséquent, nous avons posé : $r_l < r_e$. Néanmoins, si la valeur de r_l est très inférieure à la distance dont s'est déplacé l'optimum que doit suivre l'agent, alors la convergence de l'agent vers cet optimum sera ralentie. Le pas r_l doit donc être également adapté en fonction de la sévérité des changements de la fonction objectif.

La valeur de n_m doit permettre à une partie des optima locaux d'être archivés. Le coefficient d'adaptation du pas, c_r, peut être laissé à sa valeur par défaut : $c_r = 0{,}8$. Cependant, des valeurs supérieures à 0,8 peuvent améliorer les performances de MADO pour des problèmes mal conditionnés (pour lesquels une « petite » perturbation d'une solution entraîne une « grande » perturbation de la valeur de la fonction objectif de cette solution). Les autres paramètres de MADO, c_u et δ_t, peuvent, quant à eux, être laissés à leurs valeurs par défaut : $c_u = 0{,}5$ et $\delta_t = 2$. Dans ce qui suit, nous allons voir l'influence de chaque paramètre sur les performances de MADO, et nous constaterons que ces deux paramètres n'influencent pas significativement les performances de MADO.

2.2.3.2 Analyse de la sensibilité des paramètres

La sensibilité de MADO est étudiée en faisant varier chaque paramètre de l'algorithme, en laissant les autres paramètres fixés à leurs valeurs par défaut (celles de la colonne MPB du tableau 2.1, voir [Lepagnot et al., 2009]). Chaque paramètre de MADO est ainsi étudié sur MPB (scénario 2, voir paragraphe 1.4.2.1) et sur la fonction Rosenbrock (une fonction de test classique, parmi les plus difficiles à résoudre). Cette fonction est définie dans l'équation (2.2.8), où $\vec{x}$ est une solution à évaluer, x_i est sa $i^{\text{ième}}$ coordonnée et d est la dimension de l'espace de recherche. Ici, $d = 5$ et l'espace de recherche est $[-10;\ 10]^5$. Notre choix s'est porté sur cette fonction, car elle constitue un problème de minimisation statique, mal conditionné, qui admet un seul optimum global, de valeur 0.

$$f(\vec{x}) = \sum_{i=1}^{d-1} \left[(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2\right] \tag{2.2.8}$$

L'algorithme est arrêté lorsque 5E+5 évaluations sont effectuées pour MPB, et lorsque 4E+4 évaluations sont effectuées pour la fonction Rosenbrock. Les performances de MADO sont mesurées en utilisant l'*offline error* (voir paragraphe 1.4.2.1) pour MPB et la valeur de la meilleure solution trouvée pour la fonction Rosenbrock. Les résultats sont moyennés sur 100 exécutions, et présentés aux figures 2.6, 2.7 et 2.8.

Comme nous pouvons le voir sur ces figures, faire varier la valeur de n_m ne perturbe pas

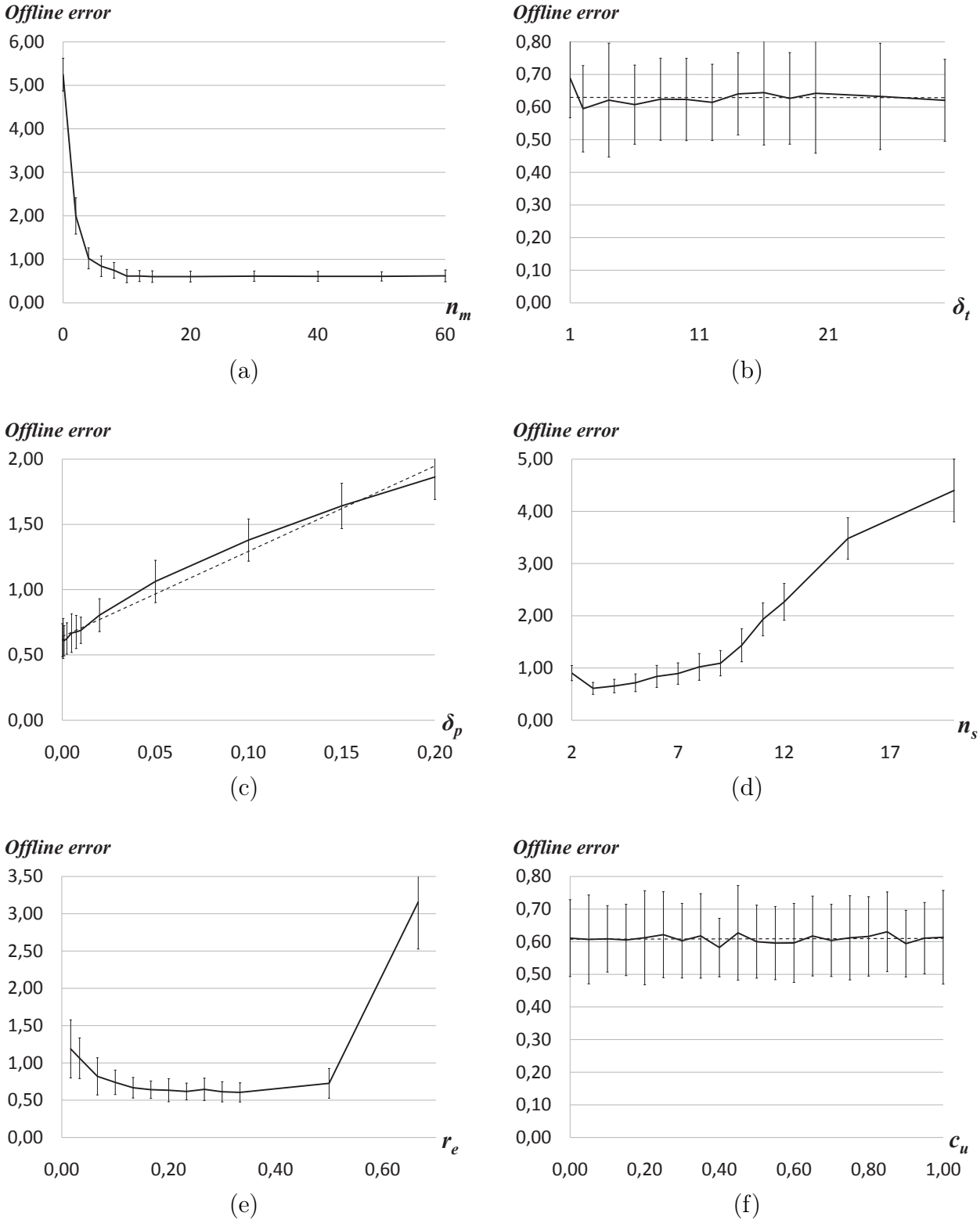


FIGURE 2.6 – L'évolution de l'*offline error*, sur MPB, pour différentes valeurs de n_m, δ_t, δ_p, n_s, r_e et c_u, est donnée en (a), (b), (c), (d), (e) et (f), respectivement.

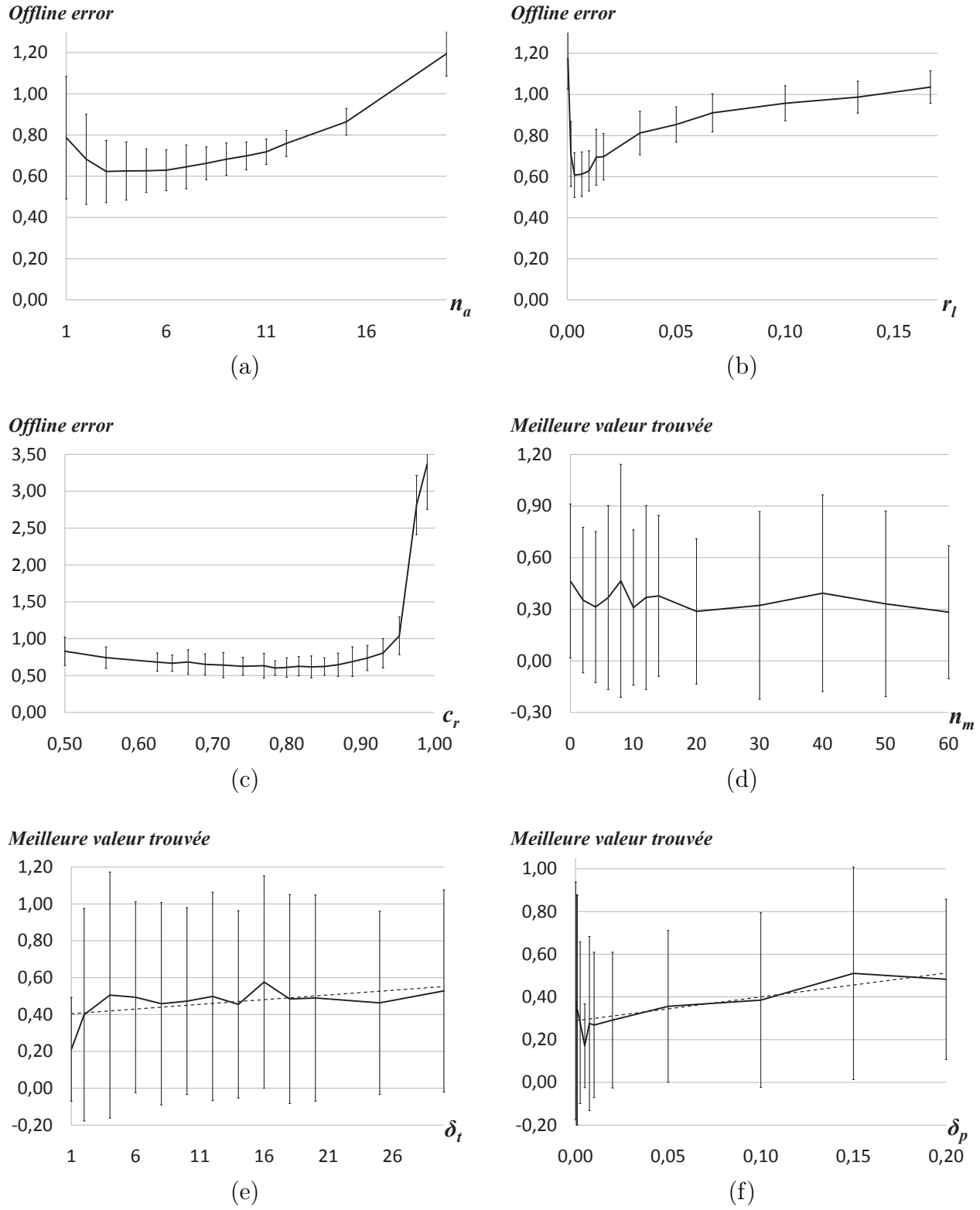


FIGURE 2.7 – L'évolution de l'*offline error*, sur MPB, pour différentes valeurs de n_a, r_l et c_r, est donnée en (a), (b) et (c), respectivement. L'évolution de la valeur de la meilleure solution trouvée, sur la fonction Rosenbrock, pour différentes valeurs de n_m, δ_t et δ_p, est donnée en (d), (e) et (f), respectivement.

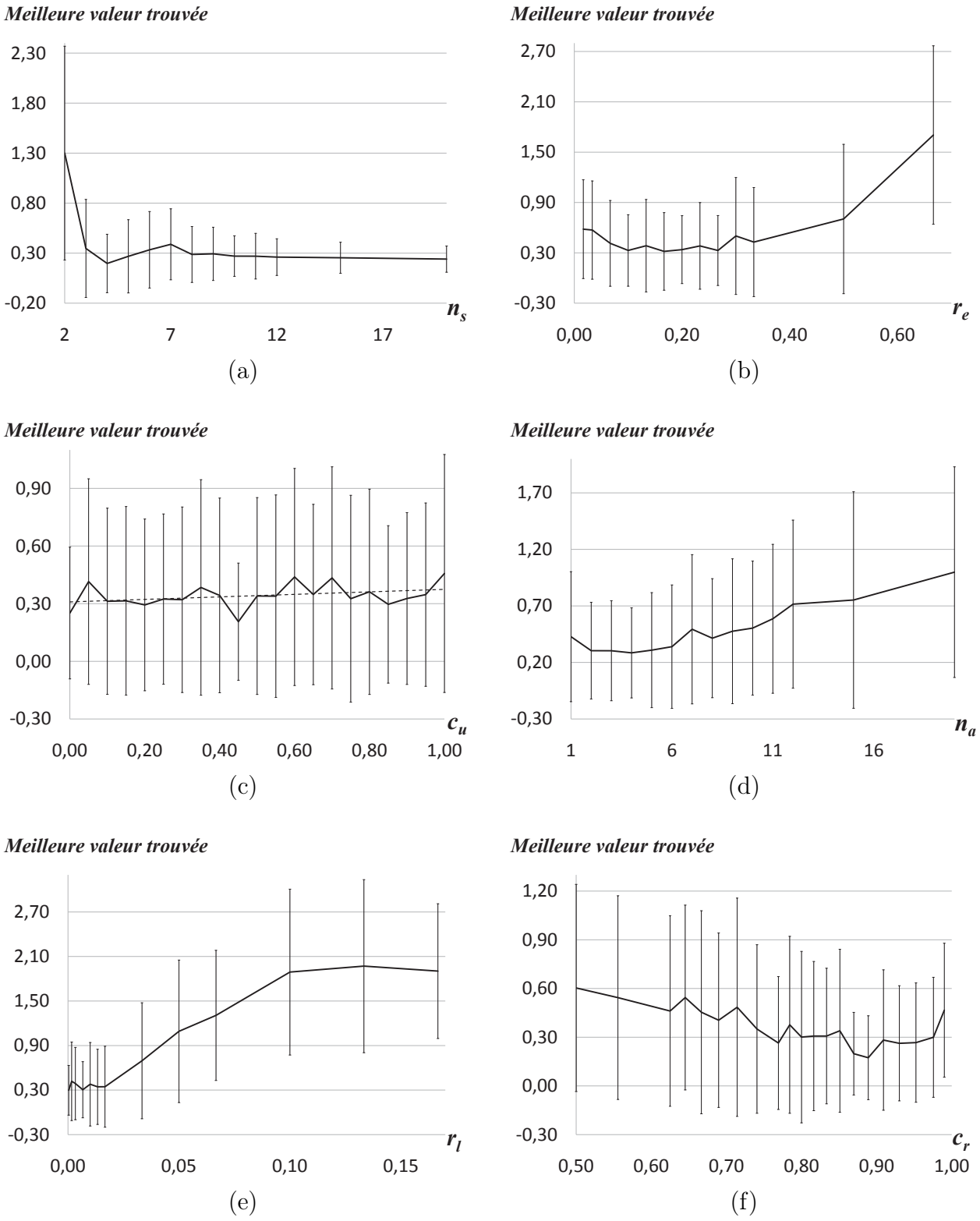


FIGURE 2.8 – L'évolution de la valeur de la meilleure solution trouvée, sur la fonction Rosenbrock, pour différentes valeurs de n_s, r_e, c_u, n_a, r_l et c_r, est donnée en (a), (b), (c), (d), (e) et (f), respectivement.

significativement les performances de MADO sur la fonction Rosenbrock. Néanmoins, sur MPB, les performances de MADO s'améliorent en augmentant la valeur de n_m. Nous pouvons ainsi en conclure que n_m n'a d'importance que pour des problèmes dynamiques, et qu'il doit être assez grand pour qu'un nombre suffisant d'optima locaux puissent être archivés et suivis. En l'occurrence, une valeur supérieure ou égale à 10 est bien adaptée à MPB.

La meilleure valeur du paramètre n_s, à la fois pour MPB et pour la fonction Rosenbrock, est de 3. Néanmoins, pour des problèmes statiques ou en plus grande dimension, une valeur plus grande de n_s peut améliorer les performances.

Le comportement de n_a, δ_t et c_u est similaire pour MPB et la fonction Rosenbrock. Il apparaît que les valeurs produisant les meilleures performances, sur ces deux problèmes, sont $n_a = 4$; $\delta_t = 2$ et $c_u = 0{,}5$. Toutefois, nous pouvons remarquer que faire varier la valeur de δ_t et de c_u ne perturbe pas significativement les performances de MADO, sur les deux problèmes. Ces deux paramètres peuvent ainsi être laissés à leurs valeurs par défaut.

La meilleure valeur du paramètre r_l, pour les deux problèmes, est de 7E−3. Néanmoins, pour des problèmes statiques, de bons résultats peuvent être obtenus en utilisant des valeurs de r_l inférieures à 7E−3.

Le comportement de r_e, δ_p et c_r est similaire pour les deux problèmes, mais les valeurs optimales de ces paramètres sont dépendantes du problème. Il est donc nécessaire d'adapter les valeurs de ces trois paramètres au problème à résoudre.

2.2.3.3 Analyse expérimentale des stratégies utilisées

Afin de justifier l'usage de chaque composant de MADO, nous allons évaluer diverses variantes de MADO, chacune ayant été simplifiée en lui retirant une stratégie spécifique employée dans l'algorithme original de MADO. Cette évaluation est effectuée sur MPB (scénario 2), en moyennant les résultats obtenus sur 100 exécutions de l'algorithme. Le nombre maximal d'évaluations est de 5E+5, ce qui correspond à 100 changements dans la fonction objectif à chaque exécution de l'algorithme. Les valeurs d'*offline error* et d'écart-type ainsi obtenues, pour chaque variante de MADO, sont données dans le tableau 2.2. Dans ce tableau, les variantes sont ordonnées de la meilleure à la moins bonne.

Pour ces variantes simplifiées, dans $\text{MADO}_{\text{noDom}}$, la procédure 2.7 n'est pas utilisée. Les optima locaux « dominés » par un autre ne sont donc pas retirés de l'archive A_m. Dans $\text{MADO}_{\text{no}\,S}$, l'ensemble S n'est pas utilisé et les solutions candidates sont tirées aléatoirement, selon une distribution uniforme, sur l'hypersphère de centre $\vec{S_c}$ et de rayon R, à chaque fois qu'un agent doit évaluer ses solutions voisines. Dans $\text{MADO}_{\text{randRel}}$, la stratégie de repositionnement des agents dans des zones inexplorées de l'espace de recherche (décrite au paragraphe 2.2.2.5) n'est pas uti-

Variante	***Offline error***	**Description succincte**
MADO	$0{,}60 \pm 0{,}11$	l'algorithme original
$\text{MADO}_{\text{noDom}}$	$0{,}61 \pm 0{,}11$	sans retirer les optima locaux « dominés » de l'archive
$\text{MADO}_{\text{no}\,S}$	$0{,}63 \pm 0{,}12$	sans utiliser l'heuristique de répulsion électrostatique pour générer les solutions candidates des agents
$\text{MADO}_{\text{randRel}}$	$0{,}64 \pm 0{,}16$	en repositionnant aléatoirement, selon une distribution uniforme, les agents
$\text{MADO}_{\text{noCDPA}}$	$0{,}64 \pm 0{,}12$	sans utiliser le produit scalaire cumulé pour adapter R
$\text{MADO}_{\text{randInit}}$	$0{,}65 \pm 0{,}15$	sans utiliser l'heuristique de répulsion électrostatique pour générer l'ensemble initial d'agents
$\text{MADO}_{\text{noEsc}}$	$0{,}69 \pm 0{,}17$	sans utiliser la procédure 2.6
$\text{MADO}_{\text{noExcl}}$	$0{,}90 \pm 0{,}25$	sans utiliser de rayon d'exclusion
$\text{MADO}_{\text{no}\,r_l}$	$1{,}06 \pm 0{,}07$	en utilisant r_e, au lieu de r_l, comme pas initial des agents de suivi
$\text{MADO}_{\text{noTrack}}$	$3{,}65 \pm 0{,}24$	sans créer d'agents de suivi pour suivre les optima locaux archivés
$\text{MADO}_{\text{noArch}}$	$5{,}30 \pm 0{,}35$	sans archivage des optima locaux

TABLEAU 2.2 – *Offline error* sur MPB, pour chaque variante simplifiée de MADO.

lisée. La recherche locale d'un agent est alors réinitialisée aléatoirement, selon une distribution uniforme, dans l'espace de recherche. Dans $\text{MADO}_{\text{noCDPA}}$, le produit scalaire cumulé n'est pas utilisé pour adapter R. L'adaptation du pas d'un agent consiste alors uniquement à réduire R, lorsque l'agent ne trouve aucune solution voisine qui soit meilleure que $\vec{S}_c$. Dans $\text{MADO}_{\text{randInit}}$, l'ensemble initial d'agents est créé aléatoirement, selon une distribution uniforme, dans l'espace de recherche. L'heuristique de répulsion électrostatique n'est pas utilisée, et il est alors probable que des agents soient très proches les uns des autres, à l'initialisation de MADO. Dans $\text{MADO}_{\text{noEsc}}$, la procédure 2.6 n'est pas utilisée, et la convergence d'un agent vers un optimum local déjà archivé n'est alors pas interrompue. Dans $\text{MADO}_{\text{noExcl}}$, la détection d'agents situés dans la rayon d'exclusion r_e d'un agent n'est pas effectuée. De ce fait, plusieurs agents peuvent explorer une même zone de l'espace de recherche, sans avoir à être repositionnés. Dans $\text{MADO}_{\text{no}\,r_l}$, le pas initial de tout agent, qu'il soit créé pour explorer l'espace de recherche ou pour suivre un optimum local de l'archive, est égal à r_e. Dans $\text{MADO}_{\text{noTrack}}$, aucun agent n'est créé pour suivre les optima locaux archivés, lorsqu'un changement est détecté dans la fonction

objectif. Enfin, dans $\text{MADO}_{\text{noArch}}$, l'archivage des optima locaux n'est pas utilisé.

D'après le tableau 2.2, nous remarquons que les variantes $\text{MADO}_{\text{noTrack}}$ et $\text{MADO}_{\text{noArch}}$ obtiennent, de loin, les plus mauvais résultats. Nous pouvons ainsi conclure que les composants les plus critiques de MADO, en termes de performance, sont l'archivage des optima locaux et leur suivi par des agents dédiés.

D'après les résultats obtenus par $\text{MADO}_{\text{noExcl}}$ et $\text{MADO}_{\text{no}\,r_l}$, nous pouvons également conclure que l'utilisation d'une zone de recherche exclusive pour chaque agent, ainsi que l'utilisation d'un pas initial r_l propre aux agents de suivi, sont des stratégies importantes en vue d'obtenir de bonnes performances.

Enfin, nous avons appliqué le test statistique de Kruskal-Wallis [Kruskal & Wallis, 1952] sur les résultats obtenus par les sept premières variantes de MADO dans ce tableau. Ce test indique qu'il existe une différence significative entre les performances d'au moins deux variantes parmi les sept, avec un niveau de confiance de 99%. Nous utilisons ensuite le test de Tukey-Kramer [Tukey, 1994; Kramer, 1957] pour déterminer, parmi ces sept variantes, celles qui diffèrent des autres, en termes d'*offline error*. Ce test indique qu'il existe une différence significative entre l'algorithme MADO original et la variante $\text{MADO}_{\text{noEsc}}$, avec un niveau de confiance de 99%. Il est donc utile d'interrompre la convergence d'un agent vers un optimum local déjà archivé. En revanche, ce test ne permet pas de mettre en évidence une différence significative entre les performances de l'algorithme original et celles des cinq autres variantes testées. L'impact des cinq composants de MADO correspondants n'est donc pas significatif pour MPB. Nous verrons au chapitre suivant comment en tenir compte, afin de proposer un algorithme simplifié et encore plus performant.

2.2.3.4 Analyse de convergence

La convergence de MADO est illustrée à la figure 2.9, en utilisant MPB (scénario 2) en dimension 2. Sur cette figure, l'axe y correspond aux 10 premiers *time spans*[1], l'axe x correspond aux 1000 premières évaluations de chaque *time span* avec une granularité de 50 évaluations, et l'axe z correspond à la valeur de l'erreur relative $\frac{f_y^* - f_{yx}^*}{f_y^*}$, en utilisant les notations de l'équation 1.4.2. Par commodité, nous utilisons une échelle logarithmique pour l'axe z.

Nous constatons sur cette figure que la convergence de MADO est très rapide. Les plus grandes valeurs d'erreur sont obtenues au premier *time span*, car MADO n'a pas encore détecté et archivé les optima locaux. Une fois ces derniers archivés, l'algorithme peut suivre ces optima locaux, sans avoir besoin de les redétecter. La convergence de MADO s'en trouve alors améliorée.

1. Pour rappel, un *time span* est une plage d'évaluations durant laquelle la fonction objectif ne change pas.

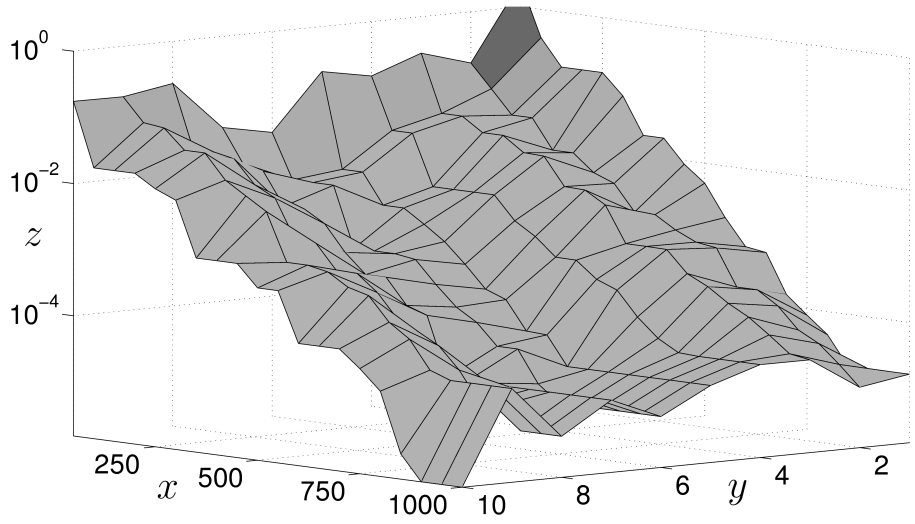


FIGURE 2.9 – Erreur relative en utilisant MPB. L'axe x correspond aux 1000 premières évaluations de chaque *time span*. L'axe y correspond aux 10 premiers *time spans*. L'axe z correspond à la valeur de l'erreur relative, en utilisant une échelle logarithmique.

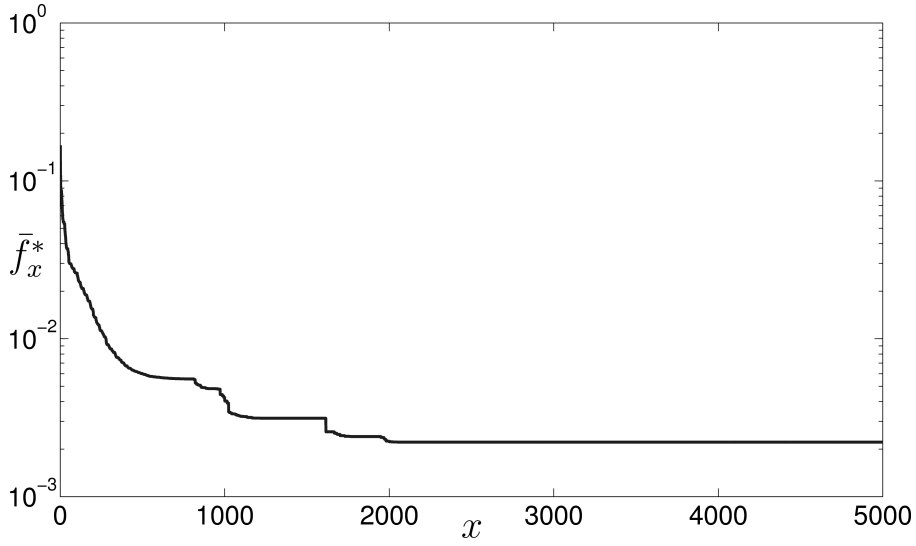


FIGURE 2.10 – Erreur relative moyenne en utilisant MPB.

L'évolution moyenne de l'erreur relative sur tous les *time spans* est illustrée à la figure 2.10. Sur cette figure, x correspond aux 5000 évaluations d'un *time span*, et $\bar{f}_x^*$ est la valeur moyenne de f_{jx}^* pour $j = 1,...,N_c$ (en utilisant les notations de l'équation 1.4.2). Pour plus de clarté, nous utilisons une échelle logarithmique pour l'axe de f_{jx}^*.

Nous constatons sur cette figure qu'il suffit de 3 évaluations pour que MADO obtienne une erreur relative moyenne inférieure à 10^{-1}, et qu'il lui en suffit 278 pour obtenir une erreur relative moyenne inférieure à 10^{-2}.

2.2.3.5 Comparaison avec d'autres algorithmes

Nous allons maintenant effectuer une comparaison de MADO avec les autres algorithmes d'optimisation dynamique les plus compétitifs de la littérature, en utilisant MPB avec le scénario 2 (le plus répandu dans la littérature) et GDBG avec le réglage de la compétition de CEC'09. Nous avons recueilli le plus d'algorithmes possibles pouvant être comparés sur ces jeux de tests. Les résultats de ces différents algorithmes ont été relevés dans les articles correspondants. La comparaison sur MPB est effectuée en utilisant l'*offline error* moyennée sur 50 exécutions de chaque algorithme. Les valeurs d'*offline error* et d'écart-type de chaque algorithme sont présentées dans le tableau 2.3. Dans ce tableau, les algorithmes sont classés du meilleur au moins bon, en termes d'*offline error*. La comparaison sur GDBG, en utilisant le score propre à ce jeu de tests (noté *op* et mesurant la performance d'un algorithme sur une échelle comprise entre 0 et 100), est présentée à la figure 2.11.

Algorithme	***Offline error***
Moser & Chiong, 2010	$0{,}25 \pm 0{,}08$
Novoa et al., 2009	$0{,}40 \pm 0{,}04$
MADO	$0{,}59 \pm 0{,}10$
Moser & Hendtlass, 2007	$0{,}66 \pm 0{,}20$
Yang & Li, 2010	$1{,}06 \pm 0{,}24$
Liu et al., 2010	$1{,}31 \pm 0{,}06$
Lung & Dumitrescu, 2007	$1{,}38 \pm 0{,}02$
Bird & Li, 2007	$1{,}50 \pm 0{,}08$
Lung & Dumitrescu, 2008	$1{,}53 \pm 0{,}01$
Blackwell & Branke, 2006	$1{,}72 \pm 0{,}06$
Mendes & Mohais, 2005	$1{,}75 \pm 0{,}03$
Li et al., 2006	$1{,}93 \pm 0{,}06$
Blackwell & Branke, 2004	$2{,}16 \pm 0{,}06$
Parrott & Li, 2006	$2{,}51 \pm 0{,}09$
Du & Li, 2008	$4{,}02 \pm 0{,}56$

TABLEAU 2.3 – Classement des algorithmes en compétition sur MPB.

Nous constatons que MADO est classé troisième sur MPB, et premier sur GDBG. Les résultats obtenus montrent ainsi que MADO se situe parmi les meilleurs algorithmes sur les deux principaux jeux de tests en optimisation dynamique. Les bonnes performances de MADO s'expliquent par sa convergence rapide vers les optima locaux, rendue possible par l'utilisation de recherches locales. Les algorithmes évolutionnaires, ou basés sur l'optimisation par essaim particulaire, nécessitent souvent plus d'évaluations pour atteindre un optimum local. Ainsi, les performances des algorithmes classés en première et deuxième positions s'expliquent également par l'utilisation d'une recherche locale et de techniques destinées à accélérer la convergence vers

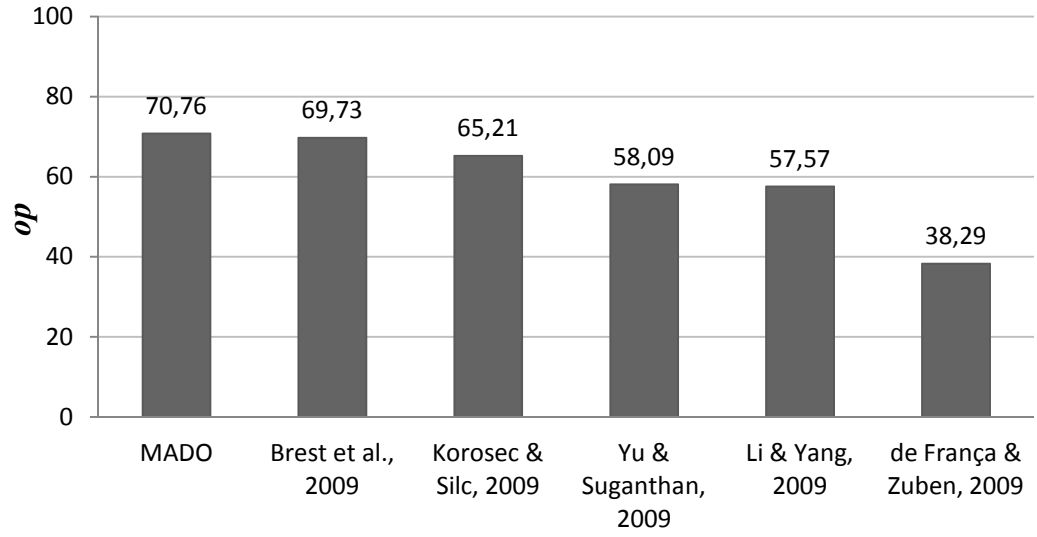


FIGURE 2.11 – Classement des algorithmes en compétition sur GDBG.

les optima locaux.

Cependant, l'utilisation d'un voisinage isotrope (une hypersphère) dans les recherches locales des agents ne permet pas à MADO d'être adapté aux problèmes mal conditionnés. Afin d'améliorer les performances de MADO, nous proposons d'y intégrer une technique lui permettant de s'adapter à ce type de problèmes. Cette variante de MADO est présentée au paragraphe suivant.

2.3 Adaptation aux problèmes mal conditionnés (CMADO)

Le *conditionnement* mesure la dépendance de la solution d'un problème numérique par rapport aux données du problème, ceci afin de contrôler la validité d'une solution calculée par rapport à ces données. Il s'agit le plus souvent d'une quantité numérique, parfois appelée *nombre de conditionnement*. De façon plus générale, on peut dire que le nombre de conditionnement associé à un problème est une mesure de la difficulté de calcul numérique du problème. Un problème possédant un nombre de conditionnement faible est dit *bien conditionné* et un problème possédant un nombre de conditionnement élevé est dit *mal conditionné*.

Un problème mal conditionné est ainsi un problème pour lequel une petite perturbation d'une solution entraîne une grande perturbation de la valeur de la fonction objectif de cette solution. Concrètement, cela peut se traduire par une « compression » des lignes de valeurs égales de la fonction objectif, telle qu'illustrée à la figure 2.12.

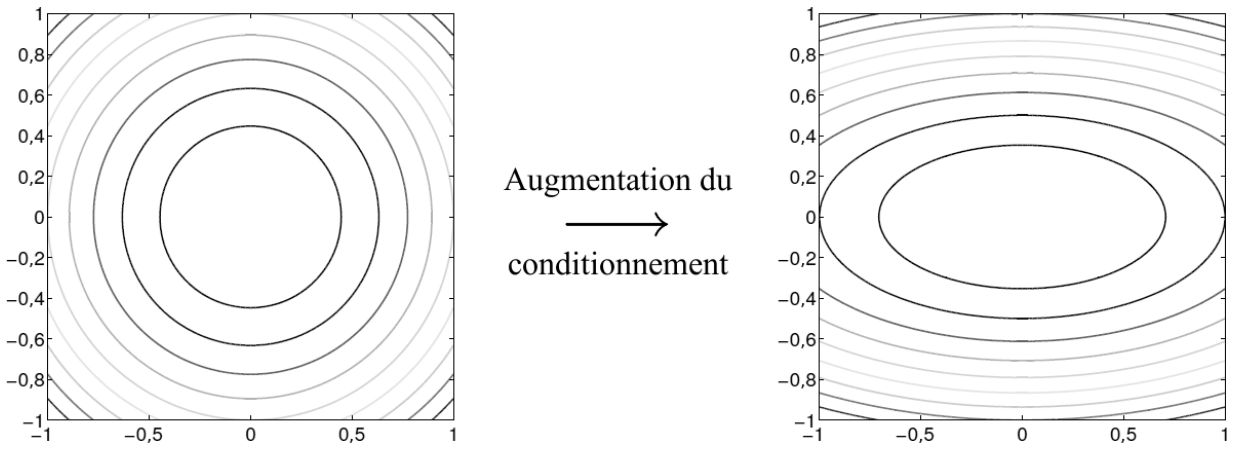


FIGURE 2.12 – Illustration du conditionnement d'un problème.

Pour qu'un algorithme d'optimisation soit performant sur ce type de problèmes, il peut être nécessaire d'utiliser des techniques particulières. Il existe notamment une technique, proposée dans [Hansen & Ostermeier, 2001], permettant à l'algorithme de s'adapter à de tels problèmes. Cette technique est basée sur l'utilisation d'une matrice de covariance, adaptée tout au long de l'exécution de l'algorithme.

Dans ce paragraphe, nous présentons une amélioration de MADO lui permettant de résoudre efficacement les problèmes mal conditionnés. Cette technique permet d'adapter le voisinage de chaque agent au paysage local de la fonction objectif, au moyen d'une matrice de covariance. Cette adaptation du voisinage permet ainsi à un agent de converger efficacement vers un optimum local d'un problème mal conditionné, comme illustré à la figure 2.13.

Sur cette figure, les lignes de valeurs égales de la fonction objectif d'un problème mal conditionné sont représentées. Sans adaptation du voisinage d'un agent, il est possible que l'agent ait du mal à sortir de la zone indiquée en noir. Depuis cette zone, l'agent doit en effet longer une fine vallée menant à l'optimum, indiquée par des flèches. L'adaptation permet ainsi de comprimer le voisinage de l'agent (adaptation de l'hypersphère, indiquée en gris, sur laquelle sont tirées les solutions voisines de la solution courante de l'agent), pour lui permettre d'atteindre efficacement l'optimum.

2.3.1 La technique d'adaptation proposée

La version améliorée de MADO que nous proposons est appelée CMADO, pour *Covariance matrix adaptation MultiAgent Dynamic Optimization*. La technique proposée consiste ainsi à adapter une matrice de covariance, notée C, à chaque déplacement d'un agent, en même temps que le produit scalaire cumulé. Chaque agent est donc équipé de sa propre matrice C, initialisée à la matrice identité, lors de la création ou du repositionnement de l'agent. L'adaptation de C

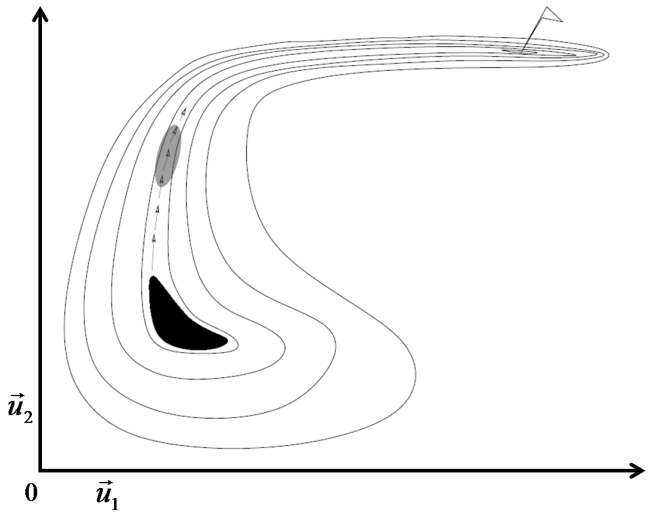


FIGURE 2.13 – Illustration de l'adaptation du voisinage d'un agent au paysage local de la fonction objectif, dans un espace de recherche à 2 dimensions.

s'effectue selon les équations suivantes :

$$p = \frac{n_s}{2 \times d} \tag{2.3.1}$$

Dans (2.3.1), le coefficient p est calculé à partir de n_s (le nombre de solutions candidates formant le voisinage d'un agent) et de d (la dimension de l'espace de recherche). Ensuite, p permet de calculer un coefficient c_{cov}, à partir de c_r (coefficient d'adaptation du pas d'un agent), comme suit :

$$c_{cov} = \frac{p \times c_r + 1}{p + 1} \tag{2.3.2}$$

Un autre coefficient, noté c_{id}, est également calculé comme suit :

$$c_{id} = (1 - c_r) \left(1 - \left|\vec{v}_{max}^{\,\mathrm{T}} \times \vec{D}\right|\right) \tag{2.3.3}$$

Dans (2.3.3), $\vec{v}_{max}$ est le vecteur propre normalisé (vecteur unitaire de même orientation que le vecteur propre correspondant) associé à la plus grande valeur propre de C, et calculé lors de la précédente adaptation de C. Le vecteur $\vec{D}$ est, quant à lui, le vecteur directeur unitaire correspondant au déplacement courant de l'agent. Dans cette équation, comme dans ce qui suit, la transposée d'une matrice A est notée A^{T}.

A partir des coefficients c_{cov} et c_{id} ainsi calculés, nous mettons à jour la matrice C de la manière suivante :

$$C' = I \times c_{id} + \left(C \times c_{cov} + D \times D^{\mathrm{T}}(1 - c_{cov})\right)(1 - c_{id}) \tag{2.3.4}$$

Dans (2.3.4), la matrice identité est notée I, et C' est une matrice utilisée pour mettre à jour C. Cette mise à jour s'effectue en affectant $(C'/eigen'_{max})$ à C, en désignant par $eigen'_{max}$ la plus grande valeur propre de C'. Ainsi, la plus grande valeur propre de C reste égale à 1 après chaque adaptation.

Ensuite, les valeurs propres $eigen_0, eigen_1, ..., eigen_d$ et les vecteurs propres normalisés $\vec{v}_0, \vec{v}_1, ..., \vec{v}_d$ sont calculés à partir de la matrice C mise à jour. Ces valeurs et vecteurs propres sont alors utilisés pour calculer un ensemble de points $S_a = \{\vec{P}'_1, \vec{P}'_2, ..., \vec{P}'_{n_s}\}$ à partir de l'ensemble précalculé de points $S = \{\vec{P}_1, \vec{P}_2, ..., \vec{P}_{n_s}\}$. Cet ensemble S_a est utilisé dans CMADO pour engendrer les solutions voisines de l'agent, tout comme S est utilisé dans MADO pour engendrer ces solutions. Le calcul de S_a s'effectue selon les équations suivantes :

$$B = \left[\sqrt{eigen_0} \times \vec{v}_0 \quad \cdots \quad \sqrt{eigen_d} \times \vec{v}_d \right] \tag{2.3.5}$$

Dans (2.3.5), nous calculons une matrice de changement de base, notée B. Cette matrice sert à calculer chaque point $\vec{P}'_i$ de S_a comme suit :

$$\vec{P}'_i = B \times \vec{P}_i \tag{2.3.6}$$

2.3.2 Résultats et discussion

Nous allons maintenant comparer les résultats obtenus par MADO et CMADO, en utilisant MPB (scénario 2) et la fonction Rosenbrock en 5 dimensions (précédemment définie dans l'équation (2.2.8)). Il s'agit en effet d'une fonction connue pour être mal conditionnée, et elle se prête donc bien à cette comparaison. L'espace de recherche utilisé pour cette fonction est $[-10; 10]^5$.

Pour MPB, l'algorithme est arrêté lorsque 5E+5 évaluations sont effectuées, et ses performances sont mesurées en utilisant l'*offline error*. Pour la fonction Rosenbrock, l'algorithme est arrêté lorsque la valeur de la meilleure solution trouvée devient inférieure à 0,01, et ses performances sont mesurées en utilisant le nombre d'évaluations effectuées jusqu'à la satisfaction de ce critère d'arrêt. Nous utilisons cette mesure de performance, pour la fonction Rosenbrock, car il est plus important en optimisation dynamique de trouver l'optimum global rapidement que précisément.

Les paramètres de MADO et de CMADO utilisés pour MPB sont ceux de la colonne « MPB » du tableau 2.1. Ceux utilisés pour la fonction Rosenbrock sont indiqués dans le tableau 2.4. Pour chacun de ces jeux de tests, les résultats sont moyennés sur 100 exécutions de l'algorithme, et présentés dans le tableau 2.5.

Paramètre	n_a	n_m	n_s	c_u	c_r	r_e	r_l	δ_t	δ_p
Valeur	1	0	3	0,5	0,8	0,01	7E−3	2	1E−9

TABLEAU 2.4 – Paramétrage de MADO et de CMADO pour la fonction Rosenbrock.

Algorithme	***Offline error* pour MPB**	**Nombre d'évaluations pour la fonction Rosenbrock**
CMADO	$0{,}58 \pm 0{,}09$	$15854{,}5 \pm 13137{,}8$
MADO	$0{,}60 \pm 0{,}11$	$276789{,}2 \pm 460376{,}1$

TABLEAU 2.5 – Comparaison de MADO et de CMADO en utilisant MPB et la fonction Rosenbrock.

Un test statistique de Wilcoxon-Mann-Whitney [Wilcoxon, 1945; Mann & Whitney, 1947], appliqué sur ces résultats, avec un niveau de confiance de 95%, ne permet pas de mettre en évidence un gain de performance significatif sur MPB. En revanche, les performances de CMADO sont significativement supérieures à celles de MADO sur la fonction Rosenbrock. Nous pouvons donc conclure qu'à défaut d'améliorer les performances de MADO sur MPB, la technique d'adaptation aux problèmes mal conditionnés les améliore de façon significative sur la fonction Rosenbrock, un problème typiquement mal conditionné.

Néanmoins, l'ajout de cette technique d'adaptation aux problèmes mal conditionnés augmente significativement la complexité de l'algorithme, notamment par le calcul de matrices de covariance, et par la nécessité de recourir à une décomposition de cette matrice en valeurs propres et en vecteurs propres. CMADO étant ainsi plus complexe que MADO, sans toutefois améliorer significativement ses performances sur MPB, nous présentons CMADO comme une variante à part.

2.4 Accélération par prédiction (PMADO)

En vue d'améliorer les performances de CMADO, une technique visant à prédire les régions d'intérêt de la fonction objectif, afin d'y placer dès que possible un agent de recherche locale, a été élaborée.

2.4.1 La technique de prédiction proposée

La version de CMADO incluant cette technique est appelée PMADO. La technique proposée fonctionne de la manière suivante :

1. Au cours d'un même *time span* (série d'évaluations consécutives de la fonction objectif, durant laquelle aucun changement du paysage de la fonction objectif ne survient), l'historique des solutions candidates évaluées par les agents, ainsi que les valeurs de la fonction objectif de ces solutions, sont archivés.
2. Dès que cette archive, notée A_p, atteint une taille suffisante, un processus est activé, en parallèle à la recherche effectuée par les agents. Ce processus a pour but de prédire, en se basant sur A_p, une zone d'intérêt de la fonction objectif. A_p est donc transmise à ce processus parallèle, puis vidée, pour pouvoir archiver de nouvelles solutions candidates.
   (a) Ce processus commence par apprendre l'allure de la fonction objectif, au moyen d'un réseau de neurones utilisant A_p comme base d'exemples. La fonction objectif estimée est ensuite optimisée au moyen d'un algorithme de recherche locale rapide (la recherche locale utilisée par les agents de CMADO).
   (b) A l'issue de cette optimisation de la fonction objectif estimée, un optimum, noté $\vec{O}_p$, est trouvé. Il est transmis au coordinateur (module de coordination des agents de CMADO), puis le processus d'apprentissage est désactivé, jusqu'à ce que A_p atteigne de nouveau une taille suffisante.
3. Ensuite, dès qu'un agent de CMADO doit démarrer une nouvelle recherche locale, le coordinateur lui fournit $\vec{O}_p$ comme solution initiale. Une fois $\vec{O}_p$ transmis à cet agent, le coordinateur efface $\vec{O}_p$ et ne s'en servira donc plus comme solution initiale d'autres agents.
4. On retourne ensuite à l'étape 1, pour qu'un nouvel apprentissage puisse se faire, utilisant une nouvelle base d'exemples. Le cycle des activations de ce processus prédictif parallèle est illustré à la figure 2.14.

La fonction estimée, plus simple que la véritable fonction, pourrait permettre de guider ainsi la recherche vers des zones prometteuses de l'espace de recherche.

La taille de A_p à partir de laquelle le processus prédictif est déclenché est fixée empiriquement à $round(\sqrt{10 \times d})$, en désignant par *round* une fonction arrondissant un nombre à l'entier le plus proche, et par d la dimension du problème.

Le réseau de neurones utilisé est de type *perceptron multicouche* (voir [Haykin, 2008]), disposant d'une couche cachée de c neurones avec, pour fonction de transfert, la fonction tangente

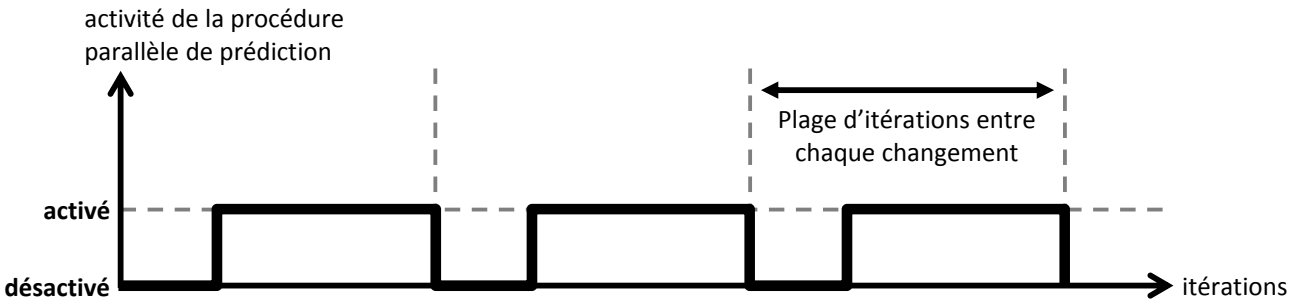


FIGURE 2.14 – Chronogramme représentant les périodes d'activation du processus prédictif effectué parallèlement aux recherches locales des agents de CMADO.

hyperbolique, et d'une couche de sortie composée d'un neurone avec, pour fonction de transfert, la fonction identité. Le nombre d'entrées du réseau de neurones est égal à la dimension d de la fonction objectif du problème. Le réseau n'a qu'une seule sortie, correspondant à la valeur prédite de la fonction objectif.

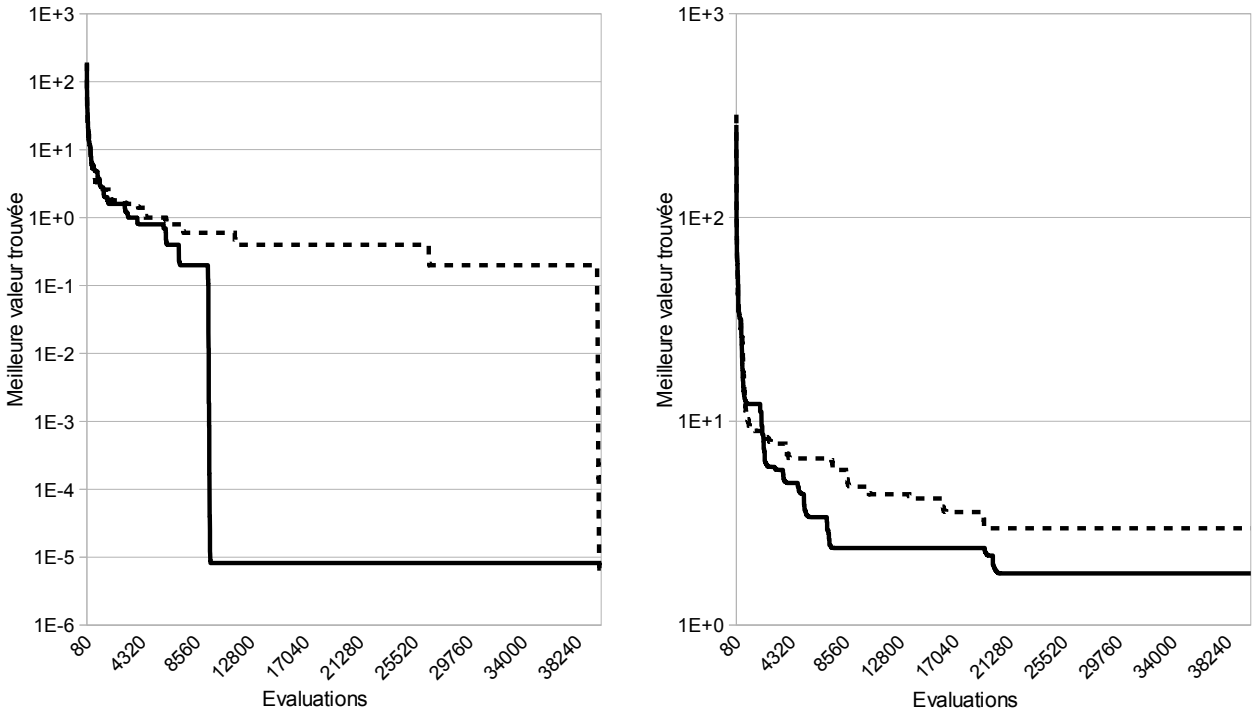
2.4.2 Résultats et discussion

Pour permettre un bon apprentissage de l'allure de la fonction objectif, le nombre de neurones dans la couche cachée doit être suffisamment grand. Dans les tests qui suivent, nous utilisons $c = 30$. Pour tester les performances de PMADO par rapport à CMADO, nous utilisons la fonction Rastrigin. Cette fonction est définie dans l'équation (2.4.1), où $\vec{x}$ est une solution à évaluer, x_i est sa $i^{\text{ième}}$ coordonnée et d est la dimension de l'espace de recherche. L'espace de recherche utilisé pour cette fonction est $[-10; 10]^d$. Il s'agit d'une fonction hautement multimodale : elle comporte un grand nombre d'optima locaux. Elle admet un seul optimum global, de valeur 0.

$$f(\vec{x}) = \sum_{i=1}^{d} (x_i^2 - 10\cos(2\pi x_i) + 10) \tag{2.4.1}$$

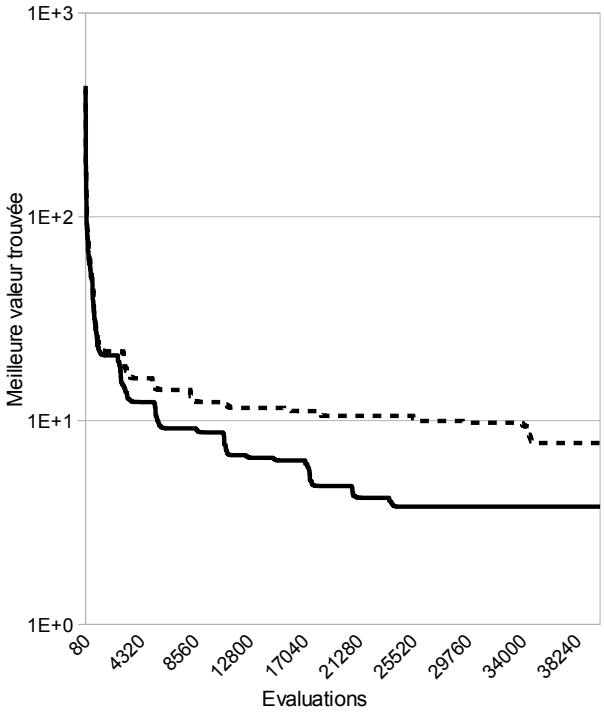
Les paramètres de CMADO et de PMADO utilisés sont donnés dans le tableau 2.6. Pour mesurer les performances de l'algorithme, nous utilisons la valeur de la meilleure solution trouvée, moyennée sur 5 exécutions. La convergence de PMADO et de CMADO, sur 4E+4 évaluations de la fonction objectif, est illustrée à la figure 2.15. Pour plus de clarté, une échelle logarithmique est utilisée sur l'axe des ordonnées.

Sur cette figure, nous observons que cette technique prédictive permet une amélioration de la convergence de CMADO sur la fonction Rastrigin. Cette amélioration est d'autant plus appréciable qu'il est important, en optimisation dynamique, d'avoir une convergence rapide.



(a)

(b)



(c)

FIGURE 2.15 – L'évolution de la meilleure solution trouvée, sur la fonction Rastrigin, en 3, 5 et 7 dimensions respectivement dans (a), (b) et (c). La courbe en trait plein correspond à PMADO, et celle en pointillés à CMADO.

Paramètre	n_a	n_m	n_s	c_u	c_r	r_e	r_l	δ_t	δ_p
Valeur	4	10	$round(0{,}5\,d + 2)$	0,5	0,8	0,2	7E−3	2	1E−5

TABLEAU 2.6 – Paramétrage de CMADO et de PMADO pour la fonction Rastrigin. La fonction *round* arrondit un réel, donné en argument, à l'entier le plus proche. La dimension du problème est notée d.

Néanmoins, l'ajout de cette technique accroît significativement la complexité de l'algorithme. De ce fait, nous présentons PMADO comme une variante à part, comme pour CMADO.

2.5 Conclusion

En conclusion, un nouvel algorithme d'optimisation dynamique, nommé MADO, a été présenté. Cet algorithme obtient de bons résulats sur les deux principaux jeux de tests (MPB et GDBG) en optimisation dynamique. Deux variantes de l'algorithme, nommées CMADO et PMADO, ont également été présentées : CMADO intègre une technique permettant d'améliorer les performances de l'algorithme sur des problèmes mal conditionnés, et PMADO intègre une technique prédictive permettant d'initier les recherches locales des agents dans des zones prometteuses de l'espace de recherche.

MADO peut encore être amélioré et simplifié de différentes façons. Dans le chapitre suivant, nous allons voir comment tenir compte des forces et des faiblesses de chaque composant utilisé dans MADO pour mettre au point un algorithme encore plus performant.

Chapitre Trois

Elaboration de notre algorithme d'optimisation dynamique MLSDO

3.1 Introduction

Nous avons vu au paragraphe 2.2.3.3 qu'un certain nombre de composants de MADO pouvaient être simplifiés ou supprimés, sans risque majeur de dégradation des performances de l'algorithme. C'est le cas notamment de l'heuristique de répulsion électrostatique, qui constitue une procédure complexe n'apportant pas d'amélioration importante des performances. Ainsi, par remplacements et améliorations successives des stratégies utilisées dans MADO, nous avons mis au point un nouvel algorithme plus simple et plus performant : MLSDO (*Multiple Local Search algorithm for Dynamic Optimization*) [Lepagnot et al., 2011c].

Cet algorithme repose toujours sur le principe d'utiliser des agents de recherche locale coordonnés, et sur un archivage des optima locaux trouvés. Ainsi, le schéma fonctionnel global de MADO, illustré à la figure 2.1 du paragraphe 2.2.1, reste valable pour MLSDO. La procédure générale des agents, décrite au paragraphe 2.2.1.3, a été simplifiée dans MLSDO, mais elle reste globalement similaire à celle de MADO. Elle est présentée à la figure 3.1. En revanche, les stratégies de recherche, d'archivage et de coordination ont été ré-étudiées. Nous présentons ainsi un nouvel algorithme, que nous allons maintenant décrire et analyser en détail.

3.2 Description de l'algorithme

Comme pour MADO, l'algorithme MLSDO est constitué d'agents coordonnés par un module dédié, le coordinateur. MLSDO comporte aussi un module de gestion de la mémoire, pour gérer l'archive des optima locaux. Nous allons décrire, dans cette section, chacune des stratégies utilisées par MLSDO.

3.2.1 Génération de l'ensemble initial d'agents

A l'initialisation de MLSDO, le coordinateur crée la population initiale d'agents. Le nombre d'agents créés est fixé par le paramètre n_a. Les positions initiales de ces agents ne sont pas

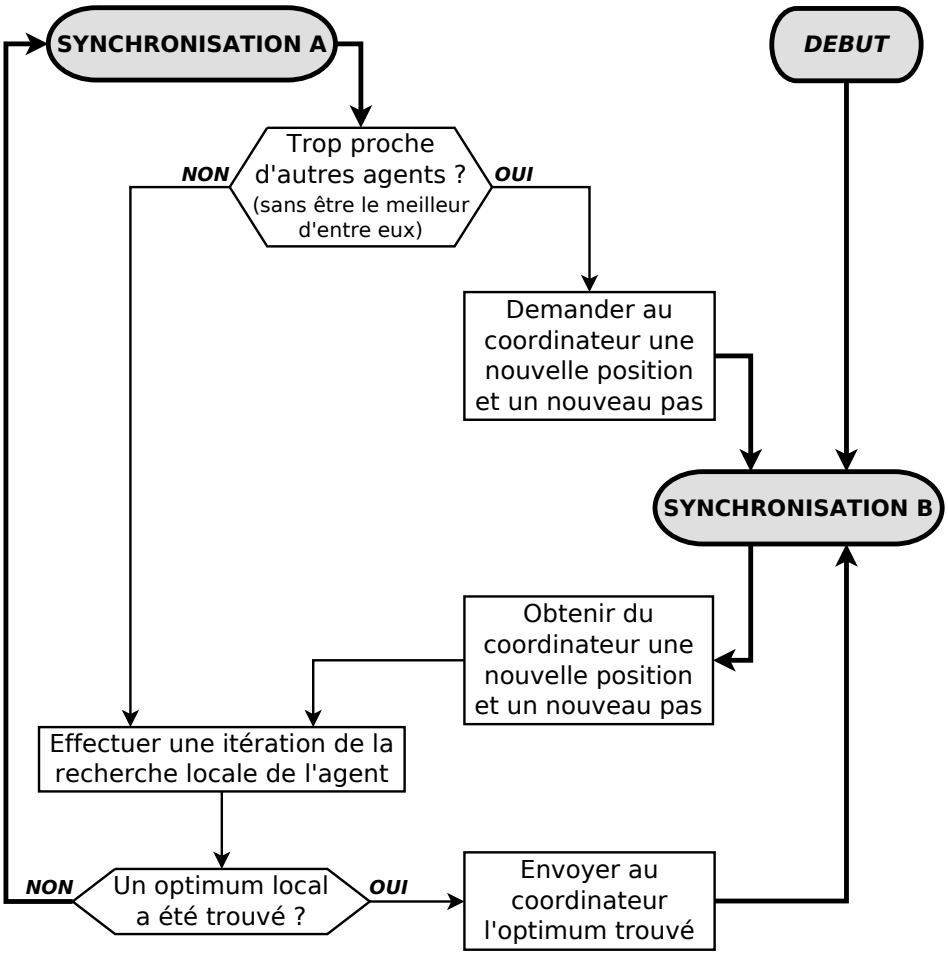


FIGURE 3.1 – Organigramme de la procédure principale d'un agent de MLSDO.

tirées aléatoirement, mais de façon à les répartir le plus possible dans l'espace de recherche : en les éloignant le plus possible les unes des autres.

Cette idée d'utiliser un ensemble initial d'agent ainsi très diversifié était également exploitée dans MADO, au moyen d'une heuristique de répulsion électrostatique. Plutôt que d'utiliser cette heuristique, nous proposons d'utiliser une heuristique plus simple. L'heuristique que nous proposons d'utiliser dans MLSDO est la même que celle qui servira à engendrer la solution initiale d'un agent devant être repositionné. Les n_a agents initiaux seront ainsi créés séquentiellement, et leur solution initiale sera engendrée par cette heuristique, que nous décrirons au paragraphe 3.2.3.

3.2.2 La stratégie d'exploration utilisée par les agents

Les agents de MLSDO peuvent être créés pour deux raisons : pour explorer l'espace de recherche (agents d'exploration), ou pour suivre un optimum local archivé lorsqu'un changement est détecté dans la fonction objectif (agents de suivi). Les agents effectuent leurs recherches

initialisationAgent

1 **Entrées :** $\vec{S}_{new}, r_{new}$
2 **Variables locales :** $\varnothing$
3 $\vec{S}_c \leftarrow \vec{S}_{new}$
4 évaluer $\vec{S}_c$
5 $\vec{D} \leftarrow \vec{0}$
6 $R \leftarrow r_{new}$
7 $U \leftarrow 0$
8 **retourner** $\left\{\vec{S}_c, \vec{D}, R, U\right\}$

PROCÉDURE 3.1 – Initialisation de la recherche locale d'un agent.

locales en se déplaçant pas à pas, de leur solution courante $\vec{S}_c$ à une meilleure solution voisine $\vec{S}'_c$, jusqu'à atteindre un optimum local.

Le pas R de la recherche locale d'un agent est initialisé à une valeur réelle, notée r_{new}, comprise dans l'intervalle $]0; 1]$. Tout comme pour MADO, un agent d'exploration nécessite un pas initial significativement supérieur à celui d'un agent de suivi. De ce fait, à l'initialisation de la recherche locale d'un agent, la valeur de r_{new} dépend des conditions de création de l'agent : s'il s'agit d'un agent de suivi, r_{new} est égal à r_l, sinon, r_{new} est égal à r_e, en désignant par r_l et r_e deux paramètres de MLSDO. De plus, lorsque la recherche locale d'un agent doit être réinitialisée (repositionnement de l'agent), r_{new} est égal à r_e.

L'initialisation de la recherche locale d'un agent est présentée dans la procédure 3.1. Cette procédure est exécutée à la création et lors du repositionnement d'un agent. Dans cette procédure, les entrées $\vec{S}_{new}$ et r_{new} correspondent à la solution initiale et au pas initial de l'agent, respectivement. Ces entrées sont fournies par le coordinateur. La fonction *évaluer*, quant à elle, calcule la valeur de la fonction objectif d'une solution donnée, et l'associe à cette solution. La variable $\vec{D}$ correspond au vecteur directeur du dernier déplacement de l'agent, et U correspond au produit scalaire cumulé. Tout comme dans MADO, $\vec{D}$ et U sont utilisées pour adapter R. Nous verrons par la suite comment cette adaptation s'effectue dans MLSDO. Enfin, le vecteur nul, noté $\vec{0}$, est ici un vecteur constitué de d zéros. Nous constatons ainsi qu'à l'initialisation de la recherche locale d'un agent, $\vec{D}$ et U sont initialisées au vecteur nul et à zéro, respectivement.

Nous allons maintenant nous focaliser sur la recherche locale d'un agent, dont la procédure principale est la procédure 3.2. Cette procédure correspond à l'état indiqué à la figure 3.1 par « effectuer une itération de la recherche locale de l'agent ».

rechercheLocale

1 **Entrées :** $\vec{S}_c, d, R, U, \vec{D}, r_l, \delta_{ph}, \delta_{pl}$
2 **Variables locales :** *stop*
3 $\left\{\vec{S}'_c, \vec{S}_w, \vec{D}'\right\} \leftarrow$ sélection$\left(\vec{S}_c, d, R\right)$
4 $stop \leftarrow$ critèreArrêt$\left(\vec{S}'_c, \vec{S}_w, \vec{S}_c, R, r_l, \delta_{ph}, \delta_{pl}\right)$
5 $\{U, R\} \leftarrow$ miseAJourDuPas$\left(\vec{S}'_c, \vec{S}_c, U, \vec{D}, \vec{D}', R\right)$
6 **si** $\vec{S}'_c \neq \vec{S}_c$ **alors**
7 $\quad \vec{S}_c \leftarrow \vec{S}'_c$
8 $\quad \vec{D} \leftarrow \vec{D}'$
9 **fin**
10 **si** $stop = vrai$ **alors**
11 $\quad$ critère d'arrêt de la recherche locale satisfait : $\vec{S}_c$ est l'optimum local trouvé
12 **fin**
13 **retourner** $\left\{\vec{S}_c, R, U, \vec{D}\right\}$

PROCÉDURE 3.2 – Procédure effectuant une itération de la recherche locale d'un agent.

Dans cette procédure, δ_{ph} et δ_{pl} sont des paramètres de MLSDO, que nous allons décrire. De plus, cette procédure principale fait appel aux sous-procédures *sélection*, *critèreArrêt* et *miseAJourDuPas*, que nous allons également détailler.

3.2.2.1 La sélection d'une meilleure solution voisine

Intéressons-nous d'abord à la sous-procédure *sélection*. Nous savons que, au cours de sa recherche locale, un agent se déplace de sa solution courante $\vec{S}_c$ vers une meilleure solution voisine $\vec{S}'_c$. La sélection de $\vec{S}'_c$ est effectuée par *sélection*, présentée dans la procédure 3.3.

Dans cette procédure, le $i^{\text{ième}}$ vecteur de base de l'espace de recherche (voir paragraphe 2.2.1.1) est noté $\vec{u}_i$. La fonction *évaluer*, comme pour la procédure 3.1, calcule la valeur de la fonction objectif d'une solution donnée, et l'associe à cette solution.

Le mécanisme de sélection d'une meilleure solution voisine, utilisé dans cette procédure, est le suivant : la solution $\vec{S}'_c$ devant remplacer $\vec{S}_c$ est tout d'abord initialisée à $\vec{S}_c$ (ligne 3). Ensuite, deux solutions candidates, notées $\vec{S}_{prev}$ et $\vec{S}_{next}$, sont évaluées par dimension de l'espace de recherche. Elle se tiennent de part et d'autre de $\vec{S}'_c$, à égale distance R de $\vec{S}'_c$ le long d'un axe de l'espace de recherche (lignes 6 et 7). Pour chaque axe de l'espace de recherche, la meilleure solution parmi $\vec{S}_{prev}$, $\vec{S}_{next}$ et $\vec{S}'_c$ est affectée à $\vec{S}'_c$ (lignes 14 à 18). A chaque itération de la

sélection

Entrées : $\vec{S}_c, d, R$
Variables locales : $i, \vec{S}_{prev}, \vec{S}_{next}, \vec{S}_i$
$\vec{S}'_c \leftarrow \vec{S}_c$
$\vec{S}_w \leftarrow \vec{S}_c$
pour $i = 1$ **à** d **faire**
    $\vec{S}_{prev} \leftarrow \vec{S}'_c - R \times \vec{u}_i$
    $\vec{S}_{next} \leftarrow \vec{S}'_c + R \times \vec{u}_i$
    si $\vec{S}_{prev}$ sort des bornes de l'espace de recherche **alors**
        $\vec{S}_{prev} \leftarrow$ le point le plus proche de $\vec{S}_{prev}$ dans l'espace de recherche
    fin
    si $\vec{S}_{next}$ sort des bornes de l'espace de recherche **alors**
        $\vec{S}_{next} \leftarrow$ le point le plus proche de $\vec{S}_{next}$ dans l'espace de recherche
    fin
    évaluer $\vec{S}_{prev}$ et $\vec{S}_{next}$
    $\vec{S}_i \leftarrow$ la meilleure solution parmi $\vec{S}_{prev}$ et $\vec{S}_{next}$
    si $\vec{S}_i$ est strictement meilleure que $\vec{S}'_c$ **alors**
        $\vec{S}'_c \leftarrow \vec{S}_i$
    fin
    $\vec{S}_i \leftarrow$ la plus mauvaise solution parmi $\vec{S}_{prev}$ et $\vec{S}_{next}$
    si $\vec{S}_i$ est plus mauvaise ou égale à $\vec{S}_w$ **alors**
        $\vec{S}_w \leftarrow \vec{S}_i$
    fin
fin
$\vec{D}' \leftarrow \vec{S}'_c - \vec{S}_c$
$\vec{D}' \leftarrow \frac{1}{\|\vec{D}'\|} \times \vec{D}'$
retourner $\left\{\vec{S}'_c, \vec{S}_w, \vec{D}'\right\}$

PROCÉDURE 3.3 – Mécanisme de sélection de la solution candidate $\vec{S}'_c$ qui va remplacer la solution courante $\vec{S}_c$ de l'agent.

recherche locale d'un agent, $2 \times d$ solutions candidates sont ainsi testées pour trouver la solution $\vec{S}'_c$ vers laquelle l'agent doit se déplacer. Ce mécanisme simple est proposé dans [Gardeux et al., 2009], où les auteurs l'utilisent pour résoudre des problèmes de grande dimension au moyen de recherches locales. Nous l'avons ainsi adapté et utilisé dans MLSDO.

A la fin de cette procédure (ligne 26), le vecteur directeur unitaire $\vec{D}'$ du déplacement en cours de l'agent (orienté de $\vec{S}_c$ à $\vec{S}'_c$), ainsi que la plus mauvaise solution $\vec{S}_w$ parmi les $2 \times d$ solutions candidates évaluées, sont également déterminés. Nous verrons par la suite que $\vec{D}'$ est utilisé dans l'adaptation du pas R de l'agent, et que $\vec{S}_w$ est utilisé dans le critère d'arrêt de la recherche locale de l'agent.

Afin d'illustrer ce mécanisme, les déplacements successifs d'un agent, après son initialisation, sont présentés à la figure 3.2 (a). Ces déplacements sont issus des pas que l'agent effectue sur chaque axe de l'espace de recherche, comme illustré à la figure 3.2 (b).

On remarque, figure 3.2 (b), que les axes de l'espace de recherche le long desquels un agent se déplace, au cours d'une itération de sa recherche locale, sont toujours choisis dans le même ordre. En effet, changer aléatoirement l'ordre dans lequel on parcourt les dimensions de l'espace de recherche, à chaque itération de la recherche locale, n'apporte aucune amélioration des performances de l'algorithme.

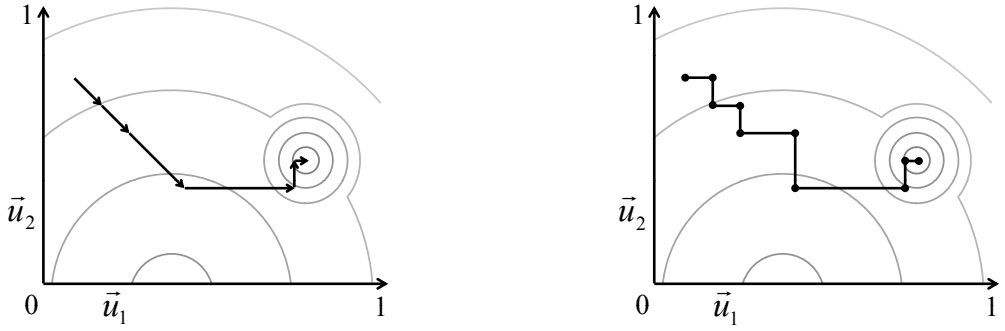


FIGURE 3.2 – Illustration de la recherche locale effectuée par un agent sur un exemple de fonction objectif à 2 dimensions. (a) montre les déplacements successifs de l'agent. Chacun d'eux est effectué durant une itération de la recherche locale de l'agent. (b) montre le chemin emprunté par l'agent, composé des meilleures solutions candidates évaluées le long de chaque axe de l'espace de recherche.

3.2.2.2 L'adaptation du pas d'un agent

Intéressons-nous maintenant à la sous-procédure *miseAJourDuPas*. Nous savons que le pas R d'un agent est adapté au fur et à mesure de ses déplacements dans l'espace de recherche. L'adaptation de R est effectuée par *miseAJourDuPas*, procédure 3.4. Cette adaptation est basée

sur la valeur du produit scalaire cumulé U, sur celle du paramètre r_e de MLSDO, et sur les vecteurs directeurs unitaires $\vec{D}'$ et $\vec{D}$, correspondant au déplacement courant et au déplacement précédent de l'agent, respectivement.

miseAJourDuPas

```
1  Entrées : S⃗'_c, S⃗_c, U, R, D⃗, D⃗', r_e
2  Variables locales : ∅
3  si S⃗'_c = S⃗_c alors
4  |  U ← 1/2 × U
5  |  R ← 1/2 × R
6  sinon
7  |  U ← 1/2 × U + ⟨D⃗, D⃗'⟩
8  |  si U < −r_e alors
9  |  |  U ← −r_e
10 |  sinon si U > r_e alors
11 |  |  R ← 2 × R
12 |  |  si R > 1 alors
13 |  |  |  R ← 1
14 |  |  fin
15 |  |  U ← 0
16 |  fin
17 fin
18 retourner {U, R}
```

PROCÉDURE 3.4 – Adaptation du pas d'un agent.

Dans cette procédure, l'adaptation du pas R s'effectue de la manière suivante :

- Si la procédure *sélection* (procédure 3.3) ne trouve pas de solution meilleure que $\vec{S}_c$ dans le voisinage de l'agent (si $\vec{S}'_c = \vec{S}_c$, à la ligne 3), alors R est divisé par deux (ligne 5).
- Si la valeur du produit scalaire cumulé U, calculé selon l'équation (3.2.1), indique que les déplacements de l'agent sont orientés dans une même direction (si $U > r_e$, à la ligne 10), alors R est doublé pour accélérer la convergence de l'agent (ligne 11). La valeur de U est alors remise à zéro (ligne 15).

De plus, pour éviter que U n'atteigne de trop fortes valeurs négatives, nous contraignons U à rester supérieur ou égal à $-r_e$ (lignes 8 et 9).

Le produit scalaire cumulé, déjà utilisé dans MADO pour adapter le pas des agents (voir paragraphe 2.2.2.2), est basé sur l'historique des déplacements d'un agent. Dans MLSDO, il

est calculé selon l'équation (3.2.1), où U_n est le produit scalaire cumulé des vecteurs directeurs unitaires $\vec{D}_n$ des déplacements de l'agent, à la $n^{\text{ième}}$ itération de sa recherche locale.

$$U_n = \begin{cases} \frac{1}{2} \times U_{n-1} + \langle \vec{D}_{n-1}, \vec{D}_n \rangle & \text{si } n > 0 \\ \\ 0 & \text{sinon} \end{cases} \tag{3.2.1}$$

Dans la procédure *miseAJourDuPas* (procédure 3.4), la suite U_n de l'équation (3.2.1) correspond à U, $\vec{D}_{n-1}$ correspond à $\vec{D}$, et $\vec{D}_n$ correspond à $\vec{D}'$ (les vecteurs directeurs unitaires du déplacement précédent et du déplacement courant de l'agent, respectivement).

3.2.2.3 Le critère d'arrêt de la recherche locale d'un agent

Intéressons-nous enfin à la sous-procédure *critèreArrêt*, qui détermine si le critère d'arrêt de la recherche locale d'un agent est satisfait ou non. Elle est basée sur les paramètres r_l, δ_{ph} et δ_{pl} de MLSDO, ainsi que sur la meilleure et la plus mauvaise solution voisine de $\vec{S}_c$, notées $\vec{S}'_c$ et $\vec{S}_w$, respectivement. Il s'agit de la procédure 3.5.

Lorsque le critère d'arrêt de la recherche locale de l'agent est satisfait, cette procédure retourne $vrai$, sinon, elle retourne $faux$. Le critère d'arrêt ne peut être satisfait que si l'agent a « suffisamment » convergé vers un optimum local : si le pas R de sa recherche locale devient inférieur au pas initial r_l d'un agent de suivi (ligne 3). Si tel est le cas, alors nous utilisons l'un des deux critères de stagnation suivants, chacun ayant un niveau de précision différent : si S_c est la meilleure solution trouvée par MLSDO depuis la dernière détection d'un changement dans la fonction objectif (ligne 9), alors un critère de stagnation de précision δ_{ph} élevée est utilisé (ligne 10). Sinon, nous utilisons un critère de stagnation de précision δ_{pl} plus faible (ligne 14). La valeur de δ_{ph} doit donc rester plus petite que celle de δ_{pl}. Ainsi, nous évitons de converger avec précision vers des optima locaux de faible qualité, ce qui permet d'économiser des évaluations de la fonction objectif. Seule la meilleure solution trouvée par l'algorithme est affinée avec une grande précision.

3.2.3 Le maintien de la diversité

Nous avons vu dans le chapitre précédent, au paragraphe 2.2.3.3, que l'utilisation d'un rayon d'exclusion r_e attribué à chaque agent était une stratégie efficace. Nous reprenons cette stratégie, pour éviter que plusieurs agents n'explorent une même zone de l'espace de recherche (voir paragraphe 2.2.2.4), dans MLSDO. Ainsi, lorsqu'un agent détecte la présence d'autres agents

critèreArrêt

Entrées : $\vec{S}'_c, \vec{S}_w, \vec{S}_c, R, r_l, \delta_{ph}, \delta_{pl}$
Variables locales : p
si $R < r_l$ **alors**
  si $\vec{S}'_c \neq \vec{S}_c$ **alors**
    $p \leftarrow \left|\text{fitness}\left(\vec{S}'_c\right) - \text{fitness}\left(\vec{S}_c\right)\right|$
  sinon
    $p \leftarrow \left|\text{fitness}\left(\vec{S}_w\right) - \text{fitness}\left(\vec{S}_c\right)\right|$
  fin
  si $\vec{S}_c$ est la meilleure solution trouvée depuis le dernier changement dans la fonction objectif **alors**
    si $p \leq \delta_{ph}$ **alors**
      retourner $vrai$
    fin
  sinon
    si $p \leq \delta_{pl}$ **alors**
      retourner $vrai$
    fin
  fin
fin
retourner $faux$

PROCÉDURE 3.5 – Procédure qui teste la satisfaction du critère d'arrêt de la recherche locale d'un agent. La fonction $fitness$ retourne la valeur de la fonction objectif d'une solution donnée.

situés à une distance inférieure à r_e, alors seul l'agent ayant la meilleure solution courante continue sa recherche à cet endroit. Les autres agents en conflit dans cette zone doivent redémarrer leur recherche locale ailleurs dans l'espace de recherche.

Ce redémarrage, ou repositionnement, s'applique également aux agents ayant terminé leur recherche locale. Dans ce cas, le coordinateur transmet d'abord l'optimum local trouvé par l'agent au module de gestion de la mémoire, afin d'archiver cet optimum (nous verrons au paragraphe 3.2.5 sous quelles conditions un optimum local est archivé dans MLSDO). Ensuite, le coordinateur transmet à l'agent une nouvelle solution initiale $\vec{S}_{new}$ et un nouveau pas initial r_{new}, afin que l'agent démarre une nouvelle recherche locale. Ce redémarrage s'effectue en exécutant la procédure *initialisationAgent* (procédure 3.1) avec, en entrées, $\vec{S}_{new}$ et r_{new}. Dans le cas d'un redémarrage de la recherche locale d'un agent, r_{new} est égal à r_e.

La procédure 3.6 permet au coordinateur de trouver la nouvelle solution initiale $\vec{S}_{new}$ d'un

repositionnementOuDestructionAgent

Entrées : N, n_a, d, A_m, A_i, r_e
Variables locales : $d_0, \vec{S}_{new}$
si $N > n_a$ **alors**
    détruire l'agent
fin
$\left\{d_0, \vec{S}_{new}\right\} \leftarrow$ nouvelleSolutionInitiale(d, A_m, A_i)
si $(N > 1)$ et $(d_0 \leq r_e)$ **alors**
    détruire l'agent
sinon
    repositionner l'agent en $\vec{S}_{new}$ avec un pas initial de r_e
fin

PROCÉDURE 3.6 – Procédure qui repositionne ou qui détruit un agent.

agent devant redémarrer sa recherche locale. Dans cette procédure, le nombre d'agents actuellement existants est noté N, et le nombre maximum d'agents d'exploration autorisés est noté n_a (il s'agit d'un paramètre de MLSDO, qui correspond également au nombre d'agents créés à l'initialisation de MLSDO). L'archive des optima locaux trouvés par les agents est notée A_m. Enfin, nous utilisons ici également une archive, notée A_i, qui contient les dernières solutions initiales engendrées pour initialiser la recherche locale des agents devant être créés ou repositionnés. La capacité de l'archive A_i est identique à celle de A_m (nous verrons au paragraphe 3.2.5 comment cette capacité est calculée). A chaque fois qu'un changement est détecté dans la fonction objectif, l'archive A_i est vidée.

Cette procédure crée une solution initiale éloignée des autres agents, et des solutions de A_m et de A_i, en exécutant une sous-procédure appelée *nouvelleSolutionInitiale* (procédure 3.7). Il s'agit d'une heuristique qui commence par engendrer aléatoirement, selon une distribution uniforme, plusieurs solutions initiales possibles dans l'espace de recherche. Elle sélectionne ensuite l'une de ces solutions initiales possibles, et la retourne. Le mécanisme de sélection de cette heuristique est le suivant : pour chaque solution initiale possible, notée $\vec{S}$, la distance à la plus proche solution parmi l'ensemble des solutions courantes des agents, et des solutions de A_m et de A_i, est calculée. Ensuite, la solution $\vec{S}$ sélectionnée est celle pour laquelle cette distance, notée d_0, est la plus grande.

Nous avons fixé empiriquement le nombre de solutions $\vec{S}$ engendrées à $10 \times d$ (où d désigne la dimension de l'espace de recherche). Ceci correspond à un compromis entre le temps de calcul

nouvelleSolutionInitiale

Entrées : d, A_m, A_i
Variables locales : $\vec{S}, d_1, d_2, a_i, \vec{S}_c, \vec{S}_i$
$d_0 \leftarrow -\infty$
répéter $10 \times d$ fois
    $\vec{S} \leftarrow$ une solution générée aléatoirement selon une distribution uniforme dans l'espace de recherche
    $d_1 \leftarrow +\infty$
    pour tout agent a_i **faire**
        $\vec{S}_c \leftarrow$ la solution courante de a_i
        $d_2 \leftarrow$ la distance entre $\vec{S}$ et $\vec{S}_c$
        si $d_2 < d_1$ **alors**
            $d_1 \leftarrow d_2$
        fin
    fin
    pour tout $\vec{S}_i \in (A_m \cup A_i)$ **faire**
        $d_2 \leftarrow$ la distance entre $\vec{S}$ et $\vec{S}_i$
        si $d_2 < d_1$ **alors**
            $d_1 \leftarrow d_2$
        fin
    fin
    si $d_1 > d_0$ **alors**
        $d_0 \leftarrow d_1$
        $\vec{S}_{new} \leftarrow \vec{S}$
    fin
fin
retourner $\left\{d_0, \vec{S}_{new}\right\}$

PROCÉDURE 3.7 – Génération d'une nouvelle solution initiale pour un agent.

et la précision de cette heuristique.

Si la nouvelle position initiale $\vec{S}_{new}$ d'un agent, ainsi trouvée, est suffisamment éloignée (d'une distance supérieure au rayon d'exclusion r_e) des autres agents et des solutions de A_m et de A_i (si $d_0 > r_e$, à la ligne 7 de la procédure 3.6), alors l'agent est repositionné sur $\vec{S}_{new}$ (ligne 10). Sinon, l'espace de recherche est considéré comme saturé d'agents, auquel cas l'agent est détruit (ligne 8). La destruction de l'agent intervient également lorsque le nombre d'agents existants N est supérieur à n_a (lignes 3 à 5). Il est en effet possible d'avoir $N > n_a$, car des agents de suivi sont créés lorsqu'un changement est détecté dans la fonction objectif. En détruisant les agents devant être repositionnés, lorsque $N > n_a$, nous régulons ainsi le nombre d'agents existants.

3.2.4 La détection de changements et le suivi des optima locaux

Comme dans MADO, le coordinateur effectue la détection de changements dans la fonction objectif par réévaluation d'une solution. En revanche, dans MLSDO, la solution réévaluée est choisie aléatoirement parmi l'ensemble des solutions de l'archive A_m et des solutions courantes des agents. Si la valeur de cette solution a changé, les opérations suivantes sont effectuées : les solutions courantes des agents, ainsi que les optima locaux archivés, sont réévalués. Ensuite, la procédure 3.8 de création d'agents additionnels est exécutée.

Dans MADO, la détection de changements se fait en réévaluant, si possible, le meilleur optimum de l'archive des optima locaux. Néanmoins, pour certains problèmes, il se peut que la fonction objectif ne change que localement, i.e. les valeurs de la fonction objectif peuvent rester inchangées dans certaines zones de l'espace de recherche. Si le meilleur optimum de l'archive se trouve dans l'une de ces zones, le changement dans la fonction objectif ne sera pas détecté par l'algorithme. C'est la raison pour laquelle nous avons proposé, dans MLSDO, de réévaluer une solution choisie aléatoirement, parmi l'ensemble des solutions de l'archive des optima locaux, et des solutions courantes des agents.

Dans la procédure 3.8, les fonctions *max* et *min* retournent respectivement la plus grande et la plus petite valeur d'un ensemble de valeurs donné. Le nombre d'optima locaux présents dans l'archive A_m est noté m.

Cette procédure ne se contente pas de créer des agents de suivi, mais elle peut également créer de nouveaux agents d'exploration, si nécessaire. En effet, la procédure commence par créer au plus n_c agents de suivi (lignes 5 à 9), n_c désignant un paramètre de MLSDO. Le nombre exact d'agents de suivi créés est calculé à la ligne 3. Ensuite, si, malgré l'ajout de ces agents de

ajoutAgents

1 **Entrées :** $n_a, n_c, N, m, A_m, r_l, r_e$
2 **Variables locales :** $n_t, n_n, \vec{O}_{best}$
3 $n_t \leftarrow \max(0, \min(n_a + n_c - N, n_c, m))$
4 $n_n \leftarrow \max(0, \min(n_a - N - n_t, m - n_t))$
5 **répéter** n_t fois
6 $\quad \vec{O}_{best} \leftarrow$ le meilleur optimum local de A_m
7 $\quad A_m \leftarrow A_m - \{\vec{O}_{best}\}$
8 $\quad$ créer un agent ayant pour solution initiale $\vec{O}_{best}$ et pour pas initial r_l
9 **fin**
10 **répéter** n_n fois
11 $\quad \vec{O}_{best} \leftarrow$ le meilleur optimum local de A_m
12 $\quad A_m \leftarrow A_m - \{\vec{O}_{best}\}$
13 $\quad$ créer un agent ayant pour solution initiale $\vec{O}_{best}$ et pour pas initial r_e
14 **fin**

PROCÉDURE 3.8 – Création d'agents additionnels, lorsqu'un changement est détecté dans la fonction objectif.

suivi, le nombre N d'agents existants est inférieur à n_a, alors au plus $n_a - N$ agents d'exploration sont également créés (lignes 10 à 14). Le nombre exact d'agents d'exploration créés est calculé à la ligne 4.

La création de ces agents additionnels repose sur l'utilisation des meilleurs optima locaux de A_m. En effet, à chaque fois qu'un agent est créé dans cette procédure, le meilleur optimum local de l'archive est utilisé comme solution initiale de la recherche locale de l'agent (lignes 6 et 11). Cet optimum est ensuite retiré de A_m (lignes 7 et 12).

A la différence de MADO, où un agent de suivi est créé pour chaque optimum local archivé, au plus n_c agents de suivi sont créés dans MLSDO. Nous régulons ainsi le nombre de ces agents additionnels. En effet, la création d'un agent de suivi pour chaque optimum archivé peut engendrer un ralentissement significatif de la convergence de l'algorithme, lorsque beaucoup d'optima locaux nécessitent d'être archivés.

3.2.5 La gestion de l'archive des optima locaux

Le module de gestion de la mémoire maintient l'archive A_m des optima locaux trouvés par les agents. Comme dans MADO, il est nécessaire de borner cette archive pour que la consom-

mation de mémoire et l'effort de calcul restent raisonnables. Dans MLSDO, nous avons choisi de limiter la capacité de A_m à n_m solutions, en calculant n_m selon l'équation (3.2.2). La capacité de l'archive est ainsi déterminée automatiquement par MLSDO, à la différence de MADO, où elle devait être fixée manuellement par l'utilisateur. Dans cette expression, la fonction *round* arrondit un réel, donné en argument, à l'entier le plus proche. Cette équation, définie empiriquement, permet ainsi de calculer la capacité de l'archive à partir du rayon d'exclusion r_e des agents et de la dimension d de l'espace de recherche.

$$n_m = \text{round}\left(\frac{d}{r_e}\right) \tag{3.2.2}$$

Dans MLSDO, seul un sous-ensemble d'optima locaux archivés sont retirés de l'archive, afin de suivre leurs déplacements après un changement (régulation du nombre d'agents de suivi créés). Le suivi et la mise à jour des autres optima locaux de l'archive ne sont donc pas effectués. La gestion de l'archive nécessite alors d'introduire un « drapeau » *isNotUpToDate*, qui indique si un changement a été détecté dans la fonction objectif depuis l'archivage d'un optimum local donné : si un changement a eu lieu, il retourne $vrai$, sinon, il retourne $faux$.

Ainsi, lorsqu'un nouvel optimum local, noté $\vec{O}_c$, est trouvé par un agent, le module de gestion de la mémoire décide s'il doit être ajouté ou non à A_m selon les règles suivantes :

- Si l'archive n'est pas pleine, alors $\vec{O}_c$ y est ajouté.
- Si l'archive est pleine, alors $\vec{O}_c$ n'est ajouté à l'archive que si sa valeur est meilleure ou égale à celle du plus mauvais optimum de l'archive. Dans ce cas, $\vec{O}_c$ doit remplacer l'un des optima de l'archive. Ce remplacement est effectué de la manière suivante :
  - S'il existe un ou plusieurs optima de l'archive pour lesquels *isNotUpToDate* retourne $vrai$, alors le plus mauvais de ces optima est remplacé par $\vec{O}_c$.
  - Sinon, c'est le plus mauvais optimum de l'archive qui est remplacé par $\vec{O}_c$.
- Si une ou plusieurs solutions de l'archive sont « trop proches » de $\vec{O}_c$ (une solution archivée est considérée « trop proche » de $\vec{O}_c$ lorsqu'elle se trouve à une distance de $\vec{O}_c$ inférieure ou égale à la moyenne géométrique de r_l et de r_e), alors toutes ces solutions, proches les unes des autres, sont considérées comme « dominées » par la meilleure d'entre elles. Dans ce cas, cet ensemble de solutions est remplacé par la meilleure d'entre elles. Cette procédure de remplacement est également utilisée dans MADO (procédure 2.7).

3.3 Résultats et analyse

Comme pour MADO, nous allons tester MLSDO en utilisant le *Moving Peaks Benchmark* (MPB) et le *Generalized Dynamic Benchmark Generator* (GDBG), décrits au paragraphe 1.4.2. Nous commençons par une discussion sur le paramétrage de MLSDO et par la présentation des valeurs des paramètres utilisées pour ces deux jeux de tests. Ensuite, nous effectuons une analyse de la complexité de MLSDO, ainsi qu'une analyse de la sensibilité de ses paramètres. Puis, nous effectuons une analyse expérimentale des stratégies utilisées dans l'algorithme. Enfin, nous réalisons une étude de la convergence de l'algorithme, ainsi qu'une comparaison de ses performances à celles des autres algorithmes en compétition dans la littérature.

3.3.1 Paramétrage

Le paramétrage optimal de MLSDO, réalisé empiriquement pour MPB et GDBG, est donné dans le tableau 3.1.

Nom	Type	Intervalle	MPB	GDBG	Courte description
r_l	réel	$]0; r_e[$	0,005	0,005	pas initial d'un agent de suivi
δ_{ph}	réel	$[0; \delta_{pl}]$	0,001	0,001	paramètre du critère de stagnation de haute précision de la recherche locale des agents
δ_{pl}	réel	$[\delta_{ph}; +\infty]$	1,5	1,5	paramètre du critère de stagnation de faible précision de la recherche locale des agents
n_a	entier	$[1; 10]$	1	5	nombre maximum d'agents d'exploration
n_c	entier	$[0; 20]$	10	0	nombre maximum d'agents de suivi créés à la détection d'un changement dans la fonction objectif
r_e	réel	$]0; 1]$	0,12	0,10	rayon d'exclusion des agents, et pas initial d'un agent d'exploration

Tableau 3.1 – Paramétrage de MLSDO pour MPB et GDBG.

Nous avons fixé le nombre d'agents $n_a = 1$ pour MPB, car l'algorithme doit converger plus rapidement pour ce problème. Néanmoins, avoir plus d'un agent d'exploration peut conduire à un gain de performance pour d'autres problèmes, comme c'est le cas pour GDBG et pour celui basé sur la fonction Rosenbrock, que nous verrons plus tard (paragraphe 3.3.3).

Le paramètre δ_{pl} du critère de stagnation de faible précision doit être adapté à la fréquence des changements dans la fonction objectif (nombre d'évaluations pouvant être effectuées entre

deux changements consécutifs). En effet, avec une valeur trop faible de δ_{pl}, l'agent consommera beaucoup d'évaluations pour intensifier sa solution courante, au détriment de la diversification. Si les changements dans la fonction objectif sont fréquents, l'espace de recherche risque alors de ne pas être suffisamment exploré entre deux changements consécutifs. De ce fait, plus le nombre d'évaluations entre deux changements dans la fonction objectif est faible, plus la valeur de δ_{pl} doit être grande. Cela signifie que le compromis entre la précision des optima locaux trouvés (intensification) et l'exploration de l'espace de recherche (diversification) doit favoriser la diversification, lorsque les changements dans la fonction objectif se produisent fréquemment.

Comme pour MADO, le pas initial d'un agent de suivi, r_l, doit être suffisamment petit pour que l'agent ne quitte pas la zone d'attraction de l'optimum local qu'il doit suivre, pour aller explorer plus largement l'espace de recherche. Le rayon d'exclusion r_e doit, quant à lui, correspondre au rayon de la zone d'attraction d'un optimum local. Nous proposons donc $r_l < r_e$. Néanmoins, si la valeur de r_l est très inférieure à la distance dont s'est déplacé l'optimum que doit suivre l'agent, alors la convergence de l'agent vers cet optimum s'en trouvera ralentie. Le pas r_l doit donc être adapté à la sévérité des changements dans la fonction objectif.

Enfin, le paramètre n_c correspond au nombre d'optima locaux archivés devant être suivis, après un changement dans la fonction objectif. Plus la sévérité des changements est forte, plus la valeur de n_c va tendre vers 0. Au delà d'une certaine sévérité, les optima locaux ne peuvent plus être suivis et la création d'agents de suivi devient inutile ($n_c = 0$). Au contraire, si les changements sont suffisamment doux, alors le suivi des optima est possible, et la valeur de n_c correspond au nombre d'optima prometteurs à suivre.

3.3.2 Analyse de complexité

L'ensemble des procédures de MLSDO est exécuté, et répété, entre deux changements consécutifs détectés dans la fonction objectif. De ce fait, nous étudions la complexité de MLSDO entre deux détections de changement. Les processus exécutés entre ces détections sont présentés dans le tableau 3.2, avec leurs complexités. La complexité totale de MLSDO est calculée en additionnant les complexités de chacun de ces processus.

Ainsi, pour un jeu de paramètres de MLSDO fixé, en substituant l'expression de l'équation (3.2.2) à n_m dans l'expression de la complexité de MLSDO, nous obtenons une complexité en $O(d^3)$ pour MLSDO, d désignant la dimension de l'espace de recherche.

Processus	**Complexité**
Repositionnement des agents explorant une même zone de l'espace de recherche	$O(d^2(n_a + n_c)(n_a + n_c + n_m))$
Exécution d'une itération de la recherche locale de chaque agent	$O(d(n_a + n_c))$
Archivage des optima locaux trouvés par les agents	$O(d\, n_m(n_a + n_c))$
Détection d'un changement dans la fonction objectif, et réévaluation des solutions des agents et de A_m, si un changement est détecté	$O(n_a + n_c + n_m)$
Création des agents de suivi, si un changement est détecté	$O(n_a + n_c)$
Repositionnement des agents dont le critère d'arrêt est satisfait	$O(d^2(n_a + n_c)(n_a + n_c + n_m))$
Complexité totale de MLSDO	$O(d^2(n_a + n_c)(n_a + n_c + n_m))$

TABLEAU 3.2 – Analyse de la complexité de MLSDO.

3.3.3 Analyse de la sensibilité des paramètres

Comme pour MADO (paragraphe 2.2.3.2), nous utilisons MPB (scénario 2) et la fonction Rosenbrock en 5 dimensions (précédemment définie dans l'équation (2.2.8)) pour étudier la sensibilité de MLSDO. Nous l'étudions en faisant varier chaque paramètre de l'algorithme, en laissant les autres paramètres fixés à des valeurs par défaut. Ces valeurs par défaut sont celles de la colonne MPB du tableau 3.1, pour MPB, et celles du tableau 3.3, pour la fonction Rosenbrock. L'espace de recherche utilisé pour cette dernière est $[-10; 10]^5$.

Paramètre	n_a	n_c	r_l	r_e	δ_{ph}	δ_{pl}
Valeur	2	0	0,005	0,03	1E−10	1E−04

TABLEAU 3.3 – Paramétrage de MLSDO pour la fonction Rosenbrock.

Pour MPB, l'algorithme est arrêté lorsque 5E+5 évaluations sont effectuées, et ses performances sont mesurées en utilisant l'*offline error*. Pour la fonction Rosenbrock, l'algorithme est arrêté lorsque la valeur de la meilleure solution trouvée devient inférieure à 0,01, et ses performances sont mesurées en utilisant le nombre d'évaluations effectuées jusqu'à la satisfaction de ce critère d'arrêt. Nous utilisons cette mesure de performance, pour la fonction Rosenbrock, car il est plus important en optimisation dynamique de trouver l'optimum global rapidement que précisément.

Pour chacun de ces jeux de tests, les résultats sont moyennés sur 100 exécutions de l'algorithme, et présentés aux figures 3.3 et 3.4.

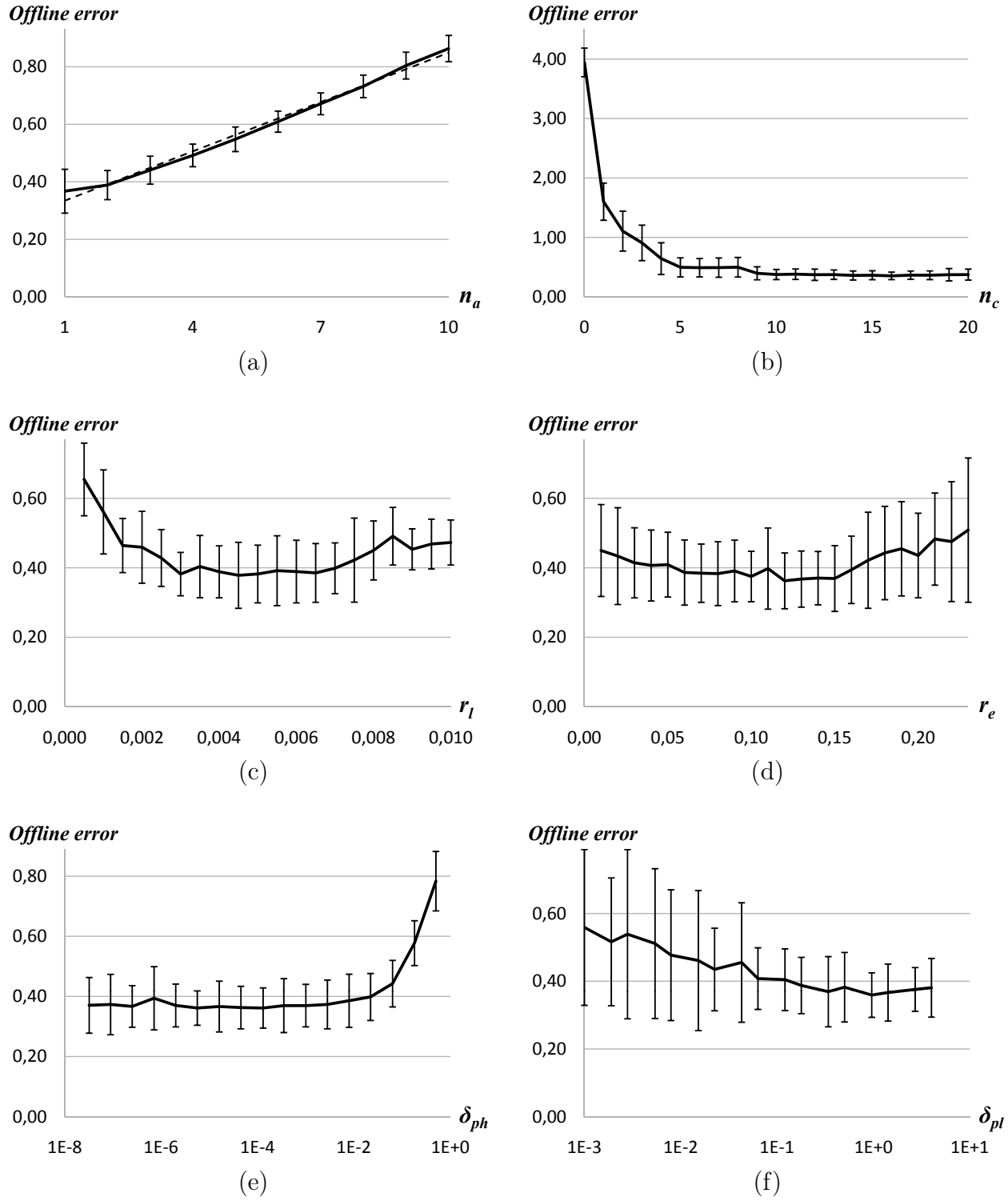


FIGURE 3.3 – L'évolution de l'*offline error*, sur MPB, pour différentes valeurs de n_a, n_c, r_l, r_e, δ_{ph} et δ_{pl}, est donnée en (a), (b), (c), (d), (e) et (f), respectivement.

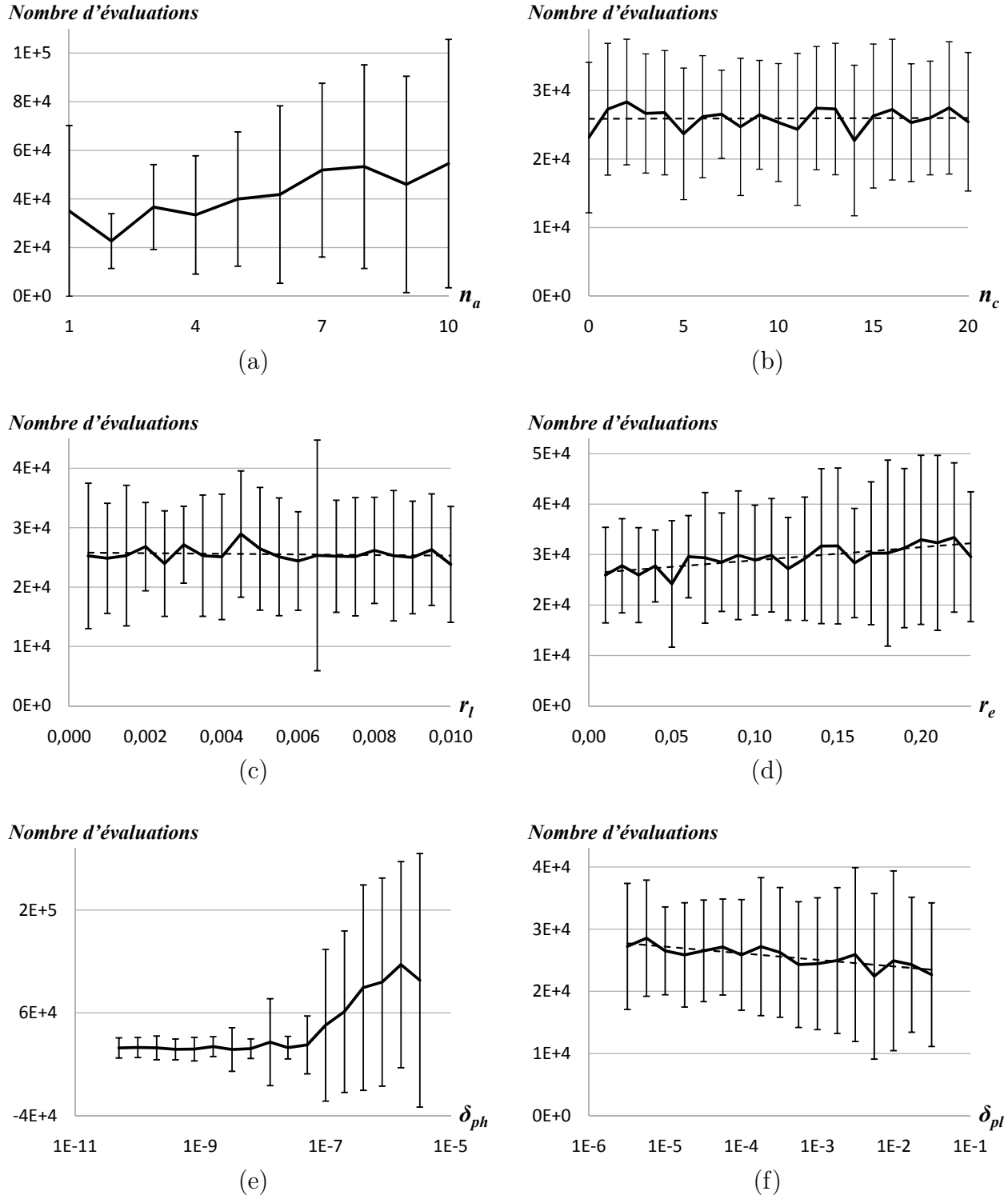


FIGURE 3.4 – L'évolution du nombre d'évaluations nécessaires de la fonction Rosenbrock, pour différentes valeurs de n_a, n_c, r_l, r_e, δ_{ph} et δ_{pl}, est donnée en (a), (b), (c), (d), (e) et (f), respectivement.

Comme nous le constatons sur ces figures, pour les deux problèmes, plus n_a augmente, et plus les performances de MLSDO se détériorent. Les meilleurs résultats sont obtenus avec $n_a = 1$ pour MPB et $n_a = 2$ pour la fonction Rosenbrock. Concernant le paramètre n_c, la variation de sa valeur ne perturbe pas significativement les performances de MLSDO pour la fonction Rosenbrock. En revanche, pour MPB, les performances de MLSDO s'améliorent en augmentant la valeur de n_c. Ainsi, n_c n'a d'importance que pour des problèmes dynamiques, et il doit être assez grand pour qu'un nombre suffisant d'optima locaux soient suivis. En l'occurrence, une valeur supérieure ou égale à 10 est bien adaptée à MPB.

Pour MPB, il apparaît que r_l, r_e et δ_{pl} se comportent de manière similaire, et qu'ils peuvent perturber significativement les performances de MLSDO s'ils ne sont pas correctement ajustés. En revanche, faire varier r_l n'influence pas les performances de MLSDO sur la fonction Rosenbrock, et faire varier r_e et δ_{pl} les perturbe peu sur cette fonction.

Le comportement de δ_{ph} est similaire pour les deux problèmes : sa valeur doit être suffisamment petite pour que MLSDO converge avec précision. La valeur optimale de δ_{ph} est dépendante du problème. Néanmoins, il n'est pas nécessaire de régler avec précision ce paramètre, dès lors qu'il est choisi suffisamment « petit » (inférieur à sa valeur optimale).

3.3.4 Analyse expérimentale des stratégies utilisées

Afin de justifier l'usage de chaque composant de MLSDO, nous allons évaluer diverses variantes de MLSDO, chacune ayant été simplifiée en supprimant l'une des stratégies employées dans l'algorithme original de MLSDO. Cette évaluation est effectuée sur MPB (scénario 2), en moyennant les résultats obtenus sur 100 exécutions de l'algorithme. Le nombre maximal d'évaluations est de 5E+5, ce qui correspond à 100 changements dans la fonction objectif à chaque exécution de l'algorithme. Les valeurs d'*offline error* et d'écart-type ainsi obtenues, pour chaque variante de MLSDO, sont données dans le tableau 3.4. Dans ce tableau, les variantes sont ordonnées de la meilleure à la moins bonne.

Pour ces variantes simplifiées, dans $\text{MLSDO}_{\text{noExcl}}$, la détection d'agents situés dans le rayon d'exclusion r_e d'un agent n'est pas effectuée. De ce fait, plusieurs agents peuvent explorer une même zone de l'espace de recherche sans avoir à être repositionnés. Dans $\text{MLSDO}_{\text{randInit}}$, l'heuristique de la procédure 3.7 n'est pas utilisée. Le rôle de cette heuristique est de créer une position initiale pour un agent qui soit éloignée des zones déjà explorées de l'espace de recherche. Ainsi, dans $\text{MLSDO}_{\text{randInit}}$, la recherche locale d'un agent est initialisée, ou réinitialisée, aléatoirement, selon une distribution uniforme, dans l'espace de recherche. Dans $\text{MLSDO}_{\text{noCDPA}}$, le produit scalaire cumulé n'est pas utilisé pour adapter R. L'adaptation du pas d'un agent consiste alors uniquement à réduire R, lorsque l'agent ne trouve aucune solution voisine qui

Variante	***Offline error***	**Description succincte**
$\text{MLSDO}_{\text{noCDPA}}$	$0{,}35 \pm 0{,}07$	sans utiliser le produit scalaire cumulé pour adapter R
MLSDO	$0{,}36 \pm 0{,}07$	l'algorithme original
$\text{MLSDO}_{\text{noExcl}}$	$0{,}39 \pm 0{,}08$	sans utiliser de rayon d'exclusion
$\text{MLSDO}_{\text{no}\,\delta_{pl}}$	$0{,}55 \pm 0{,}21$	sans utiliser le critère de stagnation de faible précision
$\text{MLSDO}_{\text{randInit}}$	$0{,}58 \pm 0{,}23$	en engendrant aléatoirement, selon une distribution uniforme, les solutions initiales des agents
$\text{MLSDO}_{\text{no}\,r_l}$	$0{,}82 \pm 0{,}08$	en utilisant r_e, au lieu de r_l, comme pas initial des agents de suivi
$\text{MLSDO}_{\text{no}\,\delta_{ph}}$	$1{,}08 \pm 0{,}10$	sans utiliser le critère de stagnation de haute précision
$\text{MLSDO}_{\text{noDom}}$	$1{,}32 \pm 0{,}41$	sans retirer les optima locaux « dominés » de l'archive A_m
$\text{MLSDO}_{\text{noTrack}}$	$3{,}95 \pm 0{,}24$	sans créer d'agents de suivi
$\text{MLSDO}_{\text{noArch}}$	$7{,}17 \pm 0{,}35$	sans archiver les optima locaux trouvés par les agents

TABLEAU 3.4 – *Offline error* sur MPB pour chaque variante simplifiée de MLSDO.

soit meilleure que $\vec{S}_c$. Dans $\text{MLSDO}_{\text{no}\,r_l}$, le pas initial d'un agent de suivi n'est pas égal à r_l mais à r_e, comme pour un agent d'exploration. Dans $\text{MLSDO}_{\text{noArch}}$, l'archivage des optima locaux trouvés par les agents n'est pas utilisé. De ce fait, l'archive A_m reste vide tout au long de l'exécution de MLSDO. Dans $\text{MLSDO}_{\text{noDom}}$, la procédure 2.7 n'est pas utilisée. Les optima locaux « dominés » par un autre ne sont donc pas retirés de l'archive A_m. Dans $\text{MLSDO}_{\text{no}\,\delta_{pl}}$, δ_{pl} est remplacé par δ_{ph}. La précision du critère de stagnation d'un agent est alors toujours égale à δ_{ph} (le paramètre δ_{pl} n'est pas utilisé dans cette variante). Le remplacement inverse est effectué dans $\text{MLSDO}_{\text{no}\,\delta_{ph}}$, où le paramètre δ_{ph} n'est pas utilisé. Enfin, dans $\text{MLSDO}_{\text{noTrack}}$, aucun agent n'est créé pour suivre les optima locaux archivés, lorsqu'un changement est détecté dans la fonction objectif (le paramètre n_c n'est pas utilisé).

Dans le tableau 3.4, nous remarquons que les variantes $\text{MLSDO}_{\text{noTrack}}$ et $\text{MLSDO}_{\text{noArch}}$ obtiennent, de loin, les plus mauvais résultats. Nous pouvons ainsi conclure que les composants les plus critiques de MLSDO, en termes de performance, sont l'archivage des optima locaux et leur suivi par des agents dédiés, tout comme c'était le cas pour MADO.

Ensuite, en comparant les résultats obtenus par $\text{MLSDO}_{\text{no}\,\delta_{pl}}$ et $\text{MLSDO}_{\text{no}\,\delta_{ph}}$, nous pouvons

conclure que δ_{ph} a plus d'influence que δ_{pl} dans la préservation des performances de MLSDO. Néanmoins, l'utilisation combinée de ces deux niveaux de précision permet d'améliorer davantage, et de façon significative, les performances de MLSDO.

Les résultats obtenus par $\text{MLSDO}_{\text{randInit}}$, $\text{MLSDO}_{\text{no}\,r_l}$ et $\text{MLSDO}_{\text{noDom}}$ montrent qu'une génération « intelligente » des solutions initiales des agents, tout comme l'utilisation d'un pas initial r_l dédié aux agents de suivi, et le retrait des optima dominés de l'archive des optima locaux, sont des stratégies importantes de MLSDO.

Enfin, nous avons appliqué le test statistique de Kruskal-Wallis [Kruskal & Wallis, 1952] sur les résultats obtenus par les trois premières variantes de MLSDO dans ce tableau. Ce test indique, pour un niveau de confiance de 99%, qu'il existe une différence significative entre les performances d'au moins deux variantes parmi les trois. Nous utilisons ensuite le test de Tukey-Kramer [Tukey, 1994; Kramer, 1957] pour déterminer, parmi ces trois variantes, celles qui diffèrent des autres, en terme d'*offline error*. Ce test ne permet pas de mettre en évidence une différence significative entre les performances de $\text{MLSDO}_{\text{noCDPA}}$ et de l'algorithme MLSDO original, que le niveau de confiance utilisé soit de 99% ou de 95%. Néanmoins, il indique, pour un niveau de confiance de 99%, que les performances de $\text{MLSDO}_{\text{noExcl}}$ sont significativement différentes de celles des deux autres variantes testées.

Nous pouvons alors conclure que tous les composants de MLSDO ainsi testés, à l'exception de l'utilisation du produit scalaire cumulé dans l'adaptation du pas d'un agent, sont utiles pour obtenir de bonnes performances sur MPB. Néanmoins, même si l'utilisation du produit scalaire cumulé n'est pas utile pour MPB, elle peut améliorer grandement les performances de MLSDO pour d'autres problèmes d'optimisation dynamique. Pour le confirmer, nous avons comparé les résultats obtenus par MLSDO et par sa variante $\text{MLSDO}_{\text{noCDPA}}$ sur le problème F_3 du jeu de tests GDBG (décrit au paragraphe 1.4.2.2). Ces résultats sont présentés dans le tableau 3.5. Comme nous le constatons, les performances de MLSDO sont significativement meilleures que celles de $\text{MLSDO}_{\text{noCDPA}}$ pour ce problème.

3.3.5 Analyse de convergence et comparaison sur MPB

La convergence de MLSDO est étudiée à la figure 3.5, en utilisant l'exécution obtenant une performance médiane parmi 100 exécutions de l'algorithme sur MPB (scénario 2). Etant donné le caractère stochastique de MLSDO, le choix de l'exécution de performance médiane permet de révéler le comportement moyen de l'algorithme. Sur cette figure, l'axe y correspond aux 40 premiers *time spans*, l'axe x correspond aux 800 premières évaluations de chaque *time span* avec une granularité de 20 évaluations, et l'axe z correspond à la valeur de l'erreur relative $\frac{f_y^* - f_{yx}^*}{f_y^*}$, en utilisant les notations de l'équation 1.4.2. Pour plus de clarté, nous utilisons une

Type de changements	$\mathbf{mark_{max}}$	$\mathbf{mark_{pct}}$	
		MLSDO	$\text{MLSDO}_{\text{noCDPA}}$
T_1	2,4	1,913	1,629
T_2	2,4	0,461	0,033
T_3	2,4	0,821	0,085
T_4	2,4	1,250	0,878
T_5	2,4	0,652	0,070
T_6	2,4	0,920	0,534
T_7	1,6	0,724	0,186
Score total pour F_3	16,0	6,74	3,41

TABLEAU 3.5 – Comparaison entre MLSDO et sa variante $\text{MLSDO}_{\text{noCDPA}}$ sur le problème F_3 de GDBG.

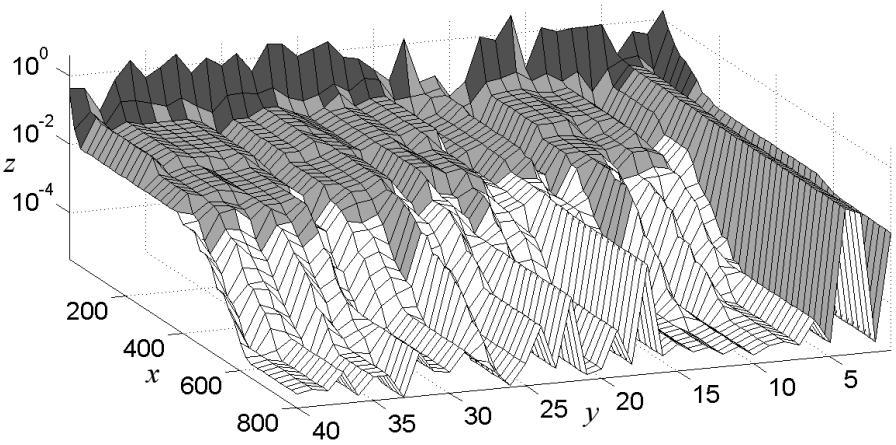


FIGURE 3.5 – Erreur relative en utilisant MPB. L'axe x correspond aux 800 premières évaluations de chaque *time span*. L'axe y correspond aux 40 premiers *time spans*. L'axe z correspond à la valeur de l'erreur relative, en utilisant une échelle logarithmique.

échelle logarithmique pour l'axe z.

Nous constatons sur cette figure que la convergence de MLSDO est très rapide. Comme pour MADO, les plus grandes valeurs d'erreur sont obtenues au premier *time span*, car MLSDO n'a pas encore détecté et archivé les optima locaux. Une fois ces derniers archivés, l'algorithme peut suivre ces optima, sans avoir besoin de les redétecter. La convergence de MLSDO s'en trouve alors améliorée.

L'évolution moyenne de l'erreur relative sur tous les *time spans* est illustrée à la figure 3.6. Sur cette figure, x correspond aux 5000 évaluations d'un *time span*, et $\bar{f}^*_x$ est la valeur moyenne de f^*_{jx} pour $j = 1,...,N_c$ (en utilisant les notations de l'équation 1.4.2). Pour plus de clarté, nous utilisons une échelle logarithmique pour l'axe de f^*_{jx}.

Nous constatons sur cette figure qu'il suffit de 18 évaluations pour que MLSDO obtienne une erreur relative moyenne inférieure à 10^{-1}, et qu'il lui en suffit 420 pour obtenir une erreur

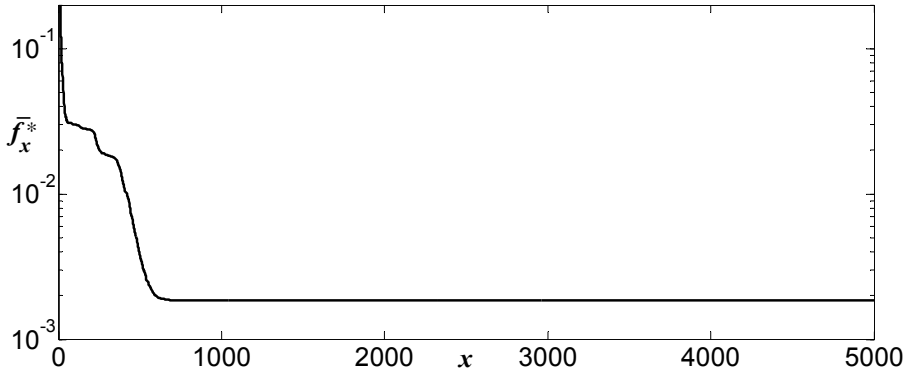


FIGURE 3.6 – Erreur relative moyenne en utilisant MPB.

Algorithme	***Offline error***
Moser & Chiong, 2010	$0{,}25 \pm 0{,}08$
MLSDO	$0{,}35 \pm 0{,}06$
Novoa et al., 2009	$0{,}40 \pm 0{,}04$
MADO	$0{,}59 \pm 0{,}10$
Moser & Hendtlass, 2007	$0{,}66 \pm 0{,}20$
Yang & Li, 2010	$1{,}06 \pm 0{,}24$
Liu et al., 2010	$1{,}31 \pm 0{,}06$
Lung & Dumitrescu, 2007	$1{,}38 \pm 0{,}02$
Bird & Li, 2007	$1{,}50 \pm 0{,}08$
Lung & Dumitrescu, 2008	$1{,}53 \pm 0{,}01$
Blackwell & Branke, 2006	$1{,}72 \pm 0{,}06$
Mendes & Mohais, 2005	$1{,}75 \pm 0{,}03$
Li et al., 2006	$1{,}93 \pm 0{,}06$
Blackwell & Branke, 2004	$2{,}16 \pm 0{,}06$
Parrott & Li, 2006	$2{,}51 \pm 0{,}09$
Du & Li, 2008	$4{,}02 \pm 0{,}56$

TABLEAU 3.6 – Classement des algorithmes en compétition sur MPB.

relative moyenne inférieure à 10^{-2}.

La comparaison, sur MPB, de MLSDO avec les autres algorithmes d'optimisation dynamique les plus compétitifs de la littérature est présentée dans le tableau 3.6. La comparaison est effectuée en utilisant l'*offline error* moyennée sur 50 exécutions. Les algorithmes sont classés du meilleur au moins bon, en termes d'*offline error*. Nous constatons que MLSDO est à la deuxième place de ce classement. Comme pour MADO, les bonnes performances de MLSDO s'expliquent par sa convergence rapide vers les optima locaux, rendue possible par l'utilisation de recherches locales. Les algorithmes évolutionnaires, ou inspirés de l'optimisation par essaim particulaire, nécessitent plus d'évaluations pour atteindre un optimum local.

Ensuite, pour comparer les performances de MLSDO avec les autres algorithmes d'optimisa-

Algorithme	***Offline error***
MLSDO	$13{,}98 \pm 2{,}35$
Moser & Chiong, 2010	$16{,}50 \pm 5{,}40$
Lung & Dumitrescu, 2007	$24{,}60 \pm 0{,}25$
MADO	$34{,}64 \pm 2{,}72$
Moser & Hendtlass, 2007	$480{,}5 \pm 70{,}1$

TABLEAU 3.7 – Classement des algorithmes en compétition sur MPB en utilisant $d = 100$.

tion dynamique en grande dimension, la même comparaison (MPB, scénario 2) est effectuée avec $d = 100$, et présentée dans le tableau 3.7. Néanmoins, les résultats numériques avec $d = 100$ de plusieurs algorithmes du tableau 3.6 ne sont pas disponibles (ils ne sont donc pas inclus dans cette comparaison). Nous avons relevé les résultats de l'algorithme proposé par [Lung & Dumitrescu, 2007] dans l'article de [Moser & Chiong, 2010].

Nous constatons que MLSDO est en tête de ce classement dans le cas de la dimension 100, en termes d'*offline error*, alors qu'il était second au classement utilisant $d = 5$ (tableau 3.6) : MLSDO est plus performant que l'algorithme de [Moser & Chiong, 2010] pour $d = 100$, alors que c'était le contraire pour $d = 5$. Cela signifie que les performances de MLSDO restent relativement bonnes quand on augmente la dimension du problème.

Enfin, nous comparons les performances de MLSDO à celles des autres algorithmes sur MPB (scénario 2), en utilisant de plus grandes sévérités des changements : $s = 1, 2, ..., 6$. Cette comparaison est présentée dans le tableau 3.8. Nous avons relevé les résultats de l'algorithme proposé par [Li et al., 2006] dans l'article de [Liu et al., 2010], et ceux de [Parrott & Li, 2006] dans l'article de [Bird & Li, 2007].

Il est normal que les optima locaux soient de plus en plus difficiles à suivre lorsque la sévérité des changements croît. C'est pourquoi les performances de tous les algorithmes se dégradent lorsque l'on augmente la valeur de s. Néanmoins, nous constatons que les performances des quatre algorithmes suivants ne se dégradent pas autant que celles de MLSDO : [Moser & Chiong, 2010] ; [Moser & Hendtlass, 2007] ; [Yang & Li, 2010] et [Lung & Dumitrescu, 2008]. Ainsi, bien que MLSDO soit à la deuxième place du classement pour $s = 1, 2, 3$, il est classé à la troisième, quatrième et cinquième place pour $s = 4, 5, 6$, respectivement.

Cette dégradation des performances de MLSDO s'explique par le fait que le pas initial r_l des agents de suivi doit être adapté à la sévérité des changements, comme indiqué au paragraphe 3.3.1. Ainsi, de meilleurs résultats peuvent être obtenus en adaptant la valeur de r_l à la sévérité des changements, comme nous le constatons dans le tableau 3.9.

Algorithme	***Offline error***					
	$s = 1$	$s = 2$	$s = 3$	$s = 4$	$s = 5$	$s = 6$
Moser & Chiong, 2010	$0{,}25 \pm 0{,}08$	$0{,}47 \pm 0{,}12$	$0{,}49 \pm 0{,}12$	$0{,}53 \pm 0{,}13$	$0{,}65 \pm 0{,}19$	$0{,}77 \pm 0{,}24$
MLSDO	$0{,}35 \pm 0{,}06$	$0{,}60 \pm 0{,}08$	$0{,}91 \pm 0{,}10$	$1{,}23 \pm 0{,}10$	$1{,}62 \pm 0{,}13$	$2{,}00 \pm 0{,}20$
MADO	$0{,}59 \pm 0{,}10$	$0{,}87 \pm 0{,}12$	$1{,}18 \pm 0{,}13$	$1{,}49 \pm 0{,}13$	$1{,}86 \pm 0{,}17$	$2{,}32 \pm 0{,}18$
Moser & Hendtlass, 2007	$0{,}66 \pm 0{,}20$	$0{,}86 \pm 0{,}21$	$0{,}94 \pm 0{,}22$	$0{,}97 \pm 0{,}21$	$1{,}05 \pm 0{,}21$	$1{,}09 \pm 0{,}22$
Yang & Li, 2010	$1{,}06 \pm 0{,}24$	$1{,}17 \pm 0{,}22$	$1{,}36 \pm 0{,}28$	$1{,}38 \pm 0{,}29$	$1{,}58 \pm 0{,}32$	$1{,}53 \pm 0{,}29$
Liu et al., 2010	$1{,}31 \pm 0{,}06$	$1{,}98 \pm 0{,}06$	$2{,}21 \pm 0{,}06$	$2{,}61 \pm 0{,}11$	$3{,}20 \pm 0{,}13$	$3{,}93 \pm 0{,}14$
Lung & Dumitrescu, 2007	$1{,}38 \pm 0{,}02$	$1{,}78 \pm 0{,}02$	$2{,}03 \pm 0{,}03$	$2{,}23 \pm 0{,}05$	$2{,}52 \pm 0{,}06$	$2{,}74 \pm 0{,}10$
Bird & Li, 2007	$1{,}50 \pm 0{,}08$	$1{,}87 \pm 0{,}05$	$2{,}40 \pm 0{,}08$	$2{,}90 \pm 0{,}08$	$3{,}25 \pm 0{,}09$	$3{,}86 \pm 0{,}11$
Lung & Dumitrescu, 2008	$1{,}53 \pm 0{,}01$	$1{,}57 \pm 0{,}01$	$1{,}67 \pm 0{,}01$	$1{,}72 \pm 0{,}03$	$1{,}78 \pm 0{,}06$	$1{,}79 \pm 0{,}03$
Blackwell & Branke, 2006	$1{,}75 \pm 0{,}06$	$2{,}40 \pm 0{,}06$	$3{,}00 \pm 0{,}06$	$3{,}59 \pm 0{,}10$	$4{,}24 \pm 0{,}10$	$4{,}79 \pm 0{,}10$
Li et al., 2006	$1{,}93 \pm 0{,}08$	$2{,}25 \pm 0{,}09$	$2{,}74 \pm 0{,}09$	$3{,}05 \pm 0{,}10$	$3{,}24 \pm 0{,}11$	$4{,}95 \pm 0{,}13$
Parrott & Li, 2006	$2{,}51 \pm 0{,}09$	$3{,}78 \pm 0{,}09$	$4{,}96 \pm 0{,}12$	$5{,}56 \pm 0{,}13$	$6{,}76 \pm 0{,}15$	$7{,}68 \pm 0{,}16$

TABLEAU 3.8 – Performances des algorithmes en compétition sur MPB en utilisant $s = 1, 2, ..., 6$.

Valeur de s	1	2	3	4	5	6
Valeur de r_l	0,005	0,010	0,015	0,019	0,023	0,027
Offline error	$0{,}35 \pm 0{,}06$	$0{,}52 \pm 0{,}07$	$0{,}66 \pm 0{,}09$	$0{,}82 \pm 0{,}07$	$0{,}99 \pm 0{,}08$	$1{,}33 \pm 0{,}16$

TABLEAU 3.9 – Performances de MLSDO sur MPB en utilisant $s = 1, 2, ..., 6$. Le paramètre r_l de MLSDO est adapté empiriquement à la valeur de s utilisée. Les autres paramètres de MLSDO restent à leurs valeurs définies au paragraphe 3.3.1.

3.3.6 Analyse de convergence et comparaison sur GDBG

Les résultats de MLSDO obtenus en utilisant GDBG sont présentés dans le tableau 3.10. Un exemple de la convergence de MLSDO sur le problème F_1 à 10 pics est illustré à la figure 3.7, où t est le nombre d'évaluations effectuées depuis le début de l'exécution de MLSDO. Sur cette figure, pour chaque type de changements T_k, l'erreur relative $r_i(t)$ est donnée par $r_i(t) + k - 1$, avec $0 \leq r_i(t) \leq 1$, i étant l'exécution de performance médiane (parmi 20) et $k = 1, 2, ..., 7$.

Nous constatons que les courbes montrant l'évolution de l'erreur relative sont très abruptes, pour tous les types de changements. Cela signifie que MLSDO nécessite peu d'évaluations pour converger vers un meilleur optimum local. Pour mettre en évidence cette convergence rapide, nous réalisons, à la figure 3.7, un agrandissement au niveau du premier changement dans la fonction objectif, pour tous les types de changements.

Sur cette figure, pour les types de changements T_1, T_4 et T_6, l'erreur relative atteint une valeur proche de 1 pour chaque *time span*. Cela signifie que l'optimum global est trouvé durant chaque *time span*, pour tous les types de changements. Le nombre moyen d'évaluations nécessaires pour atteindre une erreur relative de 0,99 au cours d'un *time span* est de 2511 pour

Fonction	**Changements**	**Avg$_{best}$**	**Avg$_{mean}$**	**Avg$_{worst}$**	**STD**	**mark$_{max}$**	**mark$_{pct}$**
F_1 (10 pics)	T_1	4,80E−04	1,23E−02	3,91E−01	7,47E−02	1,5	1,493
F_1 (10 pics)	T_2	4,83E−04	1,03E+00	2,02E+01	4,14E+00	1,5	1,455
F_1 (10 pics)	T_3	4,66E−04	1,68E+00	1,98E+01	5,15E+00	1,5	1,428
F_1 (10 pics)	T_4	5,51E−04	2,36E−02	5,88E−01	1,26E−01	1,5	1,496
F_1 (10 pics)	T_5	5,00E−04	5,64E−01	1,23E+01	2,14E+00	1,5	1,464
F_1 (10 pics)	T_6	4,85E−04	4,96E−02	2,45E+00	1,11E+00	1,5	1,481
F_1 (10 pics)	T_7	3,94E−04	1,02E+00	1,58E+01	3,80E+00	1,0	0,966
Score total pour F_1 (10 pics)						10,0	9,78
F_1 (50 pics)	T_1	4,78E−04	7,22E−02	1,04E+00	2,13E−01	1,5	1,490
F_1 (50 pics)	T_2	5,04E−04	1,75E+00	1,53E+01	3,43E+00	1,5	1,433
F_1 (50 pics)	T_3	5,13E−04	3,33E+00	1,91E+01	5,39E+00	1,5	1,386
F_1 (50 pics)	T_4	5,36E−04	2,38E−01	9,67E−01	2,83E−01	1,5	1,481
F_1 (50 pics)	T_5	5,53E−04	5,15E−01	4,49E+00	9,01E−01	1,5	1,469
F_1 (50 pics)	T_6	5,56E−04	2,53E−01	1,31E+00	3,41E−01	1,5	1,473
F_1 (50 pics)	T_7	3,95E−04	2,71E+00	1,91E+01	4,91E+00	1,0	0,934
Score total pour F_1 (50 pics)						10,0	9,67
F_2	T_1	4,62E−04	2,02E−01	5,33E+00	1,38E+00	2,4	2,277
F_2	T_2	5,10E−04	1,11E+01	2,11E+02	5,56E+01	2,4	1,876
F_2	T_3	5,30E−04	6,02E+00	1,83E+02	3,38E+01	2,4	1,940
F_2	T_4	5,12E−04	3,02E−02	1,02E+00	2,58E−01	2,4	2,348
F_2	T_5	6,02E−04	1,77E+01	3,13E+02	5,90E+01	2,4	1,762
F_2	T_6	4,70E−04	1,27E−01	4,30E+00	8,40E−01	2,4	2,258
F_2	T_7	4,23E−04	6,40E−01	1,37E+01	3,08E+00	1,6	1,469
Score total pour F_2						16,0	13,93
F_3	T_1	4,86E−04	3,19E+00	8,59E+01	2,56E+01	2,4	1,913
F_3	T_2	7,18E−04	3,91E+02	9,20E+02	4,03E+02	2,4	0,461
F_3	T_3	5,71E−04	2,04E+02	8,86E+02	3,34E+02	2,4	0,821
F_3	T_4	5,15E−04	1,68E+02	1,07E+03	3,38E+02	2,4	1,250
F_3	T_5	1,25E−02	3,45E+02	8,74E+02	3,88E+02	2,4	0,652
F_3	T_6	5,27E−04	2,79E+02	1,26E+03	4,19E+02	2,4	0,920
F_3	T_7	4,23E−04	9,39E+01	8,22E+02	2,28E+02	1,6	0,724
Score total pour F_3						16,0	6,74
F_4	T_1	4,81E−04	2,18E−01	4,24E+00	1,48E+00	2,4	2,261
F_4	T_2	4,74E−04	1,63E+01	2,88E+02	7,72E+01	2,4	1,808
F_4	T_3	5,39E−04	7,17E+00	1,74E+02	3,72E+01	2,4	1,863
F_4	T_4	4,71E−04	1,19E−01	2,44E+00	7,58E−01	2,4	2,273
F_4	T_5	5,39E−04	2,49E+01	4,06E+02	8,08E+01	2,4	1,681
F_4	T_6	4,77E−04	2,21E−01	6,28E+00	1,10E+00	2,4	2,226
F_4	T_7	4,11E−04	1,07E+00	1,88E+01	4,07E+00	1,6	1,427
Score total pour F_4						16,0	13,54
F_5	T_1	4,73E−04	2,89E−02	7,48E−01	1,45E−01	2,4	2,280
F_5	T_2	4,62E−04	3,63E−02	8,00E−01	1,52E−01	2,4	2,276
F_5	T_3	4,38E−04	3,58E−02	9,52E−01	1,66E−01	2,4	2,278
F_5	T_4	5,06E−04	1,54E−02	5,91E−01	1,10E−01	2,4	2,288
F_5	T_5	4,63E−04	5,15E−02	1,34E+00	3,09E−01	2,4	2,268
F_5	T_6	4,70E−04	2,83E−02	7,25E−01	1,30E−01	2,4	2,272
F_5	T_7	3,90E−04	2,92E−02	8,98E−01	1,51E−01	1,6	1,520
Score total pour F_5						16,0	15,18
F_6	T_1	5,04E−04	3,06E+00	2,37E+01	6,30E+00	2,4	1,945
F_6	T_2	5,34E−04	5,16E+00	4,01E+01	9,36E+00	2,4	1,792
F_6	T_3	5,60E−04	5,97E+00	2,64E+01	9,41E+00	2,4	1,763
F_6	T_4	5,26E−04	2,64E+00	1,77E+01	5,02E+00	2,4	1,891
F_6	T_5	5,51E−04	6,50E+00	5,81E+01	1,67E+01	2,4	1,935
F_6	T_6	5,10E−04	2,99E+00	1,99E+01	5,35E+00	2,4	1,832
F_6	T_7	4,99E−04	3,83E+00	2,00E+01	7,39E+00	1,6	1,277
Score total pour F_6						16,0	12,44
Score total *op*						100,0	**81,28**

TABLEAU 3.10 – Performances de MLSDO sur GDBG.

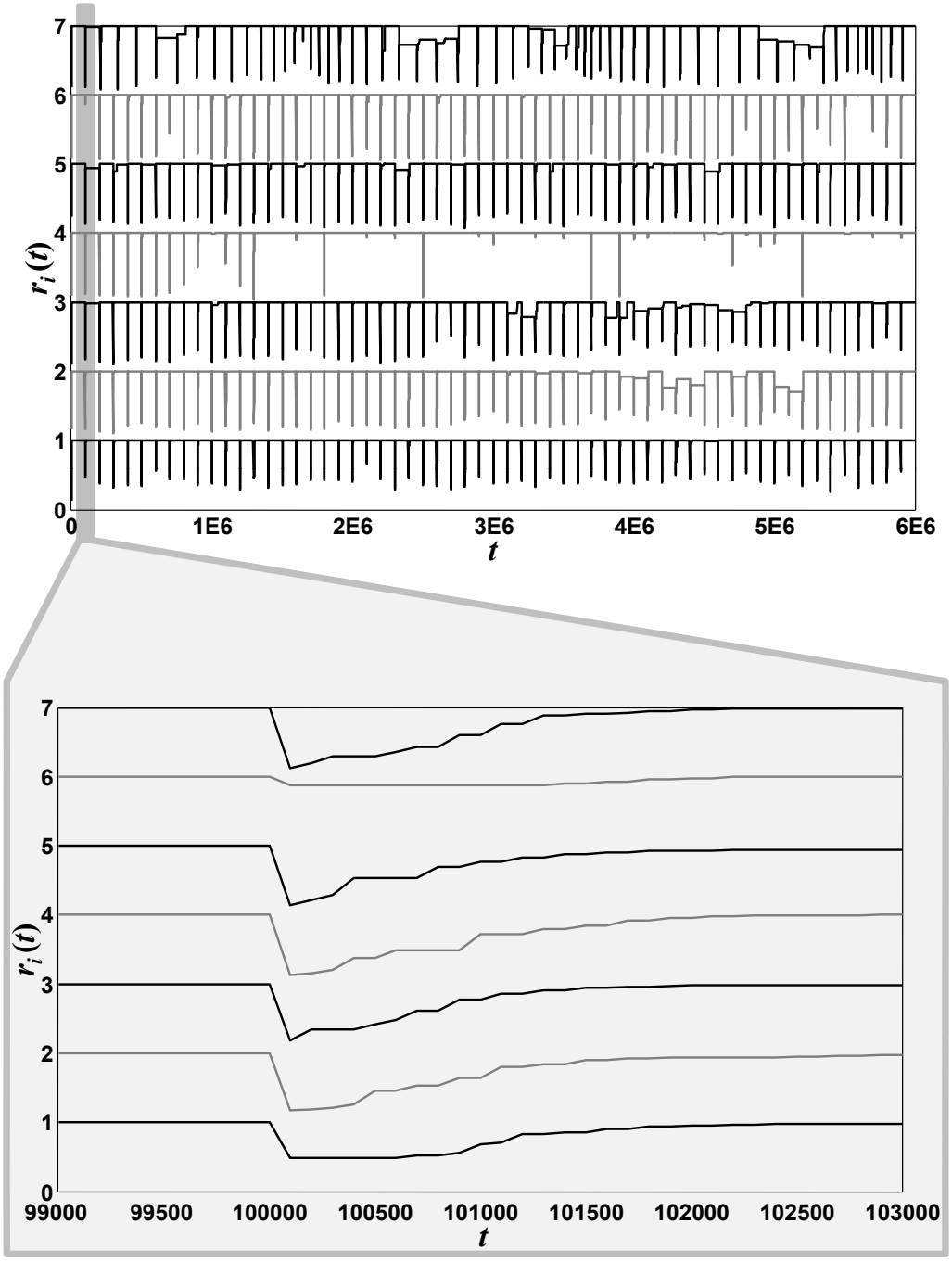


FIGURE 3.7 – Graphe de convergence de MLSDO pour le problème F_1 à 10 pics de GDBG.

T_1, 1596 pour T_4 et 5523 pour T_6. Ainsi, MLSDO converge rapidement vers l'optimum global pour ces types de changements. Pour les autres types de changements, nous constatons que l'optimum global est rapidement trouvé pour la plupart des *time spans*.

Nous constatons que MLSDO est capable de trouver de bonnes solutions pour tous les problèmes. Il n'y a que pour le problème F_3 que MLSDO ne parvient pas à trouver l'optimum assez rapidement. Il s'agit néanmoins d'un problème difficile, que tous les algorithmes concurrents ont du mal à résoudre. Le meilleur score obtenu par ces derniers sur le problème F_3 est de 5,15 [Brest et al., 2009], alors que MLSDO obtient un meilleur score de 6,74.

MLSDO nécessite peu d'évaluations par *time span* pour trouver une bonne solution. Cela représente un avantage majeur pour des problèmes d'optimisation dynamique pour lesquels les changements se produisent fréquemment.

Nous constatons, en comparant les résultats du tableau 3.10 entre eux, que MLSDO obtient les meilleures performances pour les problèmes F_1, à 10 et 50 pics, et pour le problème F_5. MLSDO obtient des performances similaires pour les problèmes F_2 et F_4, basés sur les fonctions Sphere et Griewank, respectivement.

Nous voyons dans ce tableau que MLSDO obtient les moins bonnes performances pour le type de changements T_7 (excepté pour le problème F_3). Cela est dû au fait que T_7 est le seul type de changements pour lequel la dimension de l'espace de recherche varie. En effet, cela accentue la difficulté, et tous les algorithmes concurrents [Brest et al., 2009; Yu & Suganthan, 2009; Li & Yang, 2009; Korosec & Silc, 2009; de França & Zuben, 2009] rencontrent des difficultés à résoudre les problèmes utilisant ce type de changements.

Pour le problème F_3, nous constatons dans le tableau 3.10 que les moins bonnes performances de MLSDO sont obtenues pour le type de changements T_2. Cela peut s'expliquer par le fait que les optima locaux effectuent de grands déplacements avec ce type de changements, ce qui induit une modification majeure du paysage de la fonction objectif de ce problème, déjà difficile.

Enfin, nous effectuons une comparaison, sur GDBG, de MLSDO avec les autres algorithmes d'optimisation dynamique en compétition sur ce jeu de tests. Cette comparaison est présentée dans la figure 3.8. Les algorithmes sont ordonnés selon leur score total. Nous constatons que MLSDO est largement en tête du classement sur ce jeu de tests.

3.4 Conclusion

Un nouvel algorithme d'optimisation dynamique, nommé MLSDO, a été présenté. Pour mettre au point cet algorithme, nous sommes partis de l'architecture de base de MADO et

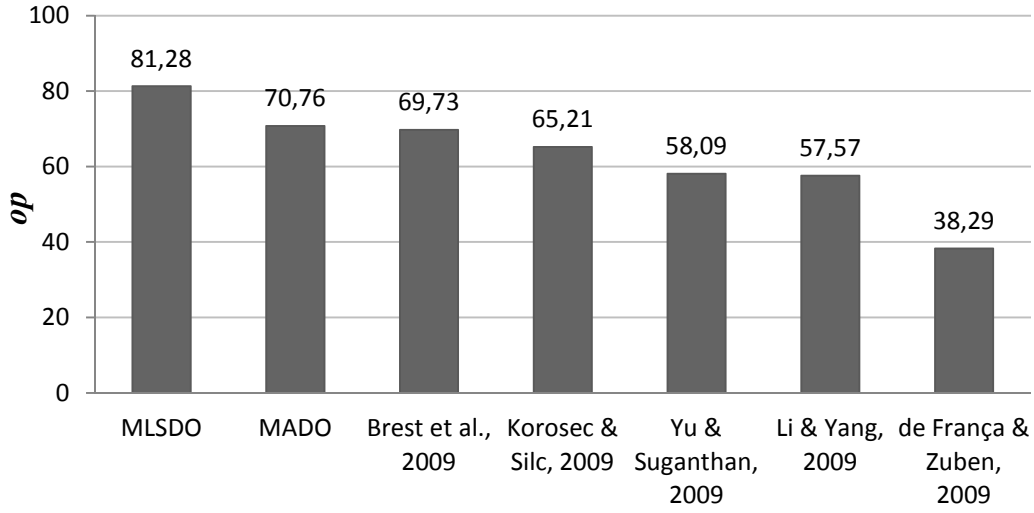


FIGURE 3.8 – Classement des algorithmes en compétition sur GDBG.

l'avons fait évoluer, d'une part en substituant aux éléments faibles des composants plus efficaces, d'autre part en introduisant des stratégies plus performantes, notamment dans la recherche locale des agents. Ce nouvel algorithme obtient des résultats qui dépassent de manière très significative ceux obtenus par MADO sur les deux principaux jeux de tests en optimisation dynamique (MPB et GDBG). De plus, il se positionne parmi les algorithmes les plus performants sur ces jeux de tests : en tête du classement sur GDBG, et à la deuxième place sur MPB.

Dans les chapitres suivants, nous allons nous intéresser à l'application de MLSDO à des problèmes réels relevant du traitement d'images médicales, thématique de recherche prépondérante du laboratoire LiSSi, au sein duquel s'est déroulée cette thèse. Comme pour l'évaluation de MLSDO sur des jeux de tests, les résultats obtenus seront présentés et commentés, et des pistes d'amélioration possibles seront proposées.

Chapitre Quatre

Application de CMADO à la segmentation de séquences d'images médicales

4.1 Introduction

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

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

Plusieurs approches de segmentation d'images existent dans la littérature [Nakib, 2007]. Dans le cadre de nos travaux, nous nous sommes limités à l'approche par seuillage d'images. Le seuillage d'image est une méthode de segmentation supervisée, c'est-à-dire que le nombre de régions et leurs propriétés sont fournis au préalable par l'utilisateur. La segmentation est effectuée en déterminant, pour chaque pixel, la classe dont les propriétés se rapprochent le plus de celles observées en ce pixel. La technique de seuillage est basée sur l'hypothèse que les différentes régions de l'image peuvent être différenciées par leurs niveaux de gris. Cette méthode repose donc sur l'utilisation de l'histogramme de l'image traitée. Le seuillage de l'image en N

classes revient à trouver les $N-1$ seuils qui vont partitionner l'histogramme en N zones. Le problème de seuillage est défini plus précisément dans la deuxième section de ce chapitre.

Dans ce chapitre, le problème de segmentation sera abordé au sein d'un problème de quantification du mouvement d'un ventricule cérébral. Tout d'abord, nous présenterons le problème et la méthode proposée pour quantifier ces mouvements. Cette méthode étant basée sur la segmentation de séquences d'images ciné-IRM du cerveau, nous nous focaliserons ensuite sur le problème de segmentation propre à cette méthode. Ce problème est résolu à l'aide de CMADO [Lepagnot et al., 2011b]. Le critère de segmentation que nous proposons d'utiliser repose sur l'approximation de l'histogramme par une somme de fonctions gaussiennes. Approcher l'histogramme par une somme de gaussiennes permet de disposer d'une expression analytique de l'histogramme, et donc de calculer plus facilement les seuils optimaux.

L'intérêt d'utiliser un algorithme d'optimisation dynamique, tel que CMADO, réside dans la réduction du temps de calcul nécessaire à la segmentation des images de la séquence. En effet, les variations entre deux images successives d'une séquence ciné-IRM étant généralement faibles, CMADO permet d'exploiter cette information afin d'accélérer le processus de segmentation. A notre connaissance, ce travail est innovant, en ce sens que l'approche de l'optimisation dynamique n'avait jamais été explorée pour segmenter une séquence d'images.

4.2 Segmentation d'images par seuillage

4.2.1 Formulation du problème

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

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

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

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

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

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

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

4.2.2 Méthodes de seuillage

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

Les deux méthodes non-paramétriques de référence sont la méthode d'Otsu [Otsu, 1979] et la méthode de Kapur et al. [Kapur et al., 1985]. Beaucoup de techniques parues ensuite sont des variantes de ces deux méthodes. La méthode d'Otsu repose sur la maximisation de la *variance interclasse*. Celle de Kapur et al. est basée sur la maximisation de l'entropie de Shannon. Cette méthode a ouvert le champ à la définition de nombreuses mesures d'information pour mieux segmenter les images, telles que l'entropie de Tsallis ou l'entropie de Renyi [Sezgin & Sankur, 2004].

Les méthodes paramétriques de seuillage supposent que les différentes classes de l'image suivent une *fonction de densité de probabilité* (fdp) prédéfinie. L'histogramme est alors approché par une somme de fdp et les seuils sont choisis aux intersections de celles-ci. Généralement, ces fdp sont supposées suivre un modèle gaussien. Le problème se décompose ici en deux étapes : estimer les paramètres des fdp et déterminer les seuils de segmentation. La plupart des méthodes existantes sont présentées dans [Sahoo et al., 1988], [Sezgin & Sankur, 2004], [Gonzalez & Woods, 2007] et [Cuevas et al., 2010].

Dans ce chapitre, nous nous intéressons à une méthode de cette catégorie pour segmenter des séquences d'images. L'histogramme est ainsi approché par une somme de fonctions gaussiennes. Une telle opération nous permet d'obtenir une expression analytique approchée de l'histogramme, et d'en déduire les seuils optimaux. L'application de cette méthode à une seule image consomme beaucoup de temps [Nakib et al., 2008], ce qui, dans le cadre de l'optimisation statique, empêche son application à des séquences d'images. D'où l'intérêt de l'optimisation dynamique pour résoudre ce problème.

4.2.3 Seuillage par apprentissage

La première étape permet d'approcher l'histogramme, noté h, d'une image quelconque codée sur une échelle de L niveaux de gris, par une somme de d fdp [Synder & Bilbro, 1990]. Dans notre cas, des fdp gaussiennes sont utilisées, et l'approximation de h est donnée par :

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

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

L'histogramme h est normalisé de la manière suivante :

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

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

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

Le jeu de paramètres Θ est donné par :

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

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

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

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

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

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

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

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

avec :

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

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

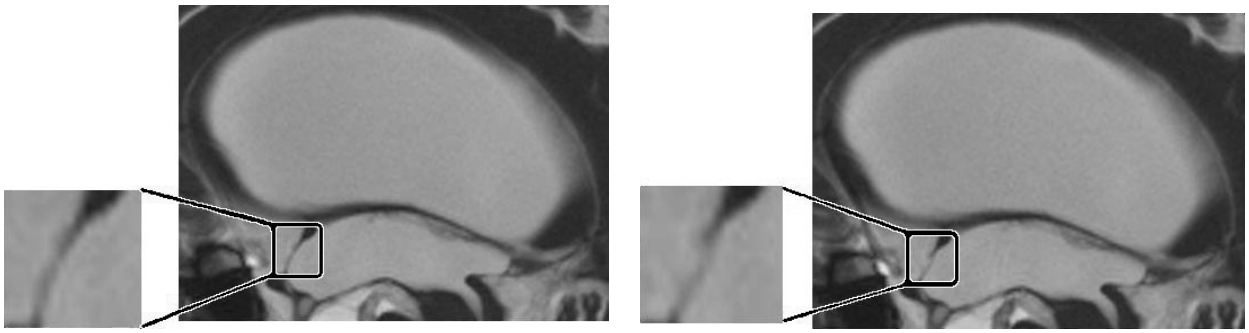
$$C = \mu_{i+1}^2 \; \sigma_i^2 - \mu_i^2 \; \sigma_{i+1}^2 + 4 \; \sigma_i^2 \; \sigma_{i+1}^2 \ln \left(\frac{\sigma_{i+1} \; P_i}{\sigma_i \; P_{i+1}} \right) \tag{4.2.11}$$

Cette expression a deux solutions possibles, mais seule l'une des deux est acceptable dans le cas présent [Kittler & Illingworth, 1986].

4.3 Application à la quantification de mouvements

Récemment, une nouvelle technique d'acquisition d'images cérébrales ciné-IRM a été mise en œuvre par le Professeur P. Decq (neurochirurgien) et le Docteur J. Hodel (neuroradiologue) au centre hospitalier Henri Mondor, dans le cadre de l'Antenne Analyse et Restauration du Mouvement (ParisTech-ENSAM CNRS 8005). Cette technique permet une bonne visualisation des mouvements des parois du troisième ventricule cérébral et des flux turbulents du liquide cérébro-spinal (LCS) au cours du cycle cardiaque.

Dans ce chapitre, nous nous intéressons à la quantification de ces mouvements. Cette quantification permet une meilleure compréhension du fonctionnement physiologique cérébral, et apporte une aide au diagnostic et à la décision thérapeutique. Dans notre cas, il s'agit de faciliter le diagnostic et d'assister les neurochirurgiens dans le traitement de l'hydrocéphalie.



(a) (b)

FIGURE 4.1 – Deux images extraites d'une séquence ciné-IRM cérébrale : (a) première image de la séquence, (b) sixième image de la séquence.

Paramètre	**Valeur**
temps de répétition / temps d'écho	78 ms / 3 ms
Nombre d'excitations	7
Angle de bascule	82°
Champ de vue	160 mm
Taille de la matrice	256×256
Epaisseur de coupe	2,5 mm
Temps d'acquisition	de 2 à 4 min
Phases durant l'intervalle R-R	20

TABLEAU 4.1 – Paramètres de l'acquisition IRM.

L'hydrocéphalie est une anomalie neurologique sévère, définie par l'augmentation du volume des espaces (ventricules cérébraux) contenant le LCS. Bien que certains symptômes puissent suggérer cette maladie, son diagnostic repose essentiellement sur l'étude d'images du cerveau.

Nous disposons de séquences d'images des parois du troisième ventricule cérébral (par exemple, la *lamina terminalis*) extraites de séquences de 20 images ciné-IRM cérébrales. Un exemple de deux images ainsi extraites est présenté à la figure 4.1. Notre jeu de données est donc constitué de séquences ciné-IRM de la *lamina terminalis* (correspondant à 80% d'un cycle cardiaque R-R ; plus de détails sont disponibles dans [Nakib et al., 2010]). Un exemple de séquence est illustré à la figure 4.2.

Ces données ont été obtenues en utilisant une machine 1.5-T MR (Siemens Avanto, Erlangen, Germany). Le protocole inclut une séquence *Cine True FISP MR* : un plan sagittal médian est défini à partir d'une coupe transversale allant du centre de la *lamina terminalis* à l'*aqueduc de Sylvius* [Hodel et al., 2009]. Les paramètres de l'acquisition sont donnés dans le tableau 4.1.

Plusieurs méthodes de quantification de mouvements cardiaques ont été proposées dans la littérature [Chenoune et al., 2005; Budoff et al., 2009; Sundar et al., 2009]. Cependant, ces

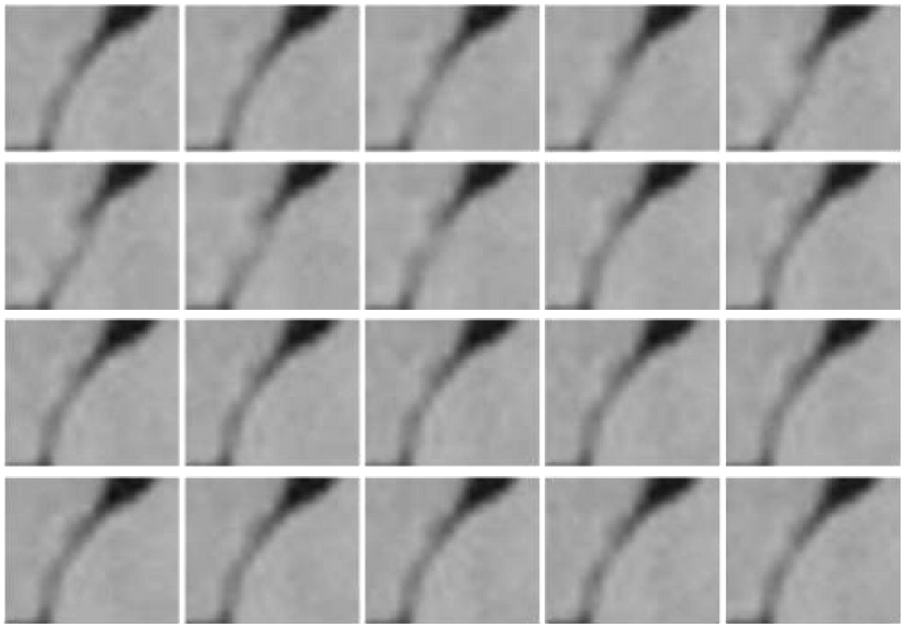


FIGURE 4.2 – Une séquence d'images ciné-IRM de la région d'intérêt (*lamina terminalis*).

méthodes ne peuvent pas être appliquées directement sur nos séquences, car l'amplitude des mouvements dans le cœur est beaucoup plus importante que celle des mouvements des parois du troisième ventricule cérébral. Du fait de l'apparition récente de la méthode d'acquisition, la seule technique actuellement disponible pour quantifier les mouvements de ce ventricule a été proposée dans [Nakib et al., 2010]. Cette méthode fonctionne en deux étapes : dans la première étape, les images de la séquence sont segmentées afin d'extraire les contours de la région d'intérêt, à différents instants du cycle cardiaque. Dans la deuxième étape, les informations fournies par ces contours sont combinées au moyen d'une procédure de recalage. Cette procédure de recalage nous permet de suivre les déplacements de chaque point du contour de la région d'intérêt au cours du temps, et de quantifier son déplacement maximum. Elle nous permet également d'obtenir une meilleure modélisation mathématique des déformations des parois du troisième ventricule.

Les deux étapes de cette méthode s'enchaînent selon l'organigramme de la figure 4.3.

Nous décrirons plus en détail la deuxième étape de cette méthode de quantification de mouvements au chapitre suivant. Dans ce chapitre, nous nous focalisons essentiellement sur la première étape. Nous proposons de segmenter les images de la séquence au moyen d'une méthode de seuillage, basée sur une approximation de l'histogramme par une somme de fonctions gaussiennes. La contribution principale de ce travail est de montrer l'intérêt d'utiliser un algorithme d'optimisation dynamique, CMADO, sur un problème de segmentation de séquences d'images tel que celui-ci.

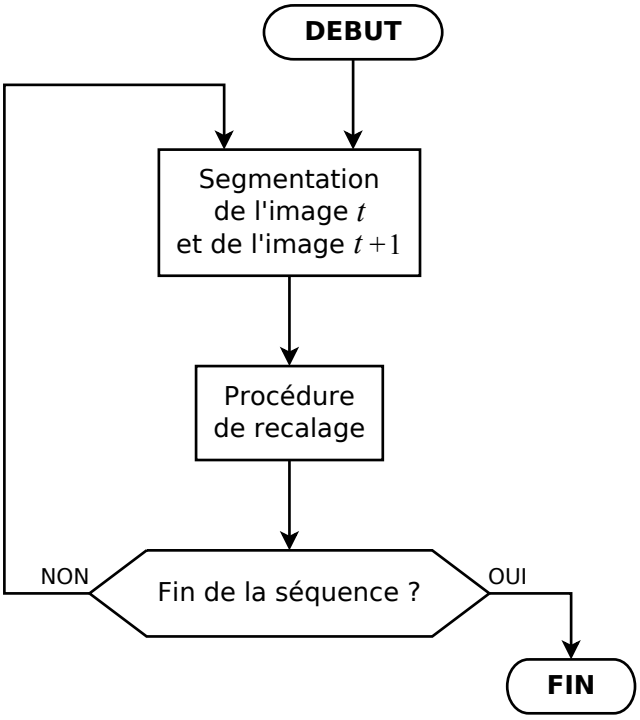


FIGURE 4.3 – Organigramme de la méthode de quantification de mouvements.

4.4 Segmentation de séquences d'images ciné-IRM

Afin de segmenter une séquence d'images, l'histogramme de chaque image de la séquence doit être approché. Pour cela, le critère de l'équation (4.2.4) doit être minimisé pour chaque image, et ce critère devient :

$$J(\Theta(t), t) = \sum_{i=0}^{L-1} |h_{norm}(i,t) - h_{approx}(\Theta(t), i)|^2 \tag{4.4.1}$$

où t est l'indice de l'image courante dans la séquence, $\Theta(t)$ est défini comme le jeu de paramètres Θ mais dépend ici de t, et $h_{norm}(i,t)$ est l'histogramme normalisé de l'image d'indice t.

La minimisation de ce critère est réalisée via CMADO, en utilisant la fonction objectif dynamique définie comme suit :

- Le critère (4.4.1) est minimisé pour l'image d'indice t.
- Lorsque la meilleure solution trouvée pour l'image t ne peut plus être améliorée (selon un critère de stagnation), alors l'histogramme de cette image est considéré comme étant correctement approché. Les seuils de segmentation de l'image t sont alors calculés, et le critère (4.4.1) est minimisé pour l'image d'indice $t+1$.
- Le critère de stagnation utilisé est satisfait lorsqu'aucune amélioration significative (supérieure à 1E−10) de la meilleure solution trouvée n'est observée pendant 5000 évaluations

successives de la fonction objectif.
– Ainsi, la fin de la segmentation d'une image et le début de la segmentation de la suivante constituent un changement dans la fonction objectif.

Les valeurs des paramètres de CMADO utilisées pour ce problème de segmentation sont indiquées dans le tableau 4.2. Ces valeurs ont été déterminées empiriquement, en les ajustant manuellement, de façon à minimiser le nombre d'évaluations nécessaires pour obtenir une segmentation optimale.

Paramètre	n_a	n_m	N	c_u	c_r	r_e	r_l	δ_t	δ_p
Valeur	2	3	5	0,5	0,8	0,1	0,001	8	1E−10

TABLEAU 4.2 – Paramétrage de CMADO.

4.5 Résultats et discussion

Nous présentons maintenant les résultats expérimentaux obtenus avec la méthode proposée. Les résultats de segmentation sont présentés à la figure 4.4. Par souci de concision, nous nous focalisons sur les trois premières et les trois dernières images d'une séquence. Sur cette figure, les images originales et leurs histogrammes sont illustrés en (a) et en (b), respectivement. Les histogrammes approchés sont illustrés en (c), et les images segmentées (en 3 classes) sont illustrées en (d). A la figure 4.4 (d), nous remarquons qu'en fusionnant les classes 2 et 3 (illustrées en gris et en blanc, respectivement), nous obtenons une segmentation précise de la *lamina terminalis*.

Les résultats d'estimation d'histogramme, pour toute une séquence, sont présentés dans le tableau 4.3. Dans ce tableau, les paramètres de chacune des trois gaussiennes utilisées pour approcher l'histogramme de chaque image sont donnés. Les paramètres de la $i^{\text{ième}}$ gaussienne d'une image sont notés P_i, μ_i et σ_i.

En fusionnant les classes 2 et 3, nous obtenons ainsi les contours de la *lamina terminalis* pour chaque image de la séquence illustrée à la figure 4.5 (a). Ces contours sont donnés à la figure 4.5 (b).

Nous comparons, à la figure 4.6, les résultats de segmentation de la méthode proposée avec ceux de cinq autres méthodes issues de la littérature. Il s'agit également de méthodes de seuillage non supervisées. Ces méthodes sont : la méthode d'Otsu classique [Otsu, 1979], une méthode basée sur l'entropie globale 2D (2DE) [Abutableb, 1989], une méthode basée sur l'entropie

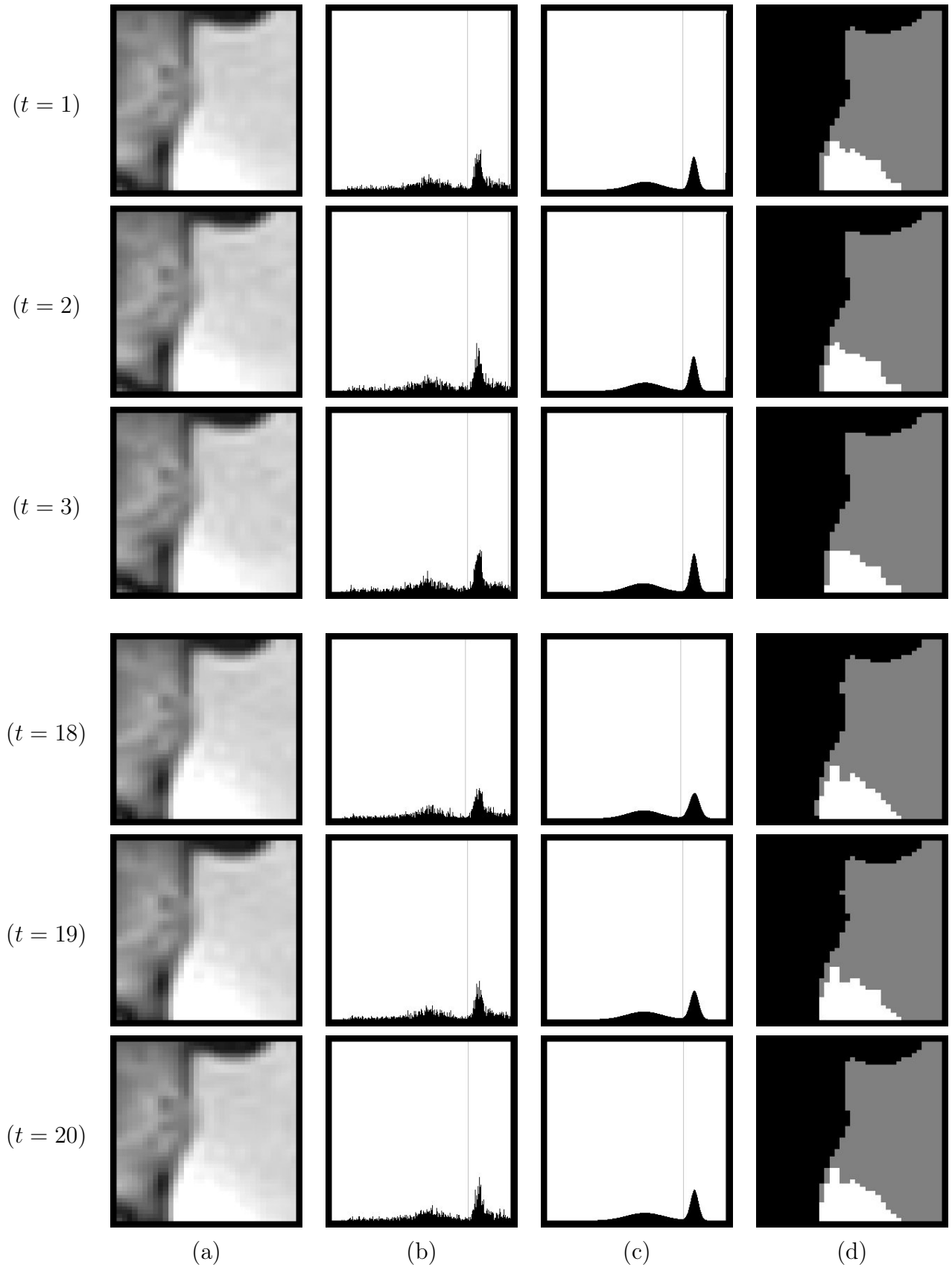
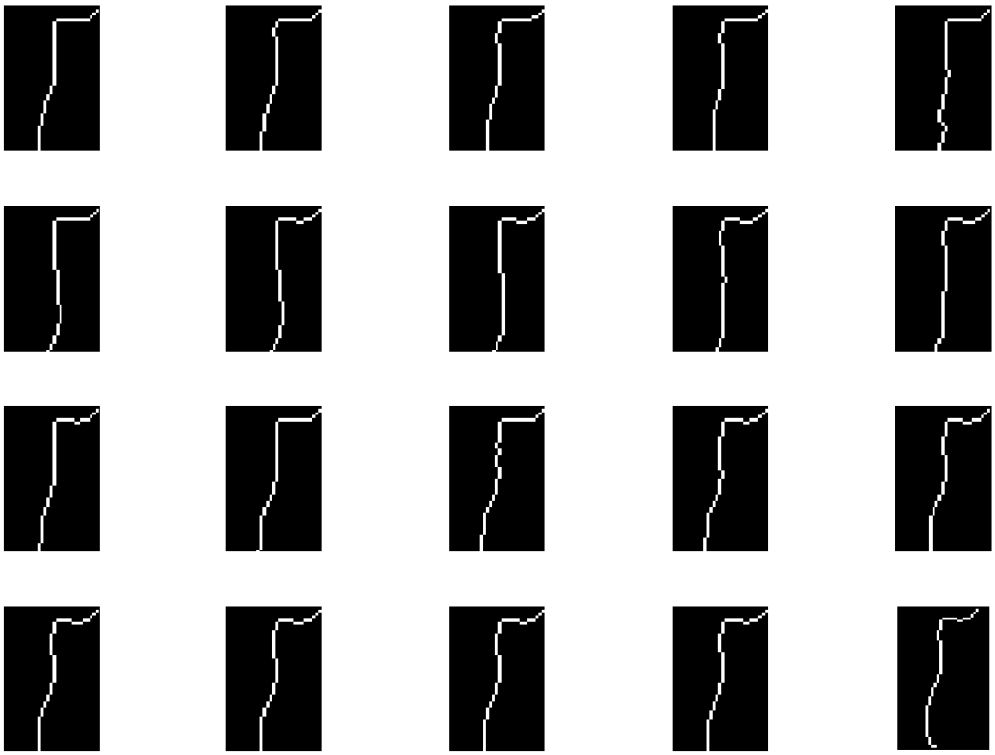


FIGURE 4.4 – Illustration de la procédure de segmentation. (a) Images originales. (b) Histogrammes originaux. (c) Histogrammes approchés. (d) Images segmentées.

(a)



(b)

FIGURE 4.5 – Illustration des résultats de segmentation. (a) Images originales de la région d'intérêt. (b) Contours des images segmentées après fusion des classes 2 et 3.

t	μ_1	P_1	σ_1	μ_2	P_2	σ_2	μ_3	P_3	σ_3
1	140	0,0051	31,5511	209	0,0213	7,7273	255	0,1139	0,6525
2	139	0,0051	32,1886	208	0,0215	7,7592	255	0,1099	0,6206
3	137	0,0052	32,0611	209	0,0225	7,5042	255	0,1041	0,6206
4	132	0,0050	32,8261	209	0,0211	8,5242	255	0,0892	0,6206
5	131	0,0048	33,2086	209	0,0183	10,5642	255	0,0759	0,6206
6	129	0,0052	36,7167	209	0,0182	10,4491	255	0,0761	0,2252
7	131	0,0054	35,0113	210	0,0168	11,4691	255	0,0751	0,2252
8	133	0,0057	32,6526	211	0,0164	11,9791	255	0,0766	0,2252
9	136	0,0057	30,9313	211	0,0145	14,1466	255	0,0792	0,2252
10	137	0,0059	28,4451	213	0,0127	16,6009	255	0,0864	0,2252
11	139	0,0060	25,3692	214	0,0118	18,9597	255	0,0957	0,2252
12	136	0,0058	25,9748	213	0,0118	18,8641	255	0,1108	0,2252
13	134	0,0056	28,7798	212	0,0126	16,7922	255	0,1192	0,2252
14	134	0,0055	32,8120	210	0,0151	12,5209	255	0,1256	0,2252
15	134	0,0057	33,6407	209	0,0163	11,0866	255	0,1240	0,2252
16	136	0,0057	32,8757	209	0,0159	11,5647	255	0,1246	0,2252
17	137	0,0057	32,9076	209	0,0170	10,8634	255	0,1261	0,2252
18	137	0,0056	32,9395	210	0,0180	10,3534	255	0,1261	0,2252
19	138	0,0055	35,7445	210	0,0198	8,8872	255	0,1240	0,2252
20	139	0,0053	38,1670	210	0,0210	8,1222	255	0,1230	0,2252

TABLEAU 4.3 – Paramètres des gaussiennes utilisées pour estimer l'histogramme de chaque image de la séquence considérée.

relative 2D (2DRE) [Qiao et al., 2007], une méthode basée sur l'entropie locale 2D (2DLE) [Althouse & Chang, 1995], et une méthode basée sur les K-moyennes [Muthukannan & Moses, 2010]. Par souci de concision, nous nous focalisons sur deux images successives d'une séquence. On remarque, aux figures 4.6(c) à (g), que les résultats de segmentation des cinq méthodes issues de la littérature sont moins bons : au niveau de la paroi du troisième ventricule cérébral, la séparation entre les régions intérieures et extérieures du ventricule n'est pas satisfaisante. Seule la méthode proposée permet de bien segmenter ces images (voir figure 4.6(b)).

Nous pouvons ensuite estimer l'amplitude des mouvements de chaque point des contours de la *lamina terminalis*, au cours d'un cycle cardiaque. Une illustration de cette quantification de mouvements est présentée à la figure 4.7, pour chaque point appartenant aux contours de la *lamina terminalis*, et pour chaque image d'une séquence. Sur cet exemple, l'amplitude maximale du mouvement de la *lamina terminalis* est de 3,47 mm. Ce résultat a été cliniquement validé par un expert, et il est du même ordre de grandeur que les résultats publiés dans [Hodel et al., 2009].

Enfin, nous calculons, pour chaque image (t) de la séquence, la différence entre la valeur de

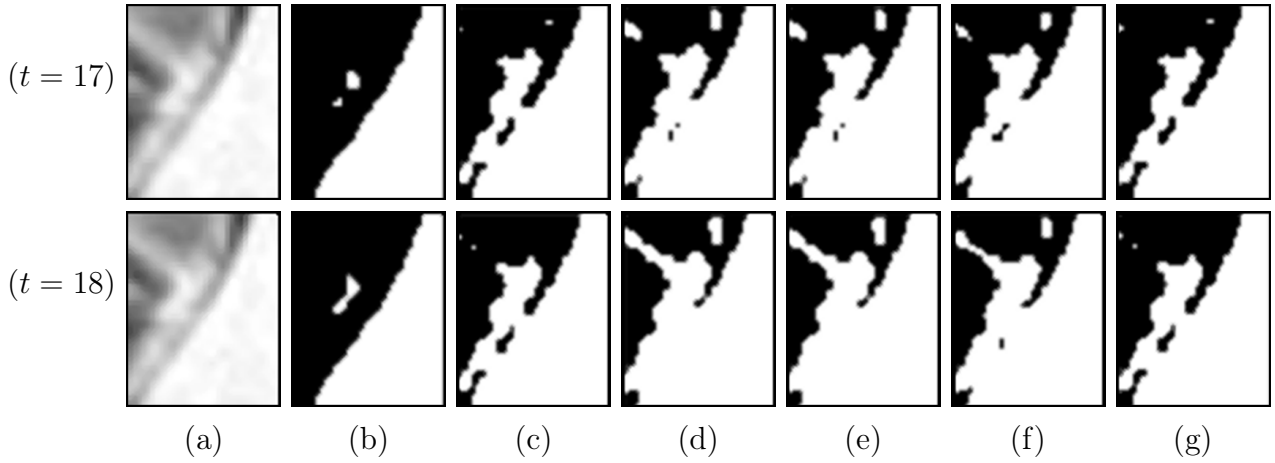


FIGURE 4.6 – Comparaison des résultats de segmentation de la méthode proposée avec ceux de méthodes issues de la littérature. (a) Images originales de la région d'intérêt. (b) Images segmentées en utilisant la méthode proposée. (c) à (g) Images segmentées en utilisant respectivement les méthodes : Otsu, 2DE, 2DRE, 2DLE et K-moyennes.

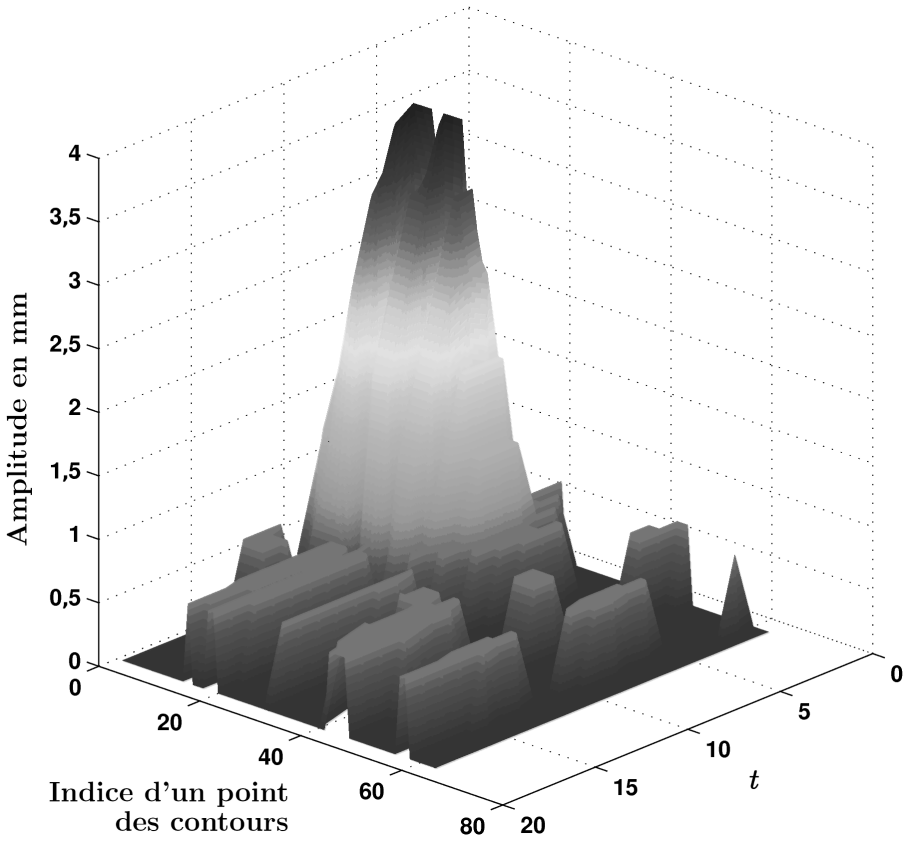


FIGURE 4.7 – Illustration de l'estimation finale de l'amplitude des mouvements dans la région d'intérêt (*lamina terminalis*).

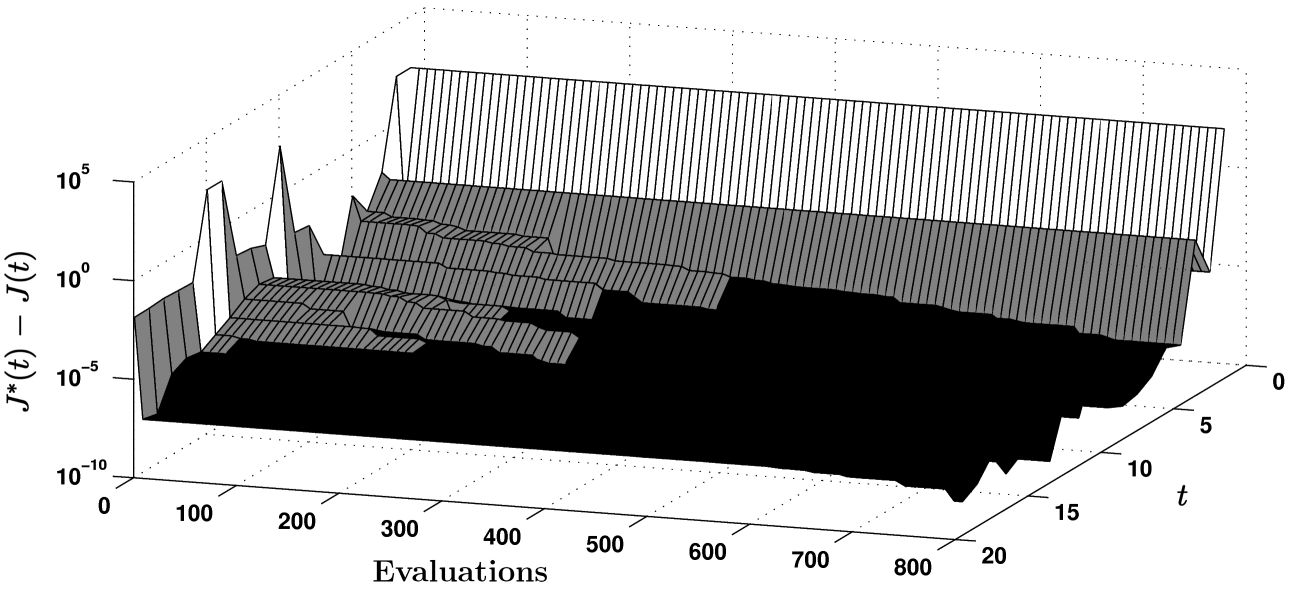


FIGURE 4.8 – Illustration de la convergence de CMADO.

la meilleure solution trouvée ($J^*(t)$) et celle de la solution courante ($J(t)$). L'évolution de cette différence (pouvant être considérée comme une mesure de la stagnation de l'algorithme), sur les 800 premières évaluations de la fonction objectif, est présentée à la figure 4.8. Sur cette figure, l'indice de chaque image est donné sur l'axe des abscisses, le nombre d'évaluations de la fonction objectif est donné sur l'axe des ordonnées, et l'écart ($J^*(t) - J(t)$) est donné sur le troisième axe. Pour plus de clarté, nous utilisons une échelle logarithmique sur l'axe z. Cette figure nous donne ainsi des indications sur la convergence de CMADO. Notons également que les résultats présentés sur cette figure sont ceux de l'exécution de performance médiane, parmi 20 exécutions de l'algorithme. Le temps d'exécution pour segmenter une séquence de 20 images, moyenné sur 20 exécutions de l'algorithme, est de $11{,}18 \pm 1{,}44$ secondes. Le nombre moyen d'évaluations pour segmenter une séquence est : 7537 ± 988.

Sur cette figure, nous constatons que le nombre d'évaluations nécessaires à CMADO pour converger vers une solution acceptable décroît à mesure que le nombre d'images segmentées augmente. Nous montrons ainsi l'intérêt d'utiliser un algorithme d'optimisation dynamique tel que CMADO sur ce problème.

4.6 Conclusion

Dans ce chapitre, nous avons présenté une méthode permettant de quantifier de manière précise les mouvements des parois du troisième ventricule cérébral, à partir de séquences ciné-IRM du cerveau. Il s'agit d'une méthode innovante, pouvant grandement aider les médecins, en particulier dans le diagnostic et le traitement de l'hydrocéphalie. Nous avons proposé d'intégrer

à cette méthode une procédure de segmentation par estimation d'histogramme, bénéficiant de l'efficacité de l'algorithme d'optimisation dynamique CMADO. Les résultats montrent en effet l'intérêt d'utiliser un algorithme d'optimisation dynamique, tel que CMADO sur ce type de problème.

La méthode de quantification de mouvements que nous avons présentée comporte deux étapes : une étape de segmentation, suivie d'une étape de recalage permettant de suivre les déplacements des parties en mouvement. Dans ce chapitre, nous nous sommes focalisés sur l'étape de segmentation. Dans le chapitre suivant, nous allons voir comment accélérer l'étape de recalage, au moyen de l'optimisation dynamique.

Chapitre Cinq

Application de MLSDO au recalage de séquences d'images médicales

5.1 Introduction

Dans le chapitre précédent, nous avons présenté une méthode de quantification des mouvements des parois du troisième ventricule cérébral, à partir de séquences ciné-IRM du cerveau. Nous nous étions focalisés sur la première étape de cette méthode, visant à segmenter chaque image de la séquence ciné-IRM traitée. Le but de cette étape de segmentation est d'extraire les contours de la région d'intérêt, à différents instants du cycle cardiaque. Nous pouvons ensuite, à la deuxième étape de la méthode, combiner les informations fournies par ces contours, au moyen d'une procédure de recalage. Cette procédure de recalage nous permet de suivre les déplacements de chaque point du contour de la région d'intérêt au cours du temps, et de quantifier son déplacement maximum. Elle nous permet également d'obtenir une meilleure modélisation mathématique des déformations des parois du troisième ventricule.

Dans ce chapitre, nous formulons tout d'abord le problème du recalage, puis nous présentons un état de l'art des méthodes de recalage. Ensuite, nous décrivons en détail la procédure de recalage utilisée dans cette méthode de quantification de mouvements. Enfin, nous proposons deux améliorations de cette méthode : la première consiste à accélérer la procédure de recalage au moyen de l'algorithme d'optimisation dynamique MLSDO [Lepagnot et al., 2011d], et la seconde consiste à améliorer la précision de la méthode [Lepagnot et al., 2011a,e].

5.2 Problème du recalage

5.2.1 Définition du problème

L'idée directrice du recalage est de corréler les informations fournies par deux images (ou plus) d'une même scène mais obtenues à différents instants, sous différents points de vue ou par différentes modalités d'acquisition. Un système de recalage permet ainsi la mise en correspondance d'informations (positions, niveaux de gris, structures...), représentant une même réalité physique sur différentes images.

Le recalage d'images consiste à trouver la meilleure transformation (ou déformation) $\widetilde{T}$, parmi l'ensemble des transformations T, appliquée à une image source I, pour donner une image qui ressemble au mieux à une image cible J. Dès lors, un certain nombre de questions se posent : quelles informations utiliser pour guider le recalage ? Comment définir la ressemblance entre deux images ? Comment déformer une image ? Comment trouver la meilleure déformation ?

Ces questions mènent aux quatre critères utilisés dans la littérature pour la définition des composants des méthodes de recalage [Brown, 1992; Maintz & Viergever, 1998; Barillot, 1999] :

1. **Primitives :** ce sont les caractéristiques extraites des images. Elle peuvent être géométriques (surfaces, courbes, points) ou iconiques (intensités des pixels).
2. **Critère de similarité :** il définit une certaine « distance » entre les primitives des images source et cible, afin de quantifier la notion de ressemblance.
3. **Modèle de déformation :** il conditionne la manière dont l'image est géométriquement modifiée. Il peut être global ou local.
4. **Procédure d'optimisation :** c'est la méthode qui permet de déterminer la meilleure transformation, au sens du critère de similarité, en explorant l'espace de recherche correspondant au modèle de déformation.

Le recalage d'images fonctionne selon le principe illustré à la figure 5.1.

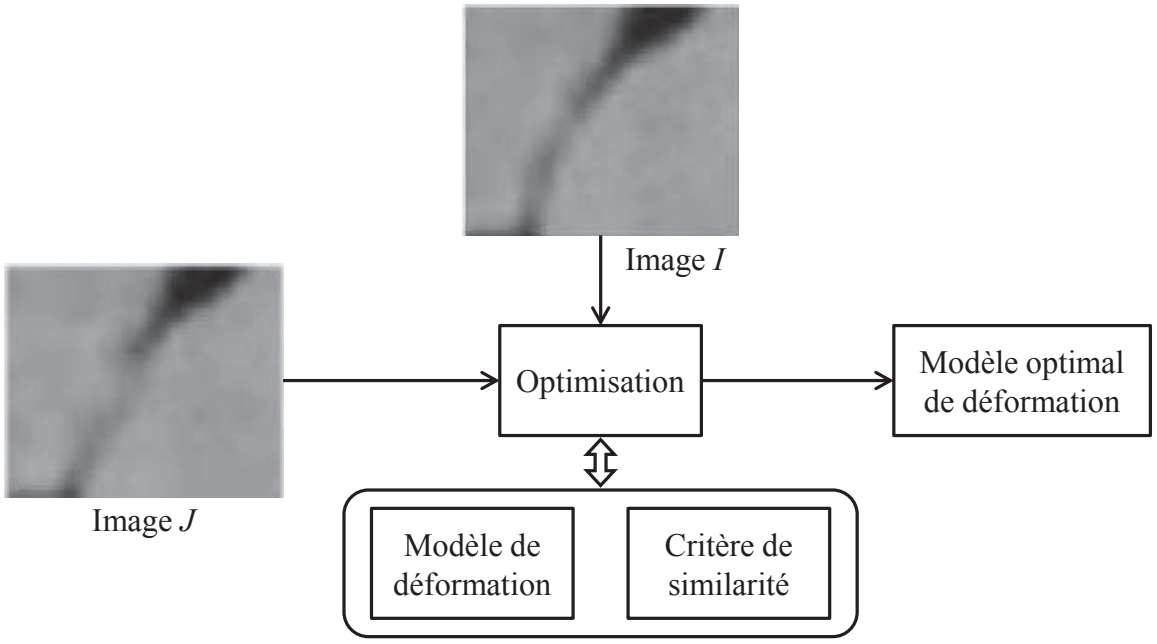


FIGURE 5.1 – Principe du recalage d'images.

5.2.2 Les différentes approches du recalage d'images

Il est quasiment impossible de recenser toutes les méthodes de recalage d'images proposées dans la littérature. Les articles de Brown et de Zitová et Flusser [Brown, 1992; Zitová & Flusser, 2003] traitent d'un grand nombre d'entre elles. L'état de l'art écrit par Zhang [Zhang, 1993] couvre plusieurs domaines de la vision artificielle. Ceux de Maintz et Viergever et de Santamaría et al. [Maintz & Viergever, 1998; Santamaría et al., 2011] sont spécifiques au traitement d'images médicales et constituent d'excellentes synthèses techniques.

Selon les primitives utilisées, deux grandes familles de méthodes de recalage existent : les approches géométriques et les approches iconiques. Chacune d'elles a ses propres critères de similarité. Par ailleurs, des méthodes hybrides combinant ces approches ont aussi été proposées.

5.2.2.1 Approches géométriques

Les approches géométriques consistent à extraire des primitives géométriques (surfaces, courbes, points), puis à les mettre en correspondance, en utilisant des critères de similarité géométriques. Dans le cas où les primitives sont des points, la distance Euclidienne est généralement utilisée [Noblet, 2006]. Dans le cas des surfaces et des courbes, l'algorithme *Iterative Closest Point* [Besl & McKay, 1992; Zhang, 1994] est le plus utilisé. D'autres critères géométriques sont basés sur le calcul des cartes de distances de Chanfrein ou le calcul des distances de Hausdorff [Noblet, 2006].

Le principal avantage des approches géométriques est la manipulation d'une représentation compacte de l'image, ayant pour conséquence une charge calculatoire beaucoup plus faible que dans le cas des approches iconiques. De plus, les primitives utilisées portent une information de haut niveau, souvent plus discriminante que l'information d'intensité.

L'inconvénient majeur de ces approches est l'imprécision relative à l'extraction des primitives. Une autre limite concerne la précision du recalage, qui n'est garantie que dans le voisinage des primitives. De plus, les points d'une même courbe ou surface sont indiscernables. Leur mise en correspondance s'avère donc arbitraire.

5.2.2.2 Approches iconiques

Le principe des approches iconiques repose sur la prise en compte des informations portées par les niveaux de gris (ou intensités lumineuses) des images, soit par comparaison directe des intensités, soit par comparaison des valeurs de l'image après traitement (par exemple, après application d'une transformée de Fourier [Mellor & Brady, 2005]).

Selon Roche [Roche, 2001], les principaux critères de similarité utilisés par les approches iconiques se divisent en quatre classes :

1. Les mesures de différences d'images en général [Buzug & Weese, 1998], qui supposent que les mêmes objets auront les mêmes intensités dans les deux images à recaler.
2. Le coefficient de corrélation, qui suppose un lien affine entre les intensités d'un même objet représenté dans les deux images à recaler.
3. Le critère de Woods [Woods et al., 1993] et ses variantes [Ardekani et al., 1995; Alpert et al., 1996; Nikou et al., 1999], ainsi que le rapport de corrélation [Roche et al., 1998], rendent compte d'une dépendance fonctionnelle entre les intensités des images à recaler.
4. Enfin, une dépendance de nature statistique entre deux images peut être quantifiée par l'entropie conjointe [Studholme et al., 1995; Collignon et al., 1995b], l'information mutuelle [Viola & III, 1995; Collignon et al., 1995a], ou l'information mutuelle normalisée [Studholme et al., 1999].

Le principal avantage de ces méthodes est d'utiliser toute l'information portée par l'image. De plus, elles sont, pour la plupart, complètement automatiques.

Néanmoins, ces méthodes souffrent d'un coût calculatoire important, dû à l'obligation de considérer chaque pixel de l'image.

5.2.2.3 Approches mixtes

Pour améliorer la robustesse des méthodes de recalage, d'autres approches, dites mixtes, ont été proposées. Elles reposent sur l'utilisation de plusieurs types d'information différents, combinant ainsi les avantages liés à chacun d'eux. Trois cas peuvent être distingués : la combinaison de primitives géométriques de nature différente [Maurer et al., 1998; Hsu et al., 1999], la combinaison de différentes informations issues des niveaux de gris [Pluim et al., 2000; Shen & Davatzikos, 2002] et la combinaison des approches géométriques et iconiques [Hellier & Barillot, 2003; Lin et al., 2010].

5.2.3 Modèles de déformation

Parmi les différents modèles de déformation proposés dans la littérature, nous distinguons les modèles linéaires, qui regroupent les transformations rigide, affine et projective, et les modèles non linéaires. Nous distinguons également les transformations dites « globales », s'appliquant à toute l'image, des transformations dites « locales », s'appliquant à des sous-régions de l'image

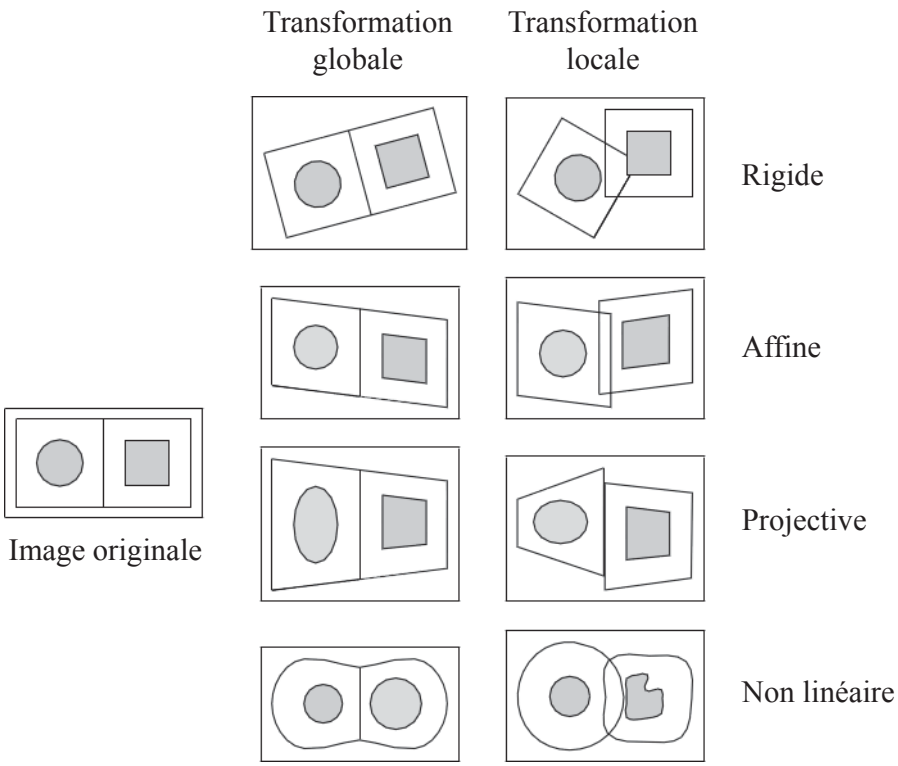


FIGURE 5.2 – Exemples synthétiques illustrant l'effet de différents types de transformations [Maintz & Viergever, 1998].

(une transformation différente est alors appliquée à chacune de ces sous-régions). Cette classification est illustrée à la figure 5.2 [Maintz & Viergever, 1998; Noblet, 2006], et détaillée ci-dessous :

1. **Modèles linéaires :** il s'agit de transformations pouvant être formulées en considérant les coordonnées homogènes grâce à un produit matriciel, selon l'équation (5.2.1). Dans cette équation, (x, y) et (x', y') sont les coordonnées initiales et les coordonnées transformées, respectivement.

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 \\ m_7 & m_8 & m_9 \end{pmatrix} . \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{5.2.1}$$

   Les modèles linéaires englobent les transformations suivantes :

   – **Transformation rigide :** elle consiste en une rotation et une translation. Elle conserve les distances, les angles et le parallélisme. Pour des images 2D, elle comporte 3 degrés de liberté.
   – **Transformation affine :** elle autorise, en plus d'une rotation et d'une translation, de prendre en compte un facteur d'échelle anisotrope et de modéliser des cisaillements. Ce

type de transformation conserve le parallélisme. Le nombre de degrés de liberté est de 6 en 2D.

- **Transformation projective :** elle est utilisée principalement pour prendre en compte les effets de perspective dans l'image. Elle ne garantit plus le parallélisme, mais impose que l'image d'une droite soit une droite. Le nombre de degrés de liberté est de 8 en 2D.

2. **Modèles non linéaires :** ils ont un nombre de degrés de liberté beaucoup plus important que les modèles linéaires. En imagerie médicale, ils permettent ainsi de mieux modéliser les déformations complexes d'un ensemble de tissus, ou les distorsions géométriques inhérentes à certaines modalités d'acquisition.

5.2.4 Formulation du problème d'optimisation

L'optimisation est une étape importante en recalage d'images. Elle a pour but de déterminer une transformation optimale, selon un critère de similarité. D'une manière générale, le problème d'optimisation est formulé ainsi :

$$\widetilde{T} = \arg\max_{T \in S} f(T(I), J) \tag{5.2.2}$$

où I est l'image source devant être recalée sur l'image cible J, S est l'espace de recherche des transformations possibles, f est le critère de similarité choisi et $\widetilde{T}$ est la transformation optimale selon ce critère.

5.3 Recalage de séquences d'images ciné-IRM segmentées

Dans ce paragraphe, nous supposons disposer de séquences ciné-IRM parfaitement segmentées. Un exemple de séquence segmentée est illustré à la figure 5.3. Le but est de mettre en correspondance les points des contours, pour chaque couple d'images consécutives de la séquence. Les transformations permettant cette mise en correspondance sont ensuite utilisées pour quantifier les mouvements des contours, au cours d'un cycle cardiaque. Afin de calculer ces transformations, nous utilisons l'algorithme d'optimisation dynamique MLSDO.

La méthode utilisée pour mettre en correspondance les points des contours, pour chaque couple d'images, est inspirée de [Nakib et al., 2010]. Cette opération permet de suivre la position de chaque point de ces contours, au cours du temps. Elle s'effectue en deux étapes : l'étape d'appariement et l'étape de recalage.

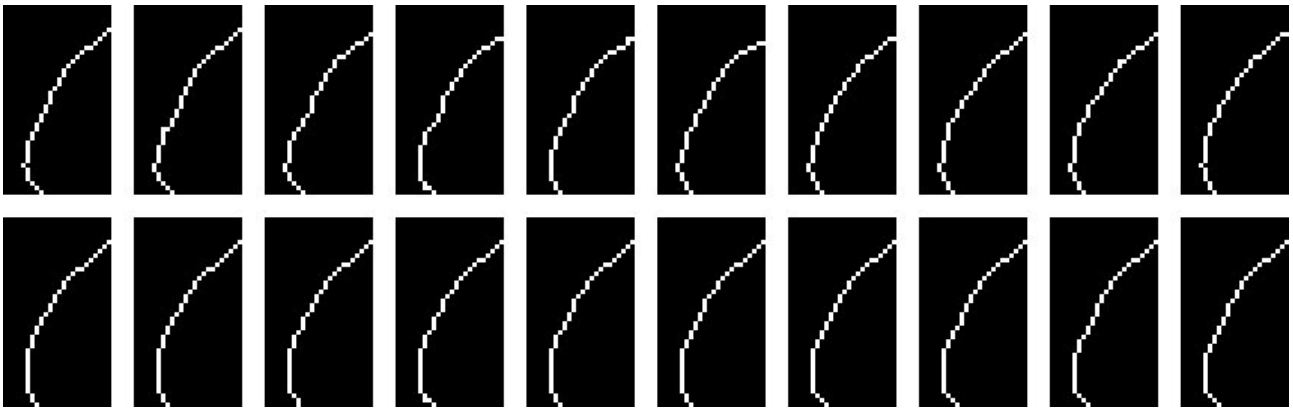
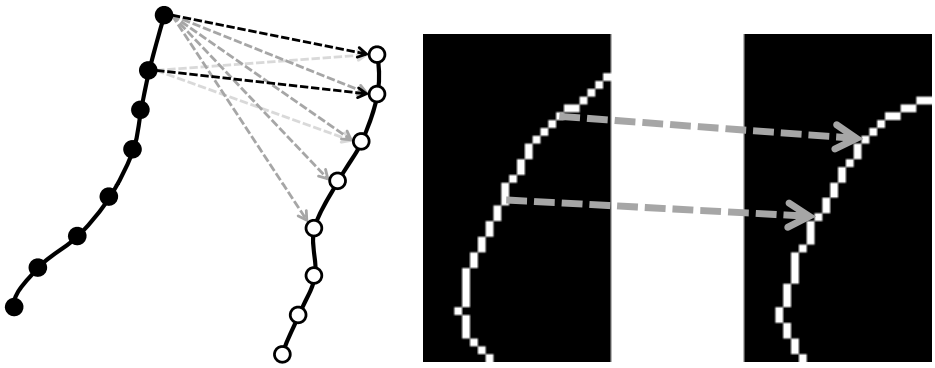


FIGURE 5.3 – Exemple de séquence d'images segmentées de la *lamina terminalis*.

5.3.1 L'étape d'appariement

Cette étape est illustrée par un exemple à la figure 5.4 : en (a), nous présentons deux contours synthétiques à recaler. Le but est d'associer chaque point du premier contour à un point du deuxième. Nous faisons ainsi l'hypothèse que chaque point du premier contour peut être associé à au moins un point du deuxième contour. Le but de cette procédure est illustré en (b) pour des contours réels.



(a) (b)

FIGURE 5.4 – Illustration de la procédure d'appariement : (a) contours synthétiques, (b) contours réels de la *lamina terminalis* [Nakib et al., 2010].

Soient C_1 et C_2 deux courbes constituées de L_1 et L_2 points, respectivement, correspondant aux contours de deux images successives d'une séquence. A la fin de cette procédure d'appariement, nous obtenons un ensemble C_1' de points triés du contour C_2, correspondant chacun à un point du contour C_1.

5.3.2 L'étape de recalage

Afin que la modélisation des mouvements de la *lamina terminalis* soit facile à interpréter, une transformation affine est utilisée. En notant A^{T} la transposée d'une matrice A, nous supposons qu'il existe une transformation T_Φ permettant d'avoir $C_2 = T_\Phi(C_1)$. Pour chaque point de vecteur position $(x_1\ y_1)^{\mathrm{T}}$ de C_1, cette transformation est définie par :

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} s_1.\cos\theta & -s_2.\sin\theta \\ s_1.\sin\theta & s_2.\cos\theta \end{pmatrix} . \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{5.3.1}$$

En utilisant les coordonnées homogènes, (5.3.1) devient :

$$\begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} = \begin{pmatrix} s_1.\cos\theta & -s_2.\sin\theta & t_x \\ s_1.\sin\theta & s_2.\cos\theta & t_y \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} \tag{5.3.2}$$

Les paramètres s_1 et s_2 correspondent à une homothétie, θ correspond à une rotation, t_x et t_y correspondent à une translation. L'ensemble des paramètres $\Phi = \{s_1, s_2, \theta, t_x, t_y\}$ est estimé en minimisant l'erreur quadratique moyenne (MSE) calculée à partir des ensembles de points appariés. Ainsi, le critère d'optimisation est donné par :

$$MSE(\Phi) = \frac{1}{L_1} \sum_{j=1}^{L_1} \left[C_1'(j) - T_\Phi(C_1(j))\right]^{\mathrm{T}} \left[C_1'(j) - T_\Phi(C_1(j))\right] \tag{5.3.3}$$

où le $j^{\text{ième}}$ point de C_1' et le $j^{\text{ième}}$ point de C_1 sont notés $C_1'(j)$ et $C_1(j)$, respectivement.

Le problème du recalage peut alors être formulé comme un problème d'optimisation défini par :

$$\min MSE(\Phi) \tag{5.3.4}$$

Nous allons voir maintenant que ce problème peut s'exprimer comme un problème d'optimisation dynamique, et nous allons utiliser MLSDO pour le résoudre.

5.3.3 Formulation sous la forme d'un problème d'optimisation dynamique

Le recalage d'une séquence ciné-IRM peut être vu comme un problème d'optimisation dynamique pour lequel la fonction objectif, optimisée par MLSDO, change, selon les règles suivantes :

– Comme nous avons à traiter une séquence, le critère (5.3.3) doit être minimisé pour chaque couple de contours. Le critère d'optimisation devient alors :

$$MSE(\Phi(t), t) = \frac{1}{L_t} \sum_{j=1}^{L_t} \left[C'_t(j) - T_{\Phi(t)}(C_t(j))\right]^{\mathrm{T}} \left[C'_t(j) - T_{\Phi(t)}(C_t(j))\right] \tag{5.3.5}$$

où t est l'indice des contours sur lesquels la transformation $T_{\Phi(t)}$ est appliquée. Il correspond également au couple de contours en cours de recalage d'une séquence. $\Phi(t)$, $C'_t(j)$, $C_t(j)$ et L_t correspondent respectivement à Φ, C'_1, C_1 et L_1 définis précédemment, mais dépendent ici du couple de contours en cours de recalage.

– Ainsi, le problème d'optimisation dynamique est défini par :

$$\min MSE(\Phi(t), t) \tag{5.3.6}$$

– Si la meilleure solution (transformation) courante trouvée pour le couple t ne peut être améliorée davantage (selon un critère de stagnation), alors le couple suivant $(t+1)$ est traité.

– Le critère de stagnation utilisé est satisfait si aucune amélioration significative (supérieure à 1E−5) n'est observée pour la meilleure solution courante, durant 200 évaluations successives de la fonction objectif.

– Ainsi, la fin du recalage d'un couple de contours et le début du recalage du couple suivant constituent un changement dans la fonction objectif.

5.3.4 Résultats et discussion

Nous allons maintenant comparer les résultats obtenus par MLSDO à ceux d'algorithmes d'optimisation statique connus, pour ce problème. Le but est de montrer l'intérêt d'utiliser pour ce problème un algorithme d'optimisation dynamique, tel que MLSDO. Les valeurs optimales des paramètres de MLSDO utilisées pour ce problème ont été trouvées empiriquement, en les ajustant manuellement, de façon à minimiser le nombre d'évaluations nécessaires pour obtenir un recalage satisfaisant. Ces valeurs sont données dans le tableau 5.1.

Paramètre	n_a	n_c	r_l	r_e	δ_{ph}	δ_{pl}
Valeur	1	2	1E−3	1E−1	1E−7	1E−5

TABLEAU 5.1 – Paramétrage de MLSDO.

Les algorithmes statiques utilisés figurent parmi les plus connus et les plus performants. Ces algorithmes et leurs paramètres, ajustés empiriquement pour ce problème, de la même manière que pour MLSDO, sont présentés ci-dessous (voir les références données, pour plus de détails sur ces algorithmes et leur paramétrage) :

- CMA-ES (*Covariance Matrix Adaptation Evolution Strategy*) [Hansen & Ostermeier, 2001] en utilisant le paramétrage recommandé, à l'exception du pas initial σ, fixé à $\sigma = 0{,}3$. La taille de la population d'enfants λ et le nombre d'individus sélectionnés μ sont fixés à $\lambda = 8$ et $\mu = 4$.
- SPSO-07 (*Standard Particle Swarm Optimization* dans sa version 2007) [Clerc et al., 2011] en utilisant le paramétrage recommandé, à l'exception du nombre S de particules ($S = 6$) et du paramètre K utilisé pour engendrer le voisinage des particules ($K = 5$).
- DE (*Differential Evolution*) [Price et al., 2005] en utilisant la stratégie « DE/target-to-best/1/bin », un nombre de parents $NP = 12$, un facteur d'amplification $F = 0{,}8$ et une constante de croisement $CR = 0{,}9$.

Comme ce sont des algorithmes d'optimisation statique, nous devons considérer le recalage de chaque couple de contours successifs comme un nouveau problème à résoudre. De ce fait, chacun de ces algorithmes est redémarré, une fois le recalage d'un couple de contours terminé, selon le critère de stagnation défini au paragraphe 5.3.3. Les résultats obtenus par MLSDO, utilisé en tant qu'algorithme d'optimisation statique (i.e. redémarré après le recalage de chaque couple de contours), sont également donnés.

Le recalage des trois premiers couples de contours, effectué par MLSDO, est illustré à la figure 5.5. Les paramètres du modèle de déformation, trouvés par chaque algorithme, sont donnés dans les tableaux 5.2 à 5.6, pour l'exécution de performance médiane parmi 20 exécutions. Le nombre d'évaluations effectuées par chaque algorithme, moyenné sur 20 exécutions, est donné dans le tableau 5.7. Le temps d'exécution, moyenné sur 20 exécutions de chaque algorithme, codé en C et exécuté sur un processeur Intel Core i5 cadencé à 2,27 GHz, est également donné. La somme des erreurs quadratiques moyennes (MSE, voir les équations (5.3.3) et (5.3.5)) du recalage d'une séquence est aussi donnée dans ce tableau, moyennée sur 20 exécutions. Les courbes de convergence de MLSDO et de l'algorithme le plus performant pour ce problème, CMA-ES, sont illustrées à la figure 5.6. Dans cette figure, le nombre d'évaluations par recalage d'un couple de contours est fixé à 2000, afin de pouvoir comparer la convergence des algorithmes. Pour plus de clarté, nous utilisons une échelle logarithmique pour l'axe des ordonnées.

t	s_1	s_2	θ	t_x	t_y	$MSE(\Phi(t), t) \times 10^5$
1	1,021	0,999	-0,012	-0,005	0,147	12755,6
2	1,004	0,977	0,000	-0,021	0,336	16656,5
3	0,995	0,986	-0,004	0,042	0,213	17894,4
4	0,973	1,007	0,017	-0,023	0,123	15675,6
5	1,005	0,997	0,003	-0,096	0,116	18183,7
6	0,996	0,996	-0,005	-0,180	0,037	21043,2
7	1,017	0,993	-0,014	-0,264	0,005	17799,0
8	1,012	0,994	-0,011	-0,150	0,000	11328,8
9	0,961	1,000	0,000	-0,174	0,000	11961,9
10	0,988	0,997	-0,006	-0,132	0,000	13053,4
11	1,000	1,000	0,000	0,000	0,000	0,1
12	1,012	0,994	-0,012	-0,012	0,021	7118,1
13	1,016	0,993	-0,015	-0,039	-0,003	6891,8
14	0,976	1,007	0,015	-0,018	0,076	8927,5
15	1,004	1,000	-0,002	0,000	0,024	2535,8
16	1,033	0,987	-0,026	0,009	0,024	8197,4
17	1,006	0,999	-0,003	0,000	0,026	2550,3
18	1,012	0,998	-0,007	-0,002	0,076	7103,7
19	1,010	0,997	-0,006	-0,002	0,054	4922,5

TABLEAU 5.2 – Transformations trouvées par MLSDO et leurs MSE correspondantes, pour le recalage de chaque couple de contours.

t	s_1	s_2	θ	t_x	t_y	$MSE(\Phi(t), t) \times 10^5$
1	1,020	1,000	-0,011	-0,005	0,148	12754,4
2	1,002	0,977	0,001	-0,022	0,334	16652,8
3	0,993	0,986	-0,003	0,042	0,213	17891,9
4	0,972	1,008	0,018	-0,022	0,123	15674,1
5	1,008	0,996	0,002	-0,098	0,115	18179,6
6	0,999	0,995	-0,007	-0,181	0,036	21039,2
7	1,020	0,992	-0,016	-0,266	0,007	17795,1
8	1,010	0,995	-0,010	-0,150	0,002	11325,9
9	0,960	1,000	0,001	-0,173	0,000	11960,6
10	0,990	0,997	-0,007	-0,132	0,000	13051,7
11	1,000	1,000	0,000	0,000	0,000	0,0
12	1,014	0,994	-0,014	-0,012	0,023	7114,3
13	1,018	0,993	-0,016	-0,040	-0,002	6889,9
14	0,975	1,007	0,015	-0,019	0,077	8926,3
15	1,005	0,999	-0,003	0,000	0,024	2533,8
16	1,034	0,986	-0,026	0,008	0,025	8196,5
17	1,004	0,999	-0,002	-0,001	0,026	2548,4
18	1,013	0,998	-0,008	-0,002	0,075	7102,1
19	1,009	0,997	-0,005	-0,002	0,053	4922,0

TABLEAU 5.3 – Transformations trouvées par CMA-ES et leurs MSE correspondantes, pour le recalage de chaque couple de contours.

t	s_1	s_2	θ	t_x	t_y	$MSE(\Phi(t), t) \times 10^5$
1	1,021	0,999	-0,012	-0,005	0,149	12755,3
2	1,002	0,977	0,001	-0,022	0,334	16652,9
3	0,993	0,987	-0,003	0,042	0,213	17892,0
4	0,972	1,008	0,018	-0,021	0,125	15674,8
5	1,011	0,995	0,000	-0,100	0,120	18188,1
6	0,999	0,995	-0,006	-0,180	0,035	21039,3
7	1,020	0,992	-0,016	-0,268	0,006	17795,5
8	1,011	0,995	-0,010	-0,150	0,002	11326,3
9	0,961	1,000	0,001	-0,174	0,001	11960,8
10	0,990	0,997	-0,007	-0,131	-0,001	13051,7
11	0,998	1,000	0,001	-0,001	0,000	2,5
12	1,016	0,994	-0,014	-0,015	0,023	7117,0
13	1,018	0,993	-0,016	-0,040	-0,003	6889,9
14	0,974	1,007	0,015	-0,018	0,078	8926,6
15	1,006	0,999	-0,003	0,000	0,024	2533,9
16	1,034	0,986	-0,027	0,007	0,025	8196,7
17	1,004	0,999	-0,002	0,000	0,026	2548,7
18	1,013	0,998	-0,008	-0,002	0,075	7102,1
19	1,009	0,997	-0,005	-0,003	0,053	4922,4

TABLEAU 5.4 – Transformations trouvées par SPSO-07 et leurs MSE correspondantes, pour le recalage de chaque couple de contours.

t	s_1	s_2	θ	t_x	t_y	$MSE(\Phi(t), t) \times 10^5$
1	1,020	1,000	-0,011	-0,004	0,148	12754,5
2	1,002	0,977	0,001	-0,021	0,334	16652,9
3	0,994	0,987	-0,003	0,041	0,212	17893,4
4	0,972	1,008	0,017	-0,022	0,122	15674,3
5	1,008	0,996	0,002	-0,097	0,116	18179,7
6	0,998	0,995	-0,006	-0,181	0,035	21039,5
7	1,020	0,992	-0,016	-0,267	0,006	17795,2
8	1,010	0,995	-0,010	-0,150	0,002	11325,9
9	0,960	1,000	0,001	-0,173	0,000	11961,1
10	0,990	0,997	-0,007	-0,131	0,000	13051,7
11	1,001	1,000	0,000	-0,001	0,000	0,4
12	1,015	0,994	-0,014	-0,012	0,023	7114,3
13	1,018	0,993	-0,016	-0,040	-0,003	6889,9
14	0,975	1,007	0,015	-0,019	0,077	8926,4
15	1,005	0,999	-0,003	-0,001	0,025	2533,9
16	1,034	0,986	-0,027	0,009	0,025	8196,5
17	1,004	0,999	-0,002	-0,001	0,026	2548,5
18	1,013	0,998	-0,007	-0,002	0,075	7102,2
19	1,009	0,997	-0,005	-0,002	0,053	4922,0

TABLEAU 5.5 – Transformations trouvées par DE et leurs MSE correspondantes, pour le recalage de chaque couple de contours.

t	s_1	s_2	θ	t_x	t_y	$MSE(\Phi(t), t) \times 10^5$
1	1,021	0,999	-0,012	-0,004	0,148	12755,6
2	0,998	0,978	0,003	-0,022	0,336	16661,0
3	0,995	0,986	-0,004	0,040	0,214	17893,5
4	0,973	1,007	0,017	-0,022	0,124	15675,7
5	1,010	0,995	0,001	-0,100	0,116	18181,9
6	1,000	0,995	-0,007	-0,182	0,037	21040,7
7	1,021	0,992	-0,017	-0,267	0,006	17797,2
8	1,012	0,994	-0,011	-0,149	0,004	11328,6
9	0,962	1,000	0,000	-0,174	0,000	11962,9
10	0,988	0,997	-0,006	-0,132	-0,001	13053,5
11	0,999	1,000	0,001	0,000	0,000	0,7
12	1,013	0,994	-0,013	-0,012	0,024	7115,2
13	1,017	0,993	-0,015	-0,039	-0,002	6890,6
14	0,976	1,007	0,015	-0,020	0,077	8927,3
15	1,007	0,999	-0,004	-0,001	0,026	2537,1
16	1,031	0,987	-0,026	0,008	0,027	8199,9
17	1,003	0,999	-0,002	-0,001	0,025	2549,4
18	1,012	0,999	-0,006	-0,003	0,076	7105,1
19	1,011	0,997	-0,006	-0,002	0,054	4923,2

TABLEAU 5.6 – Transformations trouvées par MLSDO, utilisé comme un algorithme d'optimisation statique, et leurs MSE correspondantes, pour le recalage de chaque couple de contours.

	Algorithme	Evaluations	Temps d'exécution (ms)	MSE cumulée
Optimisation dynamique	MLSDO	13841,80 ± 386,55	13,2 ± 5,7	2,05 ± 1,3E−4
Optimisation statique	CMA-ES	17651,65 ± 288,10	5029,5 ± 212,8	2,05 ± 1,6E−2
	SPSO-07	19611,70 ± 994,13	49,8 ± 22,5	2,05 ± 2,7E−2
	DE	21749,60 ± 869,46	56,4 ± 23,4	2,05 ± 2,8E−5
	MLSDO	28891,15 ± 908,88	29,8 ± 20,3	2,05 ± 1,8E−4

TABLEAU 5.7 – Nombre d'évaluations et temps d'exécution moyens pour recaler une séquence, et somme des MSE, obtenus par chaque algorithme.

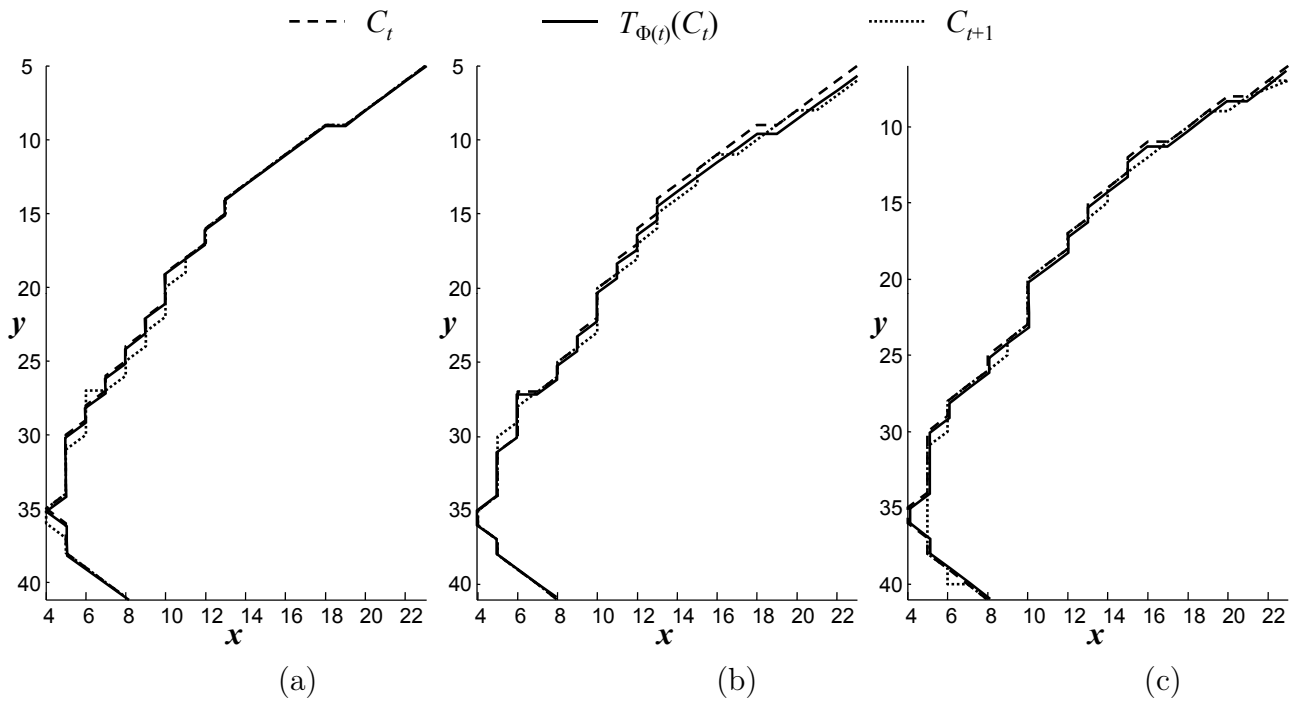


FIGURE 5.5 – Illustration du recalage des trois premiers couples de contours d'une séquence : (a) premier couple, (b) deuxième couple, (c) troisième couple.

Dans les tableaux 5.2 à 5.6, nous constatons que les transformations trouvées par les différents algorithmes ne sont pas significativement différentes : les valeurs des paramètres s_1, s_2, θ, t_x et t_y ne diffèrent pas de plus de 5,9E−3, 1,8E−3, 3,0E−3, 3,7E−3 et 4,5E−3, respectivement. En particulier, la MSE du recalage d'un couple de contours ne diffère pas de plus de 8,5E−5 entre deux algorithmes. Ce niveau de précision est satisfaisant pour ce problème de recalage. De plus, les sommes de MSE données au tableau 5.7 montrent que les algorithmes ont des précisions moyennes similaires.

Néanmoins, nous constatons dans le tableau 5.7 que les nombres d'évaluations de la fonction objectif effectuées par les différents algorithmes sont significativement différents. Pour affirmer cela, nous effectuons d'abord un test statistique de Lilliefors [Lilliefors, 1967] sur les nombres d'évaluations effectuées par chaque algorithme. Ce test nous indique, à un niveau de confiance de 95%, que les nombres d'évaluations suivent une loi normale. Nous pouvons ensuite réaliser un test d'analyse de la variance (*ANOVA de Welch*, [Welch, 1951]) sur ces nombres d'évaluations. Ce test confirme, à un niveau de confiance de 95%, qu'il existe une différence significative entre au moins deux des algorithmes testés. Nous effectuons alors une procédure de comparaisons multiples de Tukey-Kramer [Tukey, 1994; Kramer, 1957] pour déterminer quels algorithmes sont significativement différents, en termes de nombre d'évaluations. Il apparaît que la performance de chaque algorithme est significativement différente de celle des autres. Nous montrons ainsi,

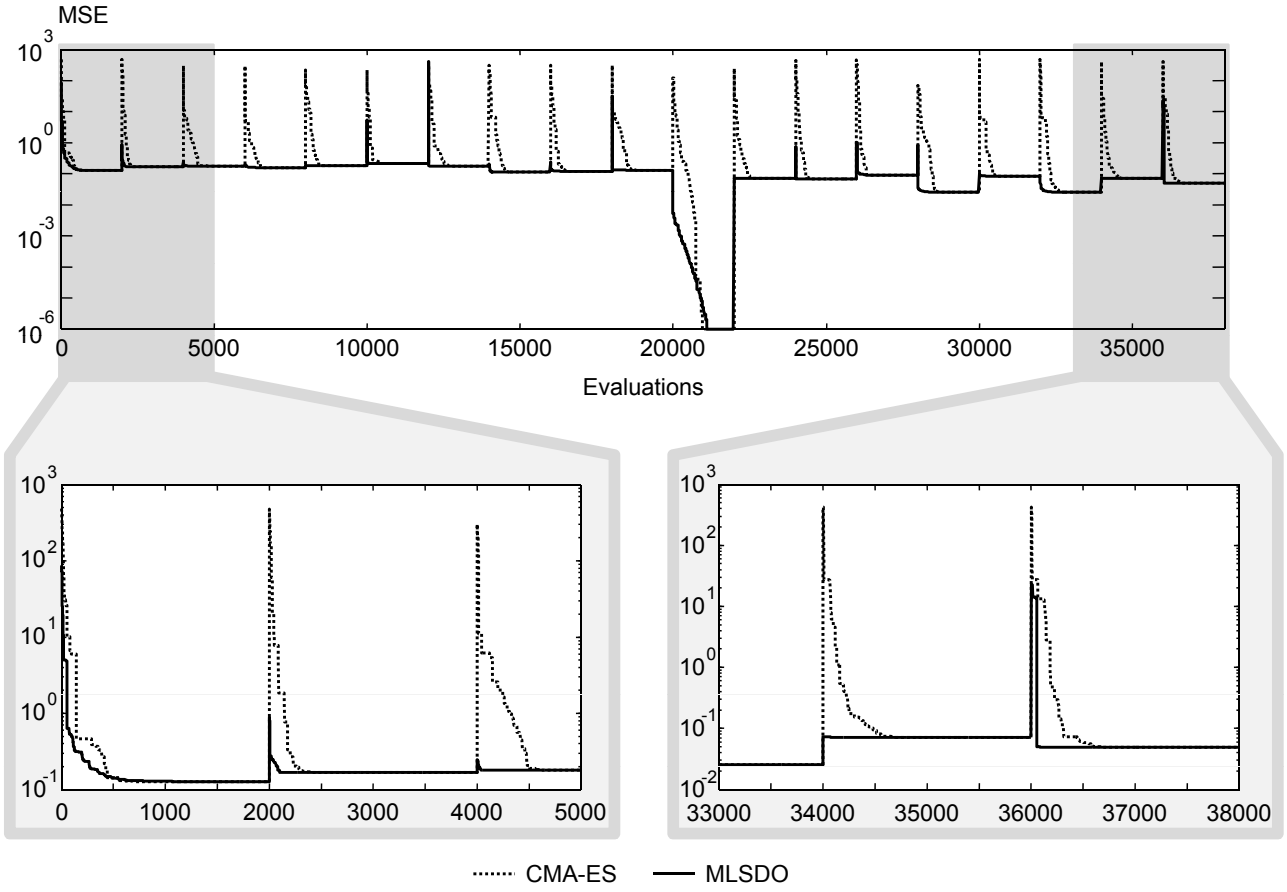


FIGURE 5.6 – Graphe de convergence de MLSDO et CMA-ES.

en particulier, que MLSDO consomme significativement moins d'évaluations que les algorithmes d'optimisation statique testés. Nous constatons également, à la figure 5.6, que la convergence de MLSDO vers une solution acceptable est plus rapide que celle de CMA-ES, pour le recalage de tous les couples de contours, excepté le premier. En effet, au recalage du premier couple, MLSDO accumule des informations sur la fonction objectif, afin d'accélérer sa convergence lors du recalage des couples suivants. Ainsi, cette comparaison montre l'efficacité de MLSDO et l'intérêt d'utiliser, pour ce problème, un algorithme d'optimisation dynamique.

5.4 Recalage de séquences d'images ciné-IRM non segmentées

Au paragraphe précédent, nous avons utilisé une transformation affine comme modèle de déformation pour que ses paramètres puissent être facilement interprétés. En effet, nous pouvons ainsi modéliser les déformations de la *lamina terminalis* par une rotation, une translation et une homothétie. Cette modélisation pourrait toutefois être améliorée en utilisant un modèle

de déformation non linéaire. Ce dernier permettrait d'améliorer la précision de notre méthode de quantification de mouvements. L'augmentation de la précision de la méthode permet une meilleure quantification des mouvements, et donc un meilleur diagnostic. Ainsi, nous proposons d'utiliser une transformation polynomiale d'ordre 2 pour modéliser les déformations de la *lamina terminalis*. Il s'agit d'un modèle de déformation non linéaire à 12 degrés de liberté (pour des images 2D), ce qui constitue un bon compromis entre précision et temps de calcul.

De plus, afin d'éliminer l'étape de segmentation, nous proposons d'utiliser une approche iconique. A cet effet, nous devons utiliser un critère de similarité adapté au type d'images traitées. Dans notre cas, il s'agit d'images ciné-IRM assez bruitées, pour lequelles l'intensité des pixels d'un même objet peut varier au cours du temps. Le critère utilisé doit alors être le moins sensible possible aux changements d'intensité. De ce fait, nous proposons d'utiliser un critère de similarité basé sur l'*information mutuelle normalisée* [Studholme et al., 1999]. L'information mutuelle et sa variante normalisée sont largement utilisées en recalage d'images [Pluim et al., 2003], du fait qu'aucune hypothèse autre que statistique n'est nécessaire sur la relation entre les intensités des images.

Nous proposons ainsi une nouvelle procédure de recalage, que nous allons maintenant décrire et analyser. Comme précédemment, nous allons montrer l'intérêt d'utiliser, dans cette procédure, un algorithme d'optimisation dynamique tel que MLSDO.

5.4.1 Procédure de recalage

Soient Im_1 et Im_2 deux images successives de la séquence à recaler. En notant A^{T} la transposée d'une matrice A, nous supposons qu'il existe une transformation T_Φ permettant de mettre en correspondance Im_1 avec $Im'_1 = T_\Phi(Im_2)$. Pour chaque pixel $o_2 = (x_2 \;\; y_2)^{\mathrm{T}}$ de Im_2, cette transformation est définie par :

$$\begin{aligned} x'_1 &= c_1\, x_2^2 \;+\; c_2\, y_2^2 \;+\; c_3\, x_2\, y_2 \;+\; (c_4\;|c_4| + 1)\, x_2 \;+\; c_5\;|c_5|\; y_2 \;+\; (c_6)^3 \\ y'_1 &= c_7\, x_2^2 \;+\; c_8\, y_2^2 \;+\; c_9\, x_2\, y_2 \;+\; c_{10}\;|c_{10}|\; x_2 \;+\; (c_{11}\;|c_{11}| + 1)\, y_2 \;+\; (c_{12})^3 \end{aligned} \tag{5.4.1}$$

où $o'_1 = (x'_1 \;\; y'_1)^{\mathrm{T}} = T_\Phi(o_2)$. L'ensemble des paramètres $\Phi = \{c_1, c_2, ..., c_{12}\}$ est estimé en maximisant le critère suivant :

$$C(\Phi) = \frac{NMI(\Phi)}{P(\Phi) + 1} \tag{5.4.2}$$

où $NMI(\Phi)$ est l'information mutuelle normalisée de Im_1 et Im'_1 ; $P(\Phi)$ fait partie d'un terme de régularisation qui pénalise les grandes déformations de Im_2, du fait que nous sommes en

présence de faibles mouvements dans la région d'intérêt. Par ailleurs, comme la taille de la région d'intérêt n'est pas constante pour chaque expérience, nous proposons de normaliser les coordonnées des pixels dans l'intervalle $[-0{,}5; 0{,}5]$. L'utilisation de cet intervalle transforme les coordonnées discrètes des pixels en coordonnées continues, selon l'expression (5.4.3). Cet intervalle a été déterminé empiriquement, il est bien adapté au terme de régularisation et au modèle de déformation utilisés. Parmi un ensemble d'intervalles possibles, nous avons sélectionné celui permettant d'obtenir les meilleurs résultats, en termes de précision et de temps de calcul. $NMI(\Phi)$ et $P(\Phi)$ sont définis en (5.4.4) et (5.4.5), respectivement.

$$\begin{aligned} x &= \tfrac{X}{S_X - 1} - 0{,}5 \\ y &= \tfrac{Y}{S_Y - 1} - 0{,}5 \end{aligned} \tag{5.4.3}$$

où S_X et S_Y sont respectivement la largeur et la hauteur de la région d'intérêt ; $X \in \{1, 2, ..., S_X - 1\}$ et $Y \in \{1, 2, ..., S_Y - 1\}$ sont les coordonnées discrètes d'un pixel ; x et y sont les coordonnées normalisées de ce pixel dans $[-0{,}5; 0{,}5]$.

$$NMI(\Phi) = \frac{H(Im_1) + H(Im_1')}{H(Im_1,\ Im_1')} \tag{5.4.4}$$

$$P(\Phi) = \max_{o_2 \in Im_1 \cap Im_1'} (o_2 - o_1')^{\mathrm{T}} (o_2 - o_1') \tag{5.4.5}$$

où $Im_1 \cap Im_1'$ est la *zone de recouvrement* de Im_1 et de Im_1' (telle qu'illustrée à la figure 5.7) ; $H(Im_1)$ et $H(Im_1')$ calculent l'entropie de Shannon de Im_1 et de Im_1', respectivement, dans leur zone de recouvrement ; $H(Im_1,\ Im_1')$ calcule l'entropie jointe de Shannon de Im_1 et de Im_1', dans leur zone de recouvrement. Ces entropies sont définies comme suit :

$$H(Im_1) = -\sum_{i=0}^{L-1} p(i)\ \log_2(p(i)) \tag{5.4.6}$$

$$H(Im_1') = -\sum_{j=0}^{L-1} p'(j)\ \log_2(p'(j)) \tag{5.4.7}$$

$$H(Im_1,\ Im_1') = -\sum_{i=0}^{L-1}\sum_{j=0}^{L-1} p(i,j)\ \log_2(p(i,j)) \tag{5.4.8}$$

où L est le nombre total de niveaux de gris (ou valeurs d'intensité) qu'un pixel peut prendre ; $p(i)$ est la probabilité d'avoir un pixel d'intensité i dans Im_1 ; $p'(j)$ est la probabilité d'avoir un

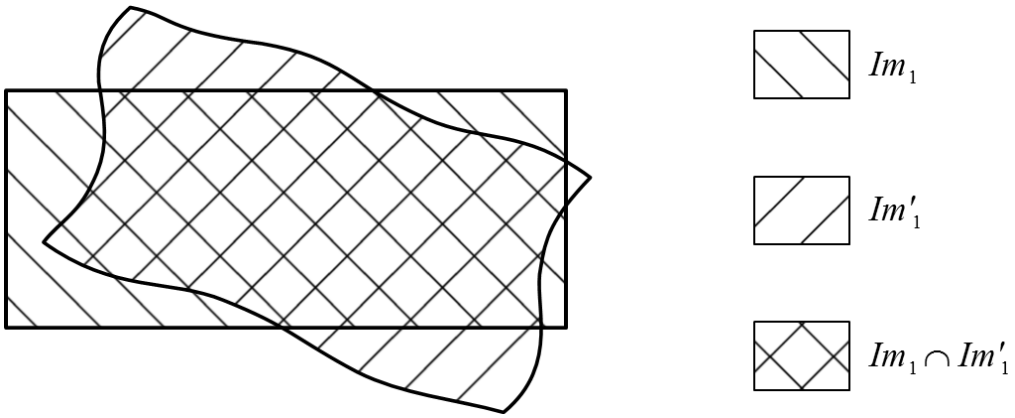


FIGURE 5.7 – Zone de recouvrement ($Im_1 \cap Im'_1$) de l'image source (Im_1) et de l'image cible transformée (Im'_1).

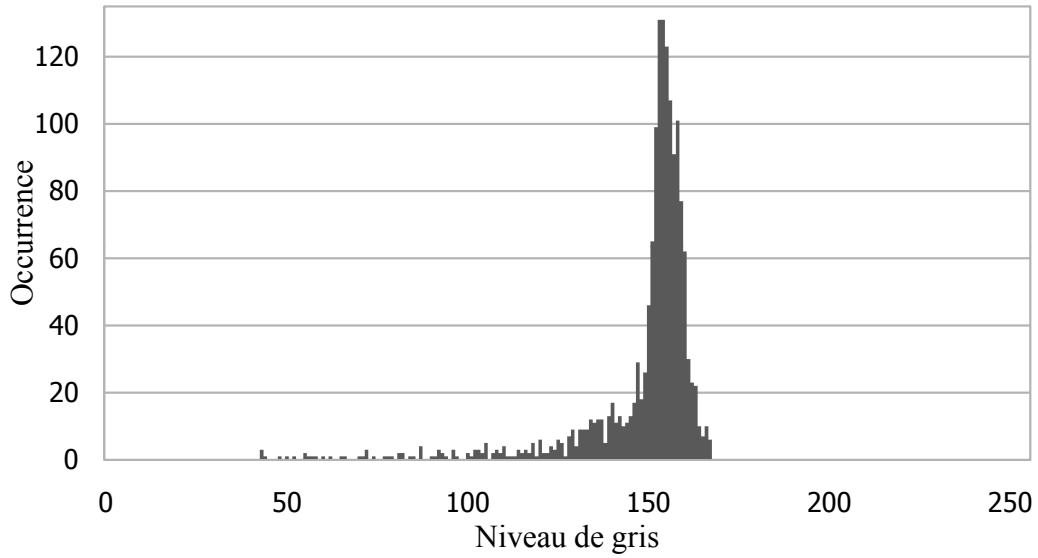
pixel d'intensité j dans Im'_1 ; $p(i,j)$ est la probabilité jointe d'avoir un pixel d'intensité i dans Im_1 et j dans Im'_1. Ces probabilités sont définies comme suit :

$$p(i) = \frac{g\,(i)}{\sum_{k=0}^{L-1} g\,(k)} \tag{5.4.9}$$

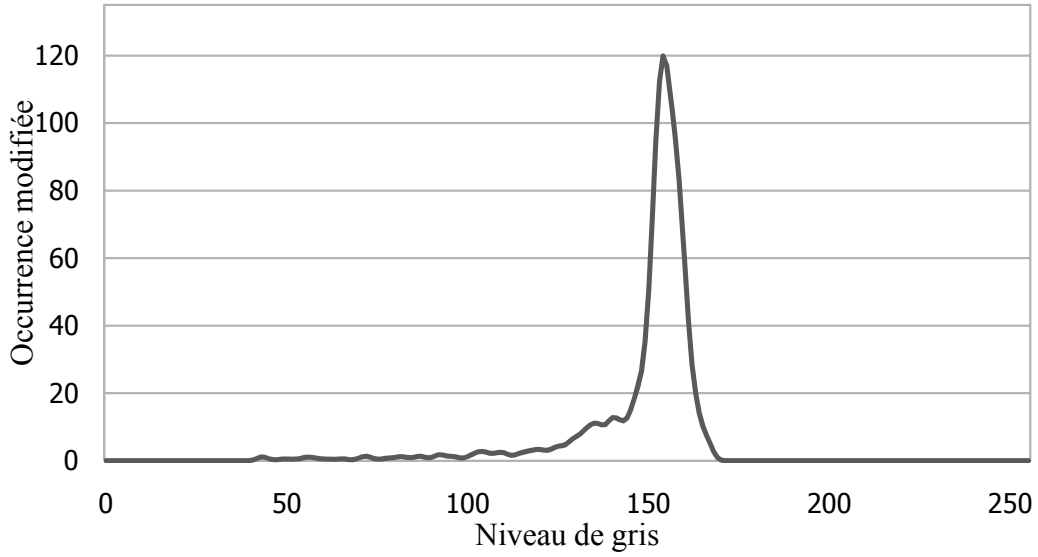
$$p'(j) = \frac{g'\,(j)}{\sum_{l=0}^{L-1} g'\,(l)} \tag{5.4.10}$$

$$p(i,j) = \frac{g\,(i,j)}{\sum_{k=0}^{L-1}\sum_{l=0}^{L-1} g\,(k,l)} \tag{5.4.11}$$

où $g(i)$ est l'histogramme de la zone de recouvrement de Im_1 (nombre d'occurrences du niveau de gris i dans Im_1) ; $g'(j)$ est l'histogramme de la zone de recouvrement de Im'_1 (nombre d'occurrences du niveau de gris j dans Im'_1) ; $g(i,j)$ est l'histogramme joint de la zone de recouvrement de Im_1 et de Im'_1 (nombre d'occurrences d'avoir à la fois un niveau de gris i dans Im_1 et j dans Im'_1, voir (5.4.12)). Néanmoins, dans notre cas, nous utilisons une technique de fenêtre de Parzen [Parzen, 1962] avec une fonction à noyau Gaussien d'écart-type 1,4 (taille de la fenêtre), afin de « lisser » ces histogrammes. Cette technique sert à estimer l'histogramme d'une image, à partir d'un sous-ensemble seulement de ses pixels. Dans notre cas, tous les pixels de l'image sont utilisés, notre but n'étant pas d'estimer, mais de lisser l'histogramme. L'utilisation de cette technique permet d'accélérer le processus d'optimisation (le nombre d'évaluations de la fonction objectif est réduit d'environ 10%). En effet, nous réduisons ainsi le nombre d'optima locaux dans la fonction objectif, en la lissant. L'histogramme de la région d'intérêt, extraite d'une séquence ciné-MRI, ainsi que son histogramme lissé correspondant, sont illustrés à la figure 5.8.



(a)



(b)

FIGURE 5.8 – Illustration de l'histogramme de la région d'intérêt : (a) histogramme original, (b) histogramme lissé utilisé pour accélérer le processus d'optimisation.

Dans (5.4.12), la fonction cardinale est notée *card*, et les fonctions $Im_1(o)$ et $Im'_1(o)$ retournent les valeurs de niveaux de gris d'un pixel donné o dans Im_1 et dans Im'_1, respectivement.

$$g(i,j) = \text{card}\,\{o \in Im_1 \cap Im'_1,\ \ Im_1(o) = i\ \wedge\ Im'_1(o) = j\} \quad \text{pour } i = 0,...,L-1 \text{ et } j = 0,...,L-1 \tag{5.4.12}$$

5.4.2 Formulation sous la forme d'un problème d'optimisation dynamique

Le recalage d'une séquence ciné-IRM peut être vu comme un problème d'optimisation dynamique, pour lequel la fonction objectif, optimisée par MLSDO, change, selon les règles suivantes :

– Comme nous avons à traiter une séquence, le critère (5.4.2) doit être maximisé pour chaque couple d'images. Le critère d'optimisation devient alors :

$$C(\Phi(t)) = \frac{NMI(\Phi(t))}{P(\Phi(t)) + 1} \tag{5.4.13}$$

où t est l'indice du couple d'images en cours de recalage dans la séquence. $\Phi(t)$, $NMI(\Phi(t))$ et $P(\Phi(t))$ correspondent respectivement à Φ, $NMI(\Phi)$ et $P(\Phi)$ définis précédemment, mais dépendent ici du couple d'images en cours de recalage.

– Ainsi, le problème d'optimisation dynamique est défini par :

$$\max\ C(\Phi(t)) \tag{5.4.14}$$

– Si la meilleure solution (transformation) courante trouvée pour le couple t ne peut être améliorée davantage (selon un critère de stagnation), alors le couple suivant $(t+1)$ est traité.

– Le critère de stagnation utilisé est satisfait si aucune amélioration significative (supérieure à 1E−5) n'est observée pour la meilleure solution courante, durant 5000 évaluations successives de la fonction objectif.

– Ainsi, la fin du recalage d'un couple d'images et le début du recalage du couple suivant constituent un changement dans la fonction objectif.

5.4.3 Résultats et discussion

Les valeurs optimales des paramètres de MLSDO utilisées pour ce problème ont été trouvées empiriquement, en les ajustant manuellement, de façon à minimiser le nombre d'évaluations nécessaires pour obtenir un recalage satisfaisant. Ces valeurs sont données dans le tableau 5.8.

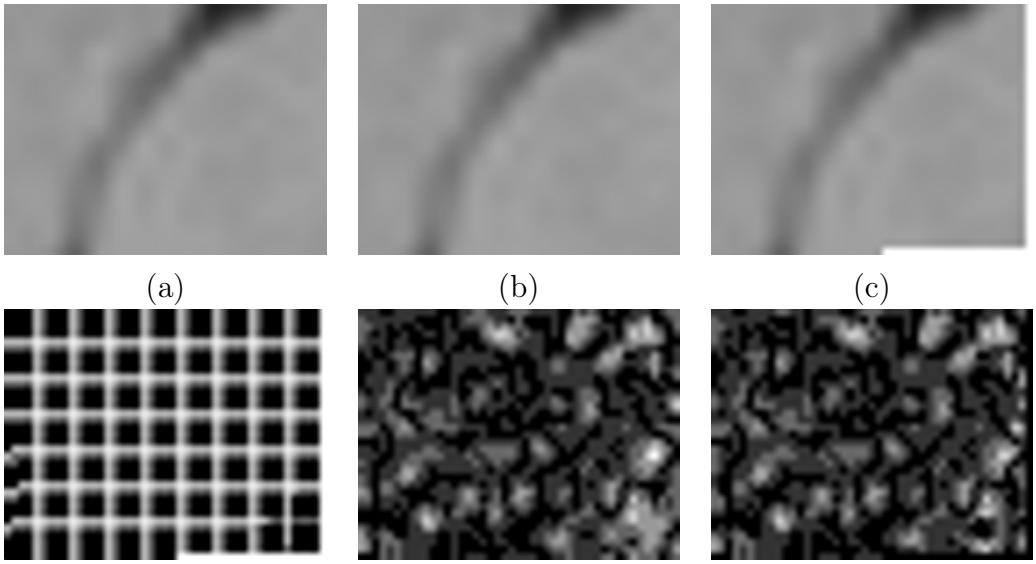
Paramètre	n_a	n_c	r_l	r_e	δ_{ph}	δ_{pl}
Valeur	1	2	5E−3	1E−1	1E−5	1E−4

TABLEAU 5.8 – Paramétrage de MLSDO.

Les recalages de deux couples d'images présentant une différence légère sont illustrés aux figures 5.9 et 5.10. Ceux d'images significativement différentes sont illustrés aux figures 5.11 et 5.12. Nous constatons, aux figures 5.9(e) et 5.9(f), comme aux figures 5.10(e) et 5.10(f), que si les mouvements dans la région d'intérêt ne sont pas significatifs, alors seul du bruit apparaît dans les images différences. Ainsi, les transformations utilisées pour recaler les couples d'images (figures 5.9(d) et 5.10(d)) ne déforment pas significativement la deuxième image des couples. En revanche, des mouvements significatifs dans la région d'intérêt laissent une importante trace blanche dans les images différences, comme illustré aux figures 5.11(e) et 5.12(e). Une transformation plus importante (figures 5.11(d) et 5.12(d)) doit être appliquée, afin d'éliminer cette trace blanche (voir figures 5.11(f) et 5.12(f)).

Comme dans le paragraphe 5.3.4, nous allons maintenant comparer les résultats obtenus par MLSDO à ceux d'algorithmes d'optimisation statique, connus pour ce problème. Le but est de montrer l'intérêt d'utiliser pour ce problème un algorithme d'optimisation dynamique, tel que MLSDO. Les algorithmes statiques utilisés sont les mêmes qu'au paragraphe 5.3.4. Leurs paramètres, ajustés empiriquement pour ce problème, de la même manière que pour MLSDO, sont présentés ci-dessous :

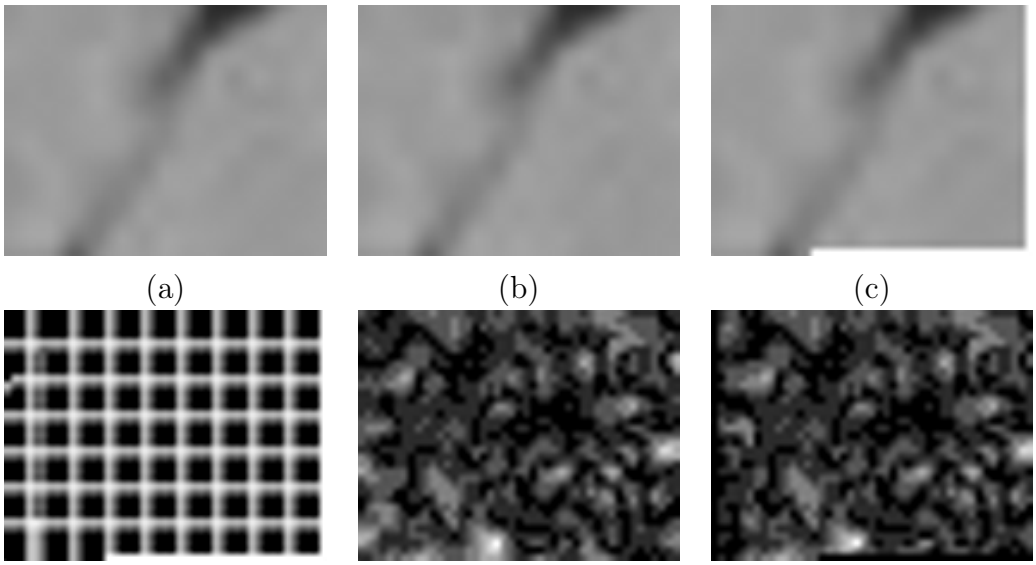
- CMA-ES, en utilisant le paramétrage recommandé, à l'exception du pas initial σ, fixé à $\sigma = 0{,}5$. La taille de la population d'enfants λ et le nombre d'individus sélectionnés μ sont fixés à $\lambda = 11$ et $\mu = 5$.
- SPSO-07, en utilisant le paramétrage recommandé, à l'exception du nombre S de particules ($S = 12$) et du paramètre K utilisé pour engendrer le voisinage des particules ($K = 8$).
- DE en utilisant la stratégie « DE/target-to-best/1/bin », un nombre de parents $NP = 30$, un facteur d'amplification $F = 0{,}8$ et une constante de croisement $CR = 0{,}9$.



(a) (b) (c)

(d) (e) (f)

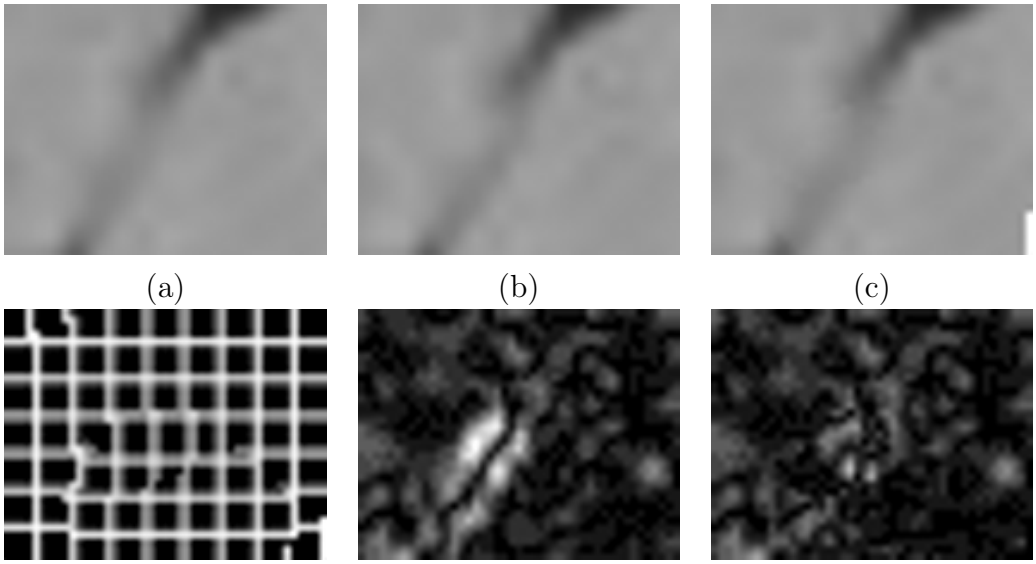
FIGURE 5.9 – Illustration du recalage d'un couple d'images présentant une différence « légère » : (a) première image du couple, (b) deuxième image du couple, (c) deuxième image après application de la transformation trouvée, (d) illustration de la transformation appliquée à la deuxième image d'un couple pour le recaler, (e) illustration de la différence, dans les intensités des pixels, entre les deux images d'un couple : un pixel noir indique que les intensités des pixels correspondants sont identiques dans les deux images, alors qu'un pixel blanc indique la plus grande différence entre les images, (f) illustration de la différence, dans les intensités des pixels, entre la première image et la deuxième image transformée.



(a) (b) (c)

(d) (e) (f)

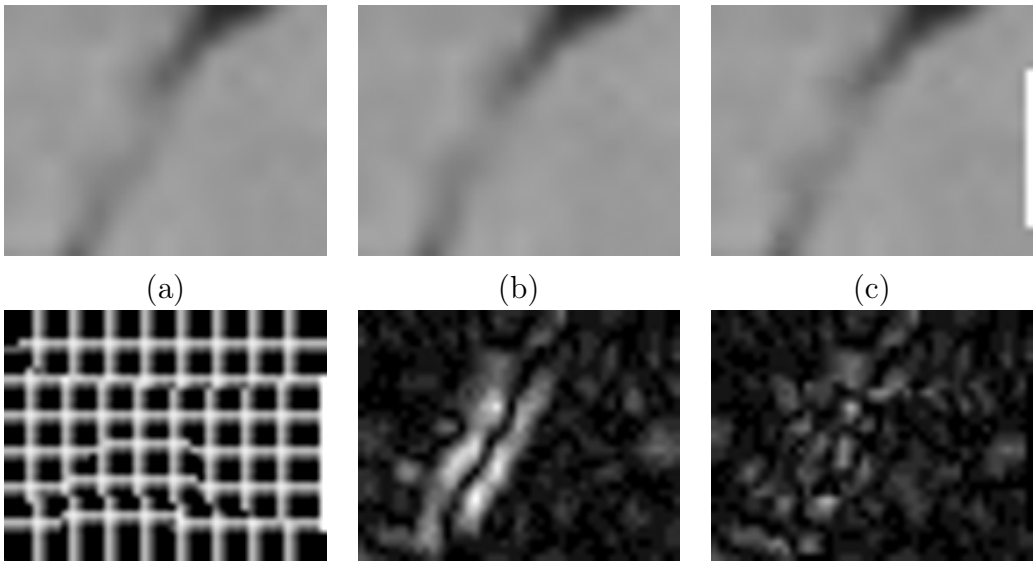
FIGURE 5.10 – Illustration du recalage d'un autre couple d'images présentant une différence « légère », comme à la figure 5.9.



(a) (b) (c)

(d) (e) (f)

FIGURE 5.11 – Illustration du recalage d'un couple d'images « significativement » différentes : (a) première image du couple, (b) deuxième image du couple, (c) deuxième image après application de la transformation trouvée, (d) illustration de la transformation appliquée à la deuxième image d'un couple pour le recaler, (e) illustration de la différence, dans les intensités des pixels, entre les deux images d'un couple : un pixel noir indique que les intensités des pixels correspondants sont identiques dans les deux images, alors qu'un pixel blanc indique la plus grande différence entre les images, (f) illustration de la différence, dans les intensités des pixels, entre la première image et la deuxième image transformée.



(a) (b) (c)

(d) (e) (f)

FIGURE 5.12 – Illustration du recalage d'un autre couple d'images « significativement » différentes, comme à la figure 5.11.

Comme ce sont des algorithmes d'optimisation statique, nous devons considérer le recalage de chaque couple d'images successives comme un nouveau problème à résoudre. De ce fait, chacun de ces algorithmes est redémarré une fois le recalage d'un couple d'images terminé, selon le critère de stagnation défini au paragraphe 5.4.2. Les résultats obtenus par MLSDO, utilisé en tant qu'algorithme d'optimisation statique (i.e. redémarré après le recalage de chaque couple d'images), sont également donnés.

Les paramètres du modèle de déformation trouvés sont donnés dans le tableau 5.9. Le nombre d'évaluations effectuées par chaque algorithme, moyenné sur 20 exécutions, est donné dans le tableau 5.10. La valeur moyenne, sur 20 exécutions, de la fonction objectif des meilleures solutions trouvées pour recaler chaque couple d'images est également donnée. La complexité de la méthode de recalage, en utilisant chacun des algorithmes testés, est aussi donnée dans ce tableau. Les convergences de MLSDO et de l'algorithme le plus performant pour ce problème, CMA-ES, sont illustrées à la figure 5.13. Cette figure montre l'évolution de l'erreur relative $\left(\frac{C^*(\Phi(t))-C(\Phi(t))}{C^*(\Phi(t))}\right)$ entre la valeur de la meilleure solution trouvée ($C^*(\Phi(t))$) et celle de la solution courante ($C(\Phi(t))$), pour chaque couple d'images (t). L'évolution de cette erreur relative (pouvant être considérée comme une mesure de la stagnation de l'algorithme) donne une idée de la convergence des algorithmes. Dans cette figure, le nombre d'évaluations par recalage d'un couple d'images est fixé à 5000, afin de pouvoir comparer la convergence des algorithmes. Pour plus de clarté, nous utilisons une échelle logarithmique pour l'axe des ordonnées.

Les valeurs moyennes de la fonction objectif des meilleures solutions trouvées, données dans le tableau 5.10, nous indiquent que les algorithmes ont une précision moyenne similaire. Néanmoins, nous constatons dans ce tableau que le nombre d'évaluations effectuées par MLSDO, utilisé en tant qu'algorithme d'optimisation dynamique, est significativement inférieur à ceux des algorithmes d'optimisation statique. Pour affirmer cela, nous effectuons d'abord un test statistique de Lilliefors sur les nombres d'évaluations effectuées par chaque algorithme. Ce test nous indique, à un niveau de confiance de 95%, que les nombres d'évaluations suivent une loi normale. Nous pouvons ensuite réaliser un test d'analyse de la variance (*ANOVA de Welch*) sur ces nombres d'évaluations. Ce test confirme, à un niveau de confiance de 95%, qu'il existe une différence significative entre au moins deux des algorithmes testés. Nous effectuons alors une procédure de comparaisons multiples de Tukey-Kramer, pour déterminer quels algorithmes sont significativement différents, en termes de nombres d'évaluations. Il apparaît que la performance de MLSDO, utilisé en tant qu'algorithme d'optimisation dynamique, est significativement meilleure que celles des autres algorithmes. Nous constatons également, dans la figure 5.13, que la convergence de MLSDO vers une solution acceptable est plus rapide que celle

t	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	$C^*(\Phi(t))$
1	0,039	-0,022	0,005	0,105	-0,034	0,139	-0,039	0,017	0,025	0,091	0,132	0,090	1,199
2	-0,005	-0,029	0,025	0,085	-0,014	0,203	0,077	0,055	0,051	0,068	-0,077	-0,264	1,201
3	0,055	0,063	0,048	0,094	-0,104	-0,239	0,068	0,000	0,000	-0,074	-0,083	-0,256	1,192
4	0,021	0,031	-0,001	0,095	-0,077	-0,223	0,025	0,013	0,006	0,081	-0,144	-0,246	1,195
5	0,063	0,000	0,003	-0,074	-0,026	-0,089	-0,026	0,041	0,011	0,100	0,145	-0,128	1,218
6	0,002	-0,063	-0,033	-0,115	0,034	0,224	-0,019	-0,027	0,024	0,015	0,087	0,258	1,209
7	0,013	-0,092	0,016	0,036	0,080	0,253	-0,060	-0,045	-0,033	-0,077	0,131	0,247	1,208
8	0,003	-0,068	-0,004	-0,023	0,117	0,238	-0,069	-0,047	-0,032	-0,078	0,131	0,247	1,195
9	0,065	-0,020	-0,007	0,044	0,061	-0,046	-0,064	-0,047	-0,023	-0,081	0,131	0,251	1,201
10	0,050	-0,004	-0,017	0,072	0,056	-0,061	0,051	0,005	0,011	-0,052	0,135	-0,043	1,216
11	0,050	0,000	-0,012	-0,004	0,073	-0,053	-0,059	0,047	0,002	0,099	0,164	-0,178	1,216
12	0,060	0,011	0,003	0,080	-0,033	-0,191	-0,024	0,032	0,036	-0,068	0,108	0,048	1,225
13	0,042	0,000	0,000	0,050	-0,018	-0,060	-0,023	0,016	0,002	-0,085	-0,064	-0,218	1,232
14	0,064	-0,005	0,000	0,094	-0,021	-0,199	-0,016	0,075	0,065	-0,039	0,065	-0,210	1,232
15	0,025	-0,008	0,042	0,049	-0,072	0,172	0,037	0,029	0,000	0,104	0,107	-0,037	1,235
16	0,060	0,007	0,003	0,082	-0,026	-0,191	-0,024	0,032	0,034	-0,063	0,111	-0,049	1,216
17	0,050	-0,005	0,000	0,021	0,010	-0,071	-0,025	0,047	0,052	0,018	0,080	-0,170	1,226
18	0,052	-0,005	-0,017	0,083	0,108	-0,121	-0,018	0,042	-0,001	0,071	0,075	0,149	1,225
19	-0,006	0,056	-0,011	-0,080	0,072	-0,210	-0,025	0,076	0,033	-0,057	0,084	-0,158	1,214

TABLEAU 5.9 – Transformations trouvées pour recaler les couples d'images. La valeur de la fonction objectif de la meilleure solution trouvée, notée $C^*(\Phi(t))$, est également donnée.

	Algorithme	Evaluations	$\sum_{t=1}^{19} \frac{C^*(\Phi(t))}{19}$	Complexité
Optimisation dynamique	MLSDO	6880,68 ± 585,92	1,21 ± 7,0E−4	$O(n\ d^3)$
Optimisation statique	CMA-ES	7709,14 ± 467,75	1,21 ± 9,1E−4	$O(n\ d^2)$
	SPSO-07	8007,21 ± 364,24	1,21 ± 8,8E−4	$O(n\ d)$
	DE	9131,25 ± 279,20	1,21 ± 9,3E−4	$O(n\ d)$
	MLSDO	9522,76 ± 648,87	1,21 ± 1,7E−3	$O(n\ d^3)$

TABLEAU 5.10 – Nombre moyen d'évaluations pour recaler un couple d'images, et valeur moyenne de $C^*(\Phi(t))$, obtenus pour chaque algorithme. La complexité de la méthode de recalage, en utilisant chaque algorithme, est également donnée, en désignant par n le nombre d'images de la séquence et par d la dimension de l'espace de recherche.

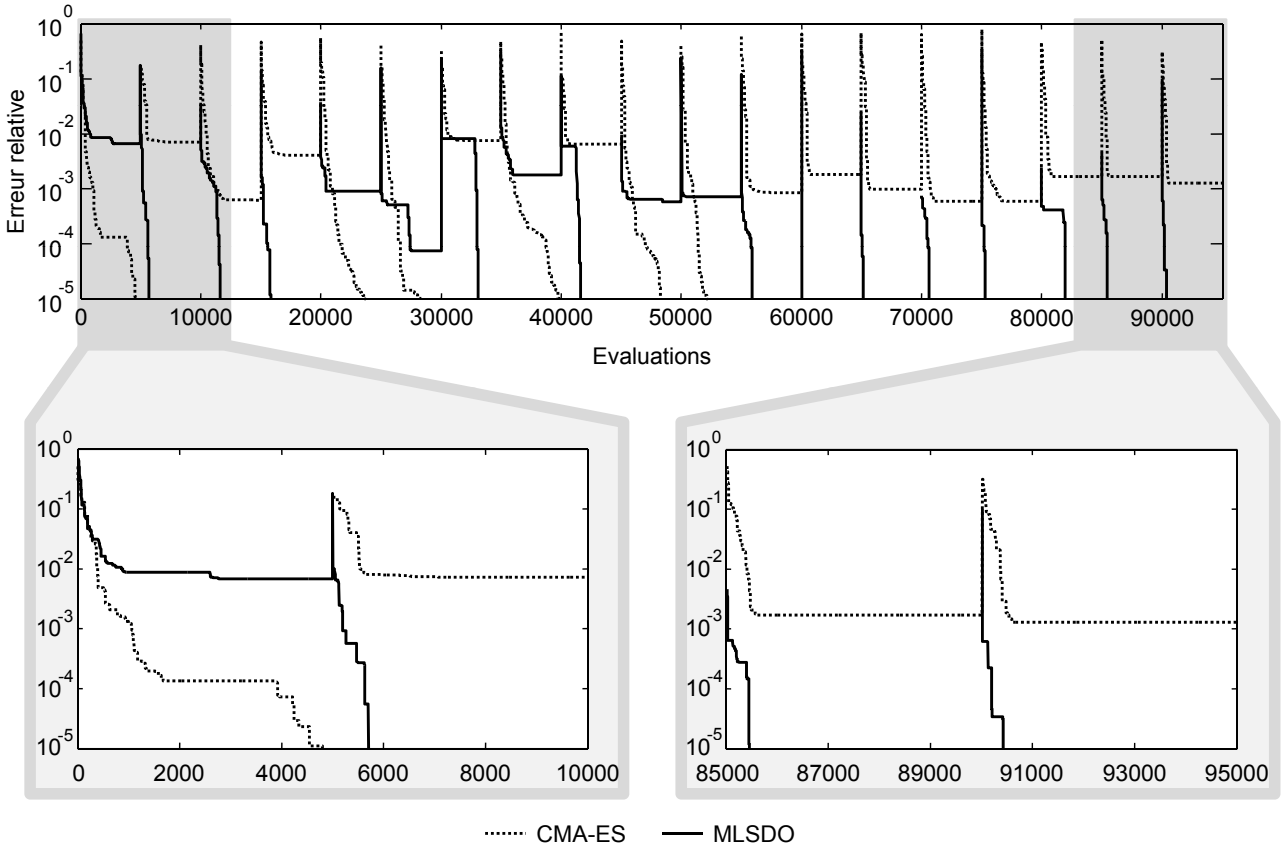


FIGURE 5.13 – Graphe de convergence de MLSDO et CMA-ES.

de CMA-ES, pour le recalage de la plupart des couples d'images, en particulier pour les derniers de la séquence. En effet, comme nous l'avons déjà remarqué, lors des premiers recalages, MLSDO accumule des informations sur la fonction objectif afin d'accélérer sa convergence, lors des recalages suivants. Ainsi, comme au paragraphe 5.3.4, cette comparaison montre l'efficacité de MLSDO et l'intérêt d'utiliser un algorithme d'optimisation dynamique, sur un problème tel que celui-ci.

5.5 Conclusion

Dans ce chapitre, nous avons proposé une procédure de recalage utilisant l'algorithme d'optimisation dynamique MLSDO. Cette procédure est utilisée au sein d'une méthode de quantification des mouvements des parois du troisième ventricule cérébral. Il s'agit d'une méthode automatique, traitant séquentiellement les images d'une séquence ciné-IRM afin d'estimer avec précision ces mouvements. L'utilisation de MLSDO permet ainsi à cette méthode de bénéficier de l'efficacité du paradigme de l'optimisation dynamique. Dans un premier temps, nous avons montré l'intérêt d'utiliser MLSDO au sein de cette procédure de recalage, afin de recaler

plus rapidement une séquence d'images ciné-IRM segmentées. Dans un deuxième temps, une amélioration de la précision de cette procédure, par le biais d'un recalage iconique basé sur un modèle de déformation non linéaire, est proposée. Ainsi, nous supprimons également l'étape de segmentation de la méthode de quantification de mouvements. De plus, comme MLSDO n'est pas conçu spécifiquement pour ce problème, mais pour permettre la résolution d'une grande variété de problèmes, la méthode que nous proposons peut aussi être appliquée à d'autres types de séquences.

Conclusion et perspectives

Dans nos travaux, un nouvel algorithme d'optimisation dynamique, nommé MADO, a été proposé. Pour explorer l'espace de recherche, MADO utilise une population d'agents effectuant des recherches locales coordonnées. Les meilleurs optima locaux trouvés par ces agents sont stockés en mémoire, afin d'accélérer la convergence de l'algorithme. La coordination des agents permet d'explorer au mieux l'espace de recherche, et de garantir la diversité des solutions. La collaboration directe entre les agents se fait essentiellement par l'attribution, à chaque agent, d'une zone de recherche locale exclusive. La détection d'un changement dans la fonction objectif se fait par réévaluation d'une solution. Lorsqu'un changement est détecté, les solutions courantes des agents et les optima de l'archive sont réévalués. Puis, des agents sont créés et positionnés sur les optima de l'archive. Une fois ces nouveaux agents créés, l'archive d'optima est vidée. Cela permet, au fil des changements dans la fonction objectif, de suivre les optima locaux trouvés auparavant.

Cet algorithme obtient de bons résultats sur les deux principaux jeux de tests (MPB et GDBG) en optimisation dynamique. Deux variantes de l'algorithme, nommées CMADO et PMADO, ont également été proposées. Chacune d'elles intègre une technique permettant d'améliorer les performances de l'algorithme. Ensuite, MADO a servi de base pour proposer un nouvel algorithme encore plus performant, nommé MLSDO. Ce nouvel algorithme obtient des résultats qui dépassent de manière très significative ceux obtenus par MADO sur les deux principaux jeux de tests en optimisation dynamique (MPB et GDBG). De plus, il se positionne parmi les algorithmes les plus performants sur ces jeux de tests :

- MLSDO est en tête du classement sur sept algorithmes évalués sur GDBG ;
- MLSDO est à la deuxième place sur seize algorithmes évalués sur MPB.

A la différence de MPB, GDBG est un jeu de tests plus récent, inspiré de MPB et composé de plusieurs fonctions dynamiques de natures différentes. La diversité des fonctions qui le composent lui permettent, à notre sens, d'être plus représentatif des problèmes réels que MPB. Néanmoins, ces jeux de tests restent complémentaires.

Par la suite, CMADO et MLSDO ont été appliqués à des problèmes de segmentation et de

recalage de séquences d'images médicales, afin de les résoudre plus rapidement. Il s'agit d'un travail inovant, du fait qu'il est inédit de résoudre ces problèmes au moyen de l'optimisation dynamique. Ces problèmes ont été abordés au sein d'un problème de quantification du mouvement d'un ventricule cérébral dans une séquence d'images ciné-IRM. Tout d'abord, l'algorithme CMADO a été utilisé au sein d'une méthode de segmentation par seuillage. Le critère de segmentation que nous avons proposé d'utiliser repose sur l'approximation de l'histogramme par une somme de fonctions gaussiennes. Ensuite, MLSDO a été employé dans une procédure de recalage, permettant de modéliser les déformations du ventricule cérébral au cours du temps, et ainsi, de quantifier son déplacement maximum. Dans un deuxième temps, une amélioration de la précision de cette procédure, au moyen d'une approche de recalage iconique et d'un modèle de déformation non-linéaire, a été proposée. Les résultats obtenus montrent l'intérêt d'utiliser ces algorithmes sur ce type de problèmes.

La méthode de quantification de mouvements proposée est automatique, à l'exception de la sélection de la région d'intérêt dans la première image de la séquence. Elle permet d'évaluer avec précision l'amplitude des mouvements d'un ventricule cérébral, afin d'aider notamment au diagnostic et au traitement de l'hydrocéphalie.

En perspective, pour PMADO, une analyse expérimentale plus poussée reste à faire. Cette variante intègre une technique prédictive permettant de guider les recherches locales des agents vers des zones prometteuses de l'espace de recherche, au moyen d'un réseau de neurones. La variante actuelle de PMADO obtient des résultats encourageants, mais de nombreuses pistes d'amélioration de la technique proposée restent à explorer. Notamment, les performances de la prédiction par réseau de neurones pourraient être améliorées par l'utilisation d'un ensemble d'apprentissage filtré : plutôt que d'utiliser directement l'historique A_p des solutions testées comme ensemble d'apprentissage du réseau de neurones, il pourrait être bénéfique d'en filtrer certaines solutions. En particulier, filtrer les solutions « trop proches » les unes des autres pourrait améliorer l'apprentissage de la fonction objectif par le réseau de neurones. En effet, plus un agent converge vers un optimum local, plus les déplacements qui le mènent vers cet optimum diminuent en amplitude (intensification). Des amas de solutions, situées autour d'un optimum local, sont alors archivés dans A_p. Ces amas de solutions peuvent perturber l'apprentissage du réseau de neurones. Filtrer ces solutions (par exemple, en les remplacant par la meilleure d'entre elles), devrait permettre d'améliorer les performances de la technique prédictive.

En perspective également, les paramètres de l'algorithme peuvent être auto-adaptés, afin de faciliter son utilisation. Des travaux ont déjà été initiés durant cette thèse afin d'adapter automatiquement, au cours de l'exécution de CMADO, les paramètres de l'algorithme [Lepagnot

et al., 2010d]. Cette technique permet de calculer, ou d'adapter à partir de valeurs par défaut, la valeur des paramètres de CMADO, en se basant sur des critères caractérisant le paysage de la fonction objectif (rugosité, conditionnement, sévérité des changements, redondance des détections des optima locaux, ...). Des tests sur MPB et sur GDBG montraient des résultats encourageants. Néanmoins, la technique proposée étant complexe, nous avons préféré interrompre les travaux de réglage automatique des paramètres de CMADO au profit de l'amélioration de l'algorithme.

Des travaux portant sur l'amélioration de la diversité des solutions archivées ont également été entrepris, bien qu'aucune amélioration des performances sur MPB et GDBG n'ait été enregistrée. L'amélioration de la diversité des optima locaux de l'archive, au moyen d'un critère d'archivage qui ne soit pas uniquement basé sur la valeur de la fonction objectif des solutions, mais aussi sur la distance entre ces solutions (favoriser l'archivage d'optima locaux éloignés des autres optima de l'archive), pourrait néanmoins être bénéfique sur certains problèmes.

De plus, les algorithmes proposés dans cette thèse sont mono-objectifs, c'est-à-dire qu'ils ne peuvent optimiser qu'une seule fonction objectif à la fois. Néanmoins, de nombreux problèmes réels d'optimisation dynamique nécessitent de pouvoir optimiser plusieurs fonctions objectifs simultanément. En perspective, il pourrait être intéressant de proposer une version multi-objectif de MLSDO. Plusieurs approches d'optimisation multi-objectif existent, les plus connues étant l'approche par agrégation et l'approche Pareto (basée sur la *dominance* et permettant d'obtenir, à la différence de l'approche par agrégation, un ensemble de solutions optimales, appelé *front de Pareto* [Dréo et al., 2003]). Comme MLSDO utilise déjà un archivage des meilleures solutions trouvées, et que l'approche Pareto est également basée sur l'utilisation d'une archive, MLSDO se prête donc déjà bien à une adaptation à l'optimisation multi-objectif par cette approche.

MLSDO étant basé sur l'utilisation de recherches locales, il sera alors nécessaire d'adapter ces recherches locales à l'optimisation multi-objectif. Une étude intéressante est réalisée dans [Liefooghe et al., 2009] pour déterminer, parmi plusieurs variantes d'une recherche locale multi-objectif basée sur la dominance, celle qui permet l'obtention du meilleur front de Pareto. Cette étude a été menée dans le cadre de l'optimisation statique combinatoire uniquement. Néanmoins, elle pourrait être étendue à bon escient à l'optimisation continue dynamique, sur plusieurs variantes multi-objectif des recherches locales utilisées par les agents de MLSDO.

Enfin, la précision de la procédure de recalage de séquences d'images, proposée au chapitre cinq, pourrait être améliorée, par l'utilisation d'un modèle de déformation plus complexe.

Références bibliographiques

[Abutableb, 1989] A. S. Abutableb. Automatic thresholding of gray-level pictures using two-dimensional entropy. *Computer Vision, Graphics, and Image Processing*, 47 (1): 22–32, 1989.

[Alpert et al., 1996] N. M. Alpert, D. Berdichevsky, Z. Levin, E. D. Morris, & A. J. Fischman. Improved methods for image registration. *NeuroImage*, 3(1): 10–18, 1996.

[Althouse & Chang, 1995] M. L. G. Althouse & C.-I. Chang. Image segmentation by local entropy methods. In *Proceedings of the IEEE International Conference on Image Processing*, pp. 61–64, Washington, DC, USA, October 1995.

[Ardekani et al., 1995] B. A. Ardekani, M. Braun, B. F. Hutton, I. Kanno, & H. Iida. A fully automatic multimodality image registration algorithm. *Journal of computer assisted tomography*, 19(4): 615–623, 1995.

[Barillot, 1999] C. Barillot. *Fusion de données et imagerie 3D en médecine*. Habilitation à diriger des recherches, Université de Rennes 1, September 1999.

[Besl & McKay, 1992] P. J. Besl & N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2): 239–256, 1992.

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

[Bird & Li, 2007] S. Bird & X. Li. Using regression to improve local convergence. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 592–599, Singapore, September 2007.

[Blackwell & Branke, 2004] T. Blackwell & J. Branke. Multi-swarm optimization in dynamic environments. *Lecture Notes in Computer Science*, 3005: 489–500, 2004.

[Blackwell & Branke, 2006] T. Blackwell & J. Branke. Multi-swarms, exclusion and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation*, 10 (4): 459–472, 2006.

[Branke, 1999] J. Branke. The Moving Peaks Benchmark website. http://www.aifb.uni-karlsruhe.de/∼jbr/MovPeaks, 1999.

[Branke et al., 2000] J. Branke, T. Kaußler, C. Schmidt, & H. Schmeck. A multi-population approach to dynamic optimization problems. In *Proceedings of Adaptive Computing in Design and Manufacturing*, pp. 299–308, Berlin, Germany, April 2000. Springer.

[Brest et al., 2009] J. Brest, A. Zamuda, B. Boskovic, M. S. Maucec, & V. Zumer. Dynamic optimization using self-adaptive differential evolution. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 415–422, Trondheim, Norway, May 2009.

[Brown, 1992] L. F. Brown. A survey of image registration techniques. *ACM Computing*

Surveys, 24(4): 325–376, 1992.

[Budoff et al., 2009] M. J. Budoff, N. Ahmadi, G. Sarraf, Y. Gao, D. Chow, F. Flores, & S. S. Mao. Determination of left ventricular mass on cardiac computed tomographic angiography. *Academic Radiology*, 16(6): 726–732, 2009.

[Buzug & Weese, 1998] T. M. Buzug & J. Weese. Voxel-based similarity measures for medical image registration in radiological diagnosis and image guided surgery. *Journal of Computing and Information Technology*, 6(2): 165–179, 1998.

[Chenoune et al., 2005] Y. Chenoune, E. Deléchelle, E. Petit, T. Goissen, J. Garot, & A. Rahmouni. Segmentation of cardiac cine-MR images and myocardial deformation assessment using level set methods. *Computerized Medical Imaging and Graphics*, 29(8): 607–616, 2005.

[Cheriet et al., 1998] M. Cheriet, J. N. Said, & C. Y. Suen. A recursive thresholding technique for image segmentation. *IEEE Transactions on Image Processing*, 7(6): 918–921, 1998.

[Clerc & Kennedy, 2002] M. Clerc & J. Kennedy. The particle swarm – explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1): 58–73, 2002.

[Clerc et al., 2011] M. Clerc et al. The *Particle Swarm Central* website. http://www.particleswarm.info, 2011.

[Collignon et al., 1995a] A. Collignon, F. Maes, D. Delaere, D. Vandermeulen, P. Suetens, & G. Marchal. Automated multi-modality image registration based on information theory. In *14th International Conference on Information Processing in Medical Imaging*, pp. 263–274, Ile de Berder, France, June 1995. Kluwer Academic Publishers.

[Collignon et al., 1995b] A. Collignon, D. Vandermeulen, P. Suetens, & G. Marchal. 3D multimodality medical image registration using space clustering. In *First International Conference on Computer Vision, Virtual Reality and Robotics in Medicine*, pp. 195–204, Nice, France, April 1995. Springer.

[Colorni et al., 1991] A. Colorni, M. Dorigo, & V. Maniezzo. Distributed optimization by ant colonies. In *Proceedings of the First European Conference on Artificial Life*, pp. 134–142, Paris, France, December 1991. MIT Press.

[Conway & Sloane, 1998] J. H. Conway & N. J. A. Sloane. *Sphere Packings, Lattices and Groups.* Springer, third edition, 1998.

[Cooren, 2008] Y. Cooren. *Perfectionnement d'un algorithme adaptatif d'Optimisation par Essaim Particulaire. Applications en génie médical et en électronique.* Thèse de doctorat, Université Paris-Est Créteil, Novembre 2008.

[Cuevas et al., 2010] E. Cuevas, D. Zaldivar, & M. Pérez-Cisneros. A novel multi-threshold segmentation approach based on differential evolution optimization. *Expert Systems with Applications*, 37(7): 5265–5271, 2010.

[Dasgupta, 1997] D. Dasgupta. Artificial neural networks and artificial immune systems : Similarities and differences. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pp. 873–878, Orlando, Florida, USA, October 1997.

[de Castro & Timmis, 2002] L. N. de Castro & J. Timmis. *Artificial Immune Systems : A New Computational Intelligence Approach.* Springer, first edition, 2002.

[de França & Zuben, 2009] F. O. de França & F. J. Von Zuben. A dynamic artificial immune algorithm applied to challenging benchmarking problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 423–430, Trondheim, Norway, May 2009.

[Dorigo & Blum, 2005] M. Dorigo & C. Blum. Ant colony optimization theory : a survey. *Theoretical Computer Science*, 344(2-3): 243–278, 2005.

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

[Dréo & Siarry, 2006] J. Dréo & P. Siarry. An ant colony algorithm aimed at dynamic continuous optimization. *Applied Mathematics and Computation*, 181(1): 457–467, 2006.

[Dréo et al., 2003] J. Dréo, A. Pétrowski, P. Siarry, & E. Taillard. *Métaheuristiques pour l'optimisation difficile*. Eyrolles, 2003.

[Du & Li, 2008] W. Du & B. Li. Multi-strategy ensemble particle swarm optimization for dynamic optimization. *Information Sciences*, 178(15): 3096–3109, 2008.

[Eiben & Smith, 2008] A. E. Eiben & J. E. Smith. *Introduction to Evolutionary Computing*. Springer, second edition, 2008.

[Fogel, 1962] L. J. Fogel. Autonomous automata. *Industrial Research Magazine*, 4(2): 14–19, 1962.

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

[Fraser, 1957] A. S. Fraser. Simulation of genetic systems by automatic digital computers. I. Introduction. *Australian Journal of Biological Science*, 10: 484–491, 1957.

[Gardeux et al., 2009] V. Gardeux, R. Chelouah, P. Siarry, & F. Glover. Unidimensional search for solving continuous high-dimensional optimization problems. In *Proceedings of the IEEE International Conference on Intelligent Systems Design and Applications*, pp. 1096–1101, Pisa, Italy, December 2009.

[Gasper & Collard, 1999] A. Gasper & P. Collard. From gas to artificial immune systems : improving adaptation in time dependent optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1859–1866, Washington, DC, USA, July 1999.

[Gonzalez & Woods, 2007] R. C. Gonzalez & R. E. Woods. *Digital image processing*. Prentice Hall, third edition, 2007.

[Goss et al., 1989] S. Goss, S. Aron, J. L. Deneubourg, & J. M. Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76(12): 579–581, 1989.

[Hansen & Ostermeier, 2001] N. Hansen & A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2): 159–195, 2001.

[Haykin, 2008] S. Haykin. *Neural Networks and Learning Machines*. Prentice Hall, third edition, 2008.

[Hellier & Barillot, 2003] P. Hellier & C. Barillot. Coupling dense and landmark-based approaches for nonrigid registration. *IEEE Transactions on Medical Imaging*, 22(2): 217–227, 2003.

[Hodel et al., 2009] J. Hodel, P. Decq, A. Rahmouni, S. Bastuji-Garin, A. Maraval, C. Combes,

B. Jarraya, C. Le Guérinel, & A. Gaston. Brain ventricular wall movement assessed by a gated cine MR trueFISP sequence in patients treated with endoscopic third ventriculostomy. *European Radiology*, 19(12): 2789–2797, 2009.

[Holland, 1975] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, first edition, 1975. Reprinted in 1992 by MIT Press.

[Householder, 1958] A. S. Householder. Unitary triangularization of a nonsymmetric matrix. *Journal of the ACM*, 5(4): 339–342, 1958.

[Hsu et al., 1999] L. Hsu, M. H. Loew, & J. Ostuni. Automated registration of brain images using edge and surface features. *IEEE Engineering in Medicine and Biology Magazine*, 18(6): 40–47, 1999.

[Jin & Branke, 2005] Y. Jin & J. Branke. Evolutionary optimization in uncertain environments – a survey. *IEEE Transactions on Evolutionary Computation*, 9(3): 303–317, 2005.

[Jin & Sendhoff, 2004] Y. Jin & B. Sendhoff. Constructing dynamic optimization test problems using the multi-objective optimization concept. In *Proceedings of EvoWorkshops, Applications of Evolutionary Computing*, pp. 525–536, Coimbra, Portugal, April 2004. Springer.

[Kapur et al., 1985] J. N. Kapur, P. K. Sahoo, & A. C. K. Wong. A new method for gray-level picture thresholding using the entropy of the histogram. *Computer Vision, Graphics and Image Processing*, 29(3): 273–285, 1985.

[Kennedy & Eberhart, 1995] J. Kennedy & R. C. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks IV*, pp. 1942–1948, Perth, Australia, November 1995.

[Kittler & Illingworth, 1986] J. Kittler & J. Illingworth. Minimum error thresholding. *Pattern Recognition*, 19(1): 41–47, 1986.

[Korosec & Silc, 2009] P. Korosec & J. Silc. The differential ant-stigmergy algorithm applied to dynamic optimization problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 407–414, Trondheim, Norway, May 2009.

[Koza, 1989] J. R. Koza. Hierarchical genetic algorithms operating on populations of computer programs. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, vol. 1, pp. 768–774, Detroit, Michigan, USA, August 1989. Morgan Kaufmann.

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

[Kramer, 1957] C. Y. Kramer. Extension of multiple range tests to group correlated adjusted means. *Biometrics*, 13(1): 13–18, 1957.

[Kruskal & Wallis, 1952] W. Kruskal & W. A. Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260): 583–621, 1952.

[Lepagnot et al., 2009] J. Lepagnot, A. Nakib, H. Oulhadj, & P. Siarry. Performance analysis of MADO dynamic optimization algorithm. In *Proceedings of the IEEE International Conference on Intelligent Systems Design and Applications*, pp. 37–42, Pisa, Italy, December 2009.

[Lepagnot et al., 2010a] J. Lepagnot, A. Nakib, H. Oulhadj, & P. Siarry. Optimisation dynamique :

état de l'art et proposition d'une métaheuristique multi-agents. *Bulletin de la Société Française de Recherche Opérationnelle et d'Aide à la Décision*, (25): 8–12, 2010.

[Lepagnot et al., 2010b] J. Lepagnot, A. Nakib, H. Oulhadj, & P. Siarry. A Multi-Agent based Algorithm for Continuous Dynamic Optimization. In *EU/MEeting 2010 workshop*, Lorient, France, June 2010.

[Lepagnot et al., 2010c] J. Lepagnot, A. Nakib, H. Oulhadj, & P. Siarry. A new multiagent algorithm for dynamic continuous optimization. *International Journal of Applied Metaheuristic Computing*, 1(1): 16–38, 2010.

[Lepagnot et al., 2010d] J. Lepagnot, A. Nakib, H. Oulhadj, & P. Siarry. Réduction du nombre de paramètres de la métaheuristique d'optimisation dynamique MADO. In *Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2010)*, Toulouse, France, Février 2010.

[Lepagnot et al., 2011a] J. Lepagnot, A. Nakib, H. Oulhadj, & P. Siarry. Elastic registration of brain cine-MRI sequences using MLSDO dynamic optimization algorithm. Under submission, 2011.

[Lepagnot et al., 2011b] J. Lepagnot, A. Nakib, H. Oulhadj, & P. Siarry. Brain Cine MRI Segmentation Based on a Multiagent Algorithm for Dynamic Continuous Optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 1, pp. 1695–1702, New Orleans, LA, USA, June 2011.

[Lepagnot et al., 2011c] J. Lepagnot, A. Nakib, H. Oulhadj, & P. Siarry. A Multiple Local Search Algorithm for Continuous Dynamic Optimization. Under submission, 2011.

[Lepagnot et al., 2011d] J. Lepagnot, A. Nakib, H. Oulhadj, & P. Siarry. Brain cine-MRI registration using MLSDO dynamic optimization algorithm. In *Proceedings of the 9th Metaheuristics International Conference (MIC 2011)*, vol. 1, pp. 241–249, Udine, Italy, July 2011.

[Lepagnot et al., 2011e] J. Lepagnot, A. Nakib, H. Oulhadj, & P. Siarry. Recalage de séquences d'images IRM basé sur un nouvel algorithme d'optimisation dynamique. In *Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2011)*, vol. 1, pp. 393–394, Saint-Etienne, France, Mars 2011.

[Li & Yang, 2008] C. Li & S. Yang. A generalized approach to construct benchmark problems for dynamic optimization. In *Proceedings of the 7th International Conference on Simulated Evolution and Learning*, pp. 391–400, Melbourne, Australia, December 2008. Springer.

[Li & Yang, 2009] C. Li & S. Yang. A clustering particle swarm optimizer for dynamic optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 439–446, Trondheim, Norway, May 2009.

[Li et al., 2008] C. Li, S. Yang, T. T. Nguyen, E. L. Yu, X. Yao, Y. Jin, H.-G. Beyer, & P. N. Suganthan. Benchmark generator for CEC 2009 competition on dynamic optimization. Technical Report TR-CEC09-DBG, University of Leicester, University of Birmingham, Nanyang Technological University, 2008.

[Li, 2004] X. Li. Adaptively choosing neighbourhood bests using species in a particle swarm optimizer for multimodal function optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 105–116, Seattle, Washington, USA, June 2004. Springer.

[Li et al., 2006] X. Li, J. Branke, & T. Blackwell. Particle swarm with speciation and adaptation in a dynamic environment. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 51–58, Seattle, Washington, USA, July 2006. ACM.

[Liefooghe et al., 2009] A. Liefooghe, S. Mesmoudi, J. Humeau, L. Jourdan, & E.-G. Talbi. A study on dominance-based local search approaches for multiobjective combinatorial optimization. In *Second International Workshop on Engineering Stochastic Local Search Algorithms : Designing, Implementing and Analyzing Effective Heuristics*, pp. 120–124, Brussels, Belgium, September 2009. Springer.

[Lilliefors, 1967] H. Lilliefors. On the Kolmogorov-Smirnov test for normality with mean and variance unknown. *Journal of the American Statistical Association*, 62(318): 399–402, 1967.

[Lin et al., 2010] X. Lin, T. Qiu, F. Morain-Nicolier, & S. Ruan. A topology preserving non-rigid registration algorithm with integration shape knowledge to segment brain subcortical structures from MRI images. *Pattern Recognition*, 43(7): 2418–2427, 2010.

[Liu et al., 2008] L. Liu, D. Wang, & S. Yang. Compound particle swarm optimization in dynamic environments. In *Proceedings of EvoWorkshops, Applications of Evolutionary Computing*, pp. 617–626, Naples, Italy, March 2008. Springer.

[Liu et al., 2010] L. Liu, S. Yang, & D. Wang. Particle swarm optimization with composite particles in dynamic environments. *IEEE Transactions on Systems, Man, and Cybernetics, Part B : Cybernetics*, 40(6): 1634–1648, 2010.

[Lung & Dumitrescu, 2007] R. I. Lung & D. Dumitrescu. Collaborative evolutionary swarm optimization with a Gauss chaotic sequence generator. *Innovations in Hybrid Intelligent Systems*, 44: 207–214, 2007.

[Lung & Dumitrescu, 2008] R. I. Lung & D. Dumitrescu. ESCA : A new evolutionary-swarm cooperative algorithm. *Studies in Computational Intelligence*, 129: 105–114, 2008.

[Maintz & Viergever, 1998] J. B. A. Maintz & M. A. Viergever. A survey of medical image registration. *Medical Image Analysis*, 2(1): 1–36, 1998.

[Mann & Whitney, 1947] H. B. Mann & D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, 18(1): 50–60, 1947.

[Maurer et al., 1998] C. R. Jr. Maurer, R. J. Maciunas, & J. M. Fitzpatrick. Registration of head CT images to physical space using a weighted combination of points and surfaces. *IEEE Transactions on Medical Imaging*, 17(5): 753–761, 1998.

[Mellor & Brady, 2005] M. Mellor & M. Brady. Phase mutual information as a similarity measure for registration. *Medical Image Analysis*, 9(4): 330–343, 2005.

[Mendes & Mohais, 2005] R. Mendes & A. Mohais. DynDE : A differential evolution for dynamic optimization problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 2808–2815, Edinburgh, Scotland, September 2005.

[Morrison, 2003] R. W. Morrison. Performance measurement in dynamic environments. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 99–102, Chicago, Illinois, USA, July 2003. Springer.

[Morrison & de Jong, 1999] R. W. Morrison & K. A. de Jong. A test problem generator for non-stationary environments. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 2047–2053, Washington, DC, USA, July 1999.

[Moser & Chiong, 2010] I. Moser & R. Chiong. Dynamic function optimisation with hybridised extremal dynamics. *Memetic Computing*, 2(2): 137–148, 2010.

[Moser & Hendtlass, 2007] I. Moser & T. Hendtlass. A simple and efficient multi-component algorithm for solving dynamic function optimisation problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 252–259, Singapore, September 2007.

[Muthukannan & Moses, 2010] K. Muthukannan & M. M. Moses. Color image segmentation using K-means clustering and optimal fuzzy C-means clustering. In *International Conference on Communication and Computational Intelligence*, pp. 229–234, Erode, India, December 2010. IEEE Computer Society.

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

[Nakib et al., 2008] A. Nakib, H. Oulhadj, & P. Siarry. Non-supervised image segmentation based on multiobjective optimization. *Pattern Recognition Letters*, 29(2): 161–172, 2008.

[Nakib et al., 2010] A. Nakib, F. Aiboud, J. Hodel, P. Siarry, & P. Decq. Third brain ventricle deformation analysis using fractional differentiation and evolution strategy in brain cine-MRI. In *Medical Imaging 2010 : Image Processing*, vol. 7623, pp. 76232I–1–76232I–10, San Diego, California, USA, February 2010. SPIE.

[Nelder & Mead, 1965] J.A. Nelder & R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4): 308–313, 1965.

[Nikou et al., 1999] C. Nikou, F. Heitz, & J. P. Armspach. Robust voxel similarity metrics for the registration of dissimilar single and multimodal images. *Pattern Recognition*, 32(8): 1351–1368, 1999.

[Noblet, 2006] V. Noblet. *Recalage non rigide d'images cérébrales 3D avec contrainte de conservation de la topologie.* Thèse de doctorat, Université Louis Pasteur - Strasbourg I, Mars 2006.

[Novoa et al., 2009] P. Novoa, D. A. Pelta, C. Cruz, & I. G. del Amo. Controlling particle trajectories in a multi-swarm approach for dynamic optimization problems. In *Proceedings of the International Work-conference on the Interplay between Natural and Artificial Computation*, pp. 285–294, Santiago de Compostela, Spain, June 2009. Springer.

[Otsu, 1979] N. A. Otsu. A threshold selection method from gray level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1): 62–66, 1979.

[Parrott & Li, 2004] D. Parrott & X. Li. A particle swarm model for tracking multiple peaks in a dynamic environment using speciation. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 98–103, San Diego, California, USA, July 2004.

[Parrott & Li, 2006] D. Parrott & X. Li. Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *IEEE Transactions on Evolutionary*

Computation, 10(4): 440–458, 2006.

[Parzen, 1962] E. Parzen. On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33(3): 1065–1076, 1962.

[Pluim et al., 2000] J. P. W. Pluim, J. B. A. Maintz, & M. A. Viergever. Image registration by maximization of combined mutual information and gradient information. *IEEE Transactions on Medical Imaging*, 19(8): 809–814, 2000.

[Pluim et al., 2003] J. P. W. Pluim, J. B. A. Maintz, & M. A. Viergever. Mutual information based registration of medical images : a survey. *IEEE Transactions on Medical Imaging*, 22(8): 986–1004, 2003.

[Price et al., 2005] K. Price, R. Storn, & J. Lampinen. *Differential Evolution - A Practical Approach to Global Optimization*. Springer, 2005.

[Prins, 2009] C. Prins. Two memetic algorithms for heterogeneous fleet vehicle routing problems. *Engineering Applications of Artificial Intelligence*, 22(6): 916–928, 2009.

[Prins et al., 2010] C. Prins, N. Labadi, C. Prodhon, & R. W. Calvo. Metaheuristics for logistics and vehicle routing. *Computers & Operations Research*, 37(11): 1833–1834, 2010.

[Qiao et al., 2007] Y. Qiao, Q. Hu, G. Qian, S. Luo, & W. L. Nowinski. Thresholding based on variance and intensity contrast. *Pattern Recognition*, 40(2): 596–608, 2007.

[Rechenberg, 1965] I. Rechenberg. Cybernetic solution path of an experimental problem. Technical report, Library translation 1122, Ministry of Aviation, Royal Aircraft Establishment (UK), 1965.

[Richter, 2009] H. Richter. Detecting change in dynamic fitness landscapes. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1613–1620, Trondheim, Norway, May 2009.

[Roche, 2001] A. Roche. *Recalage d'images médicales par inférence statistique*. Thèse de doctorat, Université de Nice Sophia-Antipolis, Février 2001.

[Roche et al., 1998] A. Roche, G. Malandain, X. Pennec, & N. Ayache. The correlation ratio as a new similarity measure for multimodal image registration. In *First International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 1115–1124, Cambridge, MA, USA, October 1998. Springer.

[Rossi et al., 2008] C. Rossi, M. Abderrahim, & J. C. Diaz. Tracking moving optima using Kalman-based predictions. *Evolutionary Computation*, 16(1): 1–30, 2008.

[Sahoo et al., 1988] P. K. Sahoo, S. Soltani, A. K. C. Wong, & Y. Chen. A survey of thresholding techniques. *Computer Vision, Graphics, and Image Processing*, 41(2): 233–260, 1988.

[Santamaría et al., 2011] J. Santamaría, O. Cordón, & S. Damas. A comparative study of state-of-the-art evolutionary image registration methods for 3d modeling. *Computer Vision and Image Understanding*, 115(9): 1340–1354, 2011.

[Sezgin & Sankur, 2004] M. Sezgin & B. Sankur. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, 13(1): 146–165, 2004.

[Shen & Davatzikos, 2002] D. Shen & C. Davatzikos. HAMMER : hierarchical attribute matching mechanism for elastic registration. *IEEE Transactions on Medical Imaging*, 21 (11): 1421–1439, 2002.

[Sorzano et al., 2005] C. O. S. Sorzano, P. Thévenaz, & M. Unser. Elastic registration of biological images using vector-spline regularization. *IEEE Transactions on Biomedical Engineering*, 52(4): 652–663, 2005.

[Srinivasan et al., 2003] B. Srinivasan, S. Palanki, & D. Bonvin. Dynamic optimization of batch processes : I. characterization of the nominal solution. *Computers & Chemical Engineering*, 27(1): 1–26, 2003.

[Storn & Price, 1997] R. Storn & K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4): 431–359, 1997.

[Studholme et al., 1995] C. Studholme, D. Hill, & D. J. Hawkes. Multiresolution voxel similarity measures for MR-PET registration. In *14th International Conference on Information Processing in Medical Imaging*, pp. 287–298, Ile de Berder, France, June 1995. Kluwer Academic Publishers.

[Studholme et al., 1999] C. Studholme, D.L.G. Hill, & D.J. Hawkes. An overlap invariant entropy measure of 3D medical image alignment. *Pattern Recognition*, 32(1): 71–86, 1999.

[Sundar et al., 2009] H. Sundar, H. Litt, & D. Shen. Estimating myocardial motion by 4D image warping. *Pattern Recognition*, 42(11): 2514–2526, 2009.

[Synder & Bilbro, 1990] W. Synder & G. Bilbro. Optimal thresholding : A new approach. *Pattern Recognition Letters*, 11(12): 803–810, 1990.

[Tfaili & Siarry, 2008] W. Tfaili & P. Siarry. A new charged ant colony algorithm for continuous dynamic optimization. *Applied Mathematics and Computation*, 197(2): 604–613, 2008.

[Trojanowski & Wierzchon, 2009] K. Trojanowski & S. T. Wierzchon. Immune-based algorithms for dynamic optimization. *Information Sciences*, 179(10): 1495–1515, 2009.

[Tukey, 1994] J. W. Tukey. The problem of multiple comparisons (unpublished manuscript, 1953). In H. I. Braun, editor, *The Collected Works of John W. Tukey VIII. Multiple Comparisons : 1948-1983*, pp. 1–300. Chapman & Hall, 1994.

[Viola & III, 1995] P. Viola & W. M. Wells III. Alignment by maximization of mutual information. In *Fifth International Conference on Computer Vision*, pp. 16–23, Boston, MA, USA, June 1995. IEEE Computer Society.

[Welch, 1951] B. L. Welch. On the comparison of several mean values : an alternative approach. *Biometrika*, 38: 330–336, 1951.

[Wilcoxon, 1945] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6): 80–83, 1945.

[Woods et al., 1993] R. P. Woods, J. C. Mazziotta, & S. R. Cherry. MRI-PET registration with automated algorithm. *Journal of computer assisted tomography*, 17(4): 536–546, 1993.

[Yang, 2003] S. Yang. Non-stationary problem optimization using the primal-dual genetic algorithm. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 2246–2253, Canberra, Australia, December 2003.

[Yang & Li, 2010] S. Yang & C. Li. A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. *IEEE Transactions on Evolutionary Computation*, 14(6): 959–974, 2010.

[Yang & Yao, 2005] S. Yang & X. Yao. Experimental study on population-based incremental

learning algorithms for dynamic optimization problems. *Soft Computing – A Fusion of Foundations, Methodologies and Applications*, 9(11): 815–834, 2005.

[Yang & Yao, 2008] S. Yang & X. Yao. Population-based incremental learning with associative memory for dynamic environments. *IEEE Transactions on Evolutionary Computation*, 12(5): 542–562, 2008.

[Yu & Suganthan, 2009] E. L. Yu & P. Suganthan. Evolutionary programming with ensemble of external memories for dynamic optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 431–438, Trondheim, Norway, May 2009.

[Zhang, 1993] Z. Zhang. Le problème de la mise en correspondance : l'état de l'art. Technical Report RR-2146, INRIA, 1993.

[Zhang, 1994] Z. Zhang. Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, 13(2): 119–152, 1994.

[Zitová & Flusser, 2003] B. Zitová & J. Flusser. Image registration methods : a survey. *Image and Vision Computing*, 21(11): 977–1000, 2003.