REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de Batna 2

Faculté des Mathématiques et de l'Informatique

Département d'Informatique

THESE

En vue de l'obtention du diplôme de

Doctorat LMD en Informatique

Spécialité : ***Systèmes et Réseaux Informatiques (SRI)***

Présentée par

Ikram BOUBECHAL

Contribution à la Parallèlisation des Algorithmiques Evolutionnaires Appliqués au Traitement de Données Multimédia

Soutenue publiquement le : 30/11/2021

Jury :

Pr. Kamal Eddine MELKEMI	*Professeur*	*Université de Batna 2*	*Président*
Pr. Rachid SEGHIR	*Professeur*	*Université de Batna 2*	*Rapporteur*
Pr. Abdelhamid DJEFFAL	*Professeur*	*Université de Biskra*	*Examinateur*
Dr. Zakaria LABOUDI	*M.C.A*	*Université de OEB*	*Examinateur*
Pr. Ali BEHLOUL	*Professeur*	*Université de Batna 2*	*Examinateur*
Pr. Redha BENZID	*Professeur*	*Université de Batna 2*	*Invité*

Résumé

La compression et la segmentation d'image font partie des techniques de traitement d'image les plus essentielles. Bien que plusieurs méthodes visant à améliorer leur performance aient été proposées dans la littérature, le compromis qualité de solution et temps de calcul reste difficile à satisfaire. Ceci nous a mené à nous intéresser aux algorithmes méta-heuristiques et leur application au traitement d'image, particulièrement ceux à base de population. En premier lieu, nous avons testé l'efficacité de trois algorithmes méta-heuristiques à savoir *Whale Optimization Algorithm* (WOA), *Particle Swarm Optimization* (PSO) et *Firefly Algorithm* (FA) dans la résolution d'un problème très répandu dans la compression par quantification vectorielle et qui consiste en la production d'un dictionnaire optimal. Les résultats obtenus indiquent que l'utilisation des algorithmes méta-heuristiques n'apportent aucune amélioration et que l'utilisation de l'algorithme *Lind-Buzo-Grey* (LBG) classique reste, à notre avis, le choix le plus performant. Quant à notre deuxième contribution, elle consiste à utiliser la métrique *SSIM* comme fonction objective optimisée par *PSO*, *FA* et *Bat Algorithm* (BA) avec des paramètres préalablement ajustés, afin de résoudre le problème de segmentation, particulièrement, le multi-seuillage. Les résultats obtenus montrent que la méthode *SSIM* basée sur *PSO* produit des solutions quasi-identiques à identiques aux solutions exactes (obtenues par la recherche exhaustive) et permet de générer des images segmentées de meilleures qualités comparés à l'utilisation de la méthode classiques : *Otsu*. Néanmoins, la fonction *SSIM* s'est avérée un peu couteuse en temps de calcul. De ce fait, la complexité de calcul a été considérablement réduite en adoptant un paradigme de programmation parallèle à mémoire partagée (*OpenMp*). En dernier lieu, nous avons apporté des modifications à l'algorithme *Bacterial Foraging Optimization* (BFO) dans le but d'améliorer sa convergence et la qualité des solutions obtenues. Notre approche ***Upgraded-BFO*** (UBFO) est comparée aux algorithmes : *PSO*, *WOA*, *Grey Wolf Optimization* (GWO), *Moth Flame Optimization* (MFO), *BFO* et *Modified BFO* (MBFO). Les expérimentations indiquent que ***UBFO*** produit de meilleurs résultats par rapport aux algorithmes *BFO*, *MBFO*, *MFO* et *GWO* et qu'il très compétitif avec *PSO* et *WOA*. De plus, les tests statistiques ont montré que ***UBFO*** est l'algorithme le plus précis et le plus stable par rapport aux autres algorithmes utilisés.

Mots clés: *Optimisation, Méta-heuristiques, Segmentation, Compression, Multi-seuillage, Quantification Vectorielle, Programmation Parallèle à mémoire partagée.*

Abstract

Image compression and segmentation are among the most essential image processing techniques. several approaches have been proposed in the literature in order to improve their performance but the trad-off between solution quality and computation time is still hard to reach. This led us to focus on meta-heuristic algorithms and their application to image processing, particularly meta-heuristic based on population. First, we tested the efficiency of three meta-heuristic algorithms, namely *Whale Optimization Algorithm* (WOA), *Particle Swarm Optimization* (PSO) and *Firefly Algorithm* (FA) to solve vector quantization problem, which consists in codebook design. The obtained results indicate that the use of the meta-heuristic algorithms does not overcome this weakness. The classical *Lind-Buzo-Grey* algorithm remains, from our point of view, the most efficient choice. The second contribution of this thesis consists in using the *SSIM* metric as an objective function optimized by *PSO*, *Bat Algorithm* (BA) and *FA* with previously adjusted parameters, in order to solve the segmentation problem, particularly, the multi-thresholding. The obtained results show that the *SSIM* method based on *PSO* produces the most near solutions to the exact one (obtained by the exhaustive search) and allows to generate better quality segmented images compared to the use of the classical method: *Otsu*. Nevertheless, the *SSIM* function requires more execution time. Therefore, the computation complexity has been considerably reduced by adopting a shared memory parallel programming paradigm (*OpenMp*). Finally, we have made modifications to the *Bacterial Foraging Optimization* (BFO) algorithm in order to improve its convergence and the obtained solutions quality. The ***Upgraded-BFO*** (UBFO) approach is compared to the following algorithms: *PSO*, *WOA*, *Grey Wolf Optimization* (GWO), *Moth Flame Optimization* (MFO), *BFO* and *Modified BFO* (MBFO). The experiments indicate that ***UBFO*** produces better results than *BFO*, *MBFO* and *MFO* algorithms and that it is very competitive with *PSO* and *WOA*. Moreover, statistical tests have shown that ***UBFO*** is the most accurate and stable algorithm compared to the other algorithms.

Key words: *Optimization, Meta-heuristics, Segmentation, Multi-thresholding, Compression, Vector Quantization, Shared-memory parallel programming*

ملخص

يعد ضغط الصور وتجزئتها من بين أهم تقنيات معالجة الصور. على الرغم من أن العديد من الأساليب التي تهدف إلى تحسين أدائها قد تم اقتراحها في الأدبيات، إلا أن الموازنة بين جودة الحل ووقت الحساب لا تزال صعبة التحقيق. قادنا هذا إلى التركيز على خوارزميات العشوائية وتطبيقها في معالجة الصور، لا سيما تلك القائمة على السكان. أولاً، اختبرنا فعالية ثلاث خوارزميات العشوائية معروفة بفعاليتها الا وهي : WOA، PSO و Firefly في حل مشكلة شائعة جدًا في الضغط عن طريق تكميم المتجهات والمتمثلة في إنتاج `` قاموس مثالي ''. تشير النتائج التي تم الحصول عليها إلى أن استخدام الخوارزميات الوصفية لا يجلب أي تحسين وأن استخدام خوارزمية LBG الكلاسيكية يظل، في رأينا، الخيار الأكثر كفاءة. أما بالنسبة لمساهمتنا الثانية، فهي تتمثل في استخدام مقياس SSIM كدالة موضوعية محسّنة بواسطة PSO، Bat و Firefly مع معلمات معدلة مسبقًا، من أجل حل مشكلة التجزئة، وبتحديد العتبات المتعددة. تظهر النتائج التي تم الحصول عليها أن طريقة SSIM القائمة على PSO تنتج حلولًا متطابقة تقريبًا مع الحلول الدقيقة (تم الحصول عليها من خلال بحث شامل) وتسمح بتوليد صور مجزأة ذات جودة أفضل مقارنة باستخدام الطرق الكلاسيكية: Otsu و Kapur. ومع ذلك، تبين أن وظيفة SSIM مكلفة بعض الشيء من حيث وقت الحوسبة. نتيجة لذلك، تم تقليل التعقيد الحسابي بشكل كبير من خلال اعتماد نموذج برمجة ذاكرة مشتركة متوازية (OpenMp). أخيرًا، أجرينا تغييرات على خوارزمية BFO من أجل تحسين طريقة بحثها وجودة الحلول التي يتم الحصول عليها. اجريت مقارنة نهج UBFO (Upgraded-BFO) بالخوارزميات: PSO، WOA، GWO، MFO MBFO و BFO . تشير التجارب إلى أن UBFO ينتج نتائج أفضل مقارنة بالخوارزميات:BFO MBFO وMFO وأنه منافس للغاية مع PSO و WOA. بالإضافة إلى ذلك، أظهرت الاختبارات الإحصائية أن UBFO هي الخوارزمية الأكثر دقة واستقرارًا مقارنة بالخوارزميات الأخرى المستخدمة.

كلمات البحث: *التحسين ، الاستدلال الفوقي ، تجزئة الصور ، ضغط الصور ، العتبات المتعددة ، القياس الكمي للمتجهات ، الموازنة.*

Remerciements

Ma profonde gratitude est envers mon ***Créateur le Tout Puissant Allah*** *qui m'a accordé patience et persévérance afin de faire aboutir ce travail.*

Je rends un grand hommage à mon directeur de thèse, ***Mr Rachid Seghir****, Professeur à l'Université Mostefa Benboulaid, Batna 2, qui a dirigé de près ce travail malgré ses nombreuses occupations. Il m'a fait bénéficier de ses compétences, de ses conseils constructifs, de ses encouragements et de ses grandes qualités humaines. Je voudrais lui témoigner ma très profonde reconnaissance et ma très haute considération. Je tiens aussi à exprimer mes sincères remerciements et toute ma gratitude à* ***Mr Redha Benzid****, Professeur à l'Universitaire Mostefa Benboulaid, Batna 2, pour l'aide précieuse qu'il m'a prodigué tout au long de cette thèse.*

Je suis très honorée que ***Mr Kamal Eddine Melkemi****, Professeur à l'université Mostefa Benboulaid, Batna2, ait accepté de présider le jury. Qu'il soit assuré de mon profond respect et de mes sincères remerciements.*

Je remercie vivement ***Mr Ali Behloul****, Professeur à l'Université Mostefa Benboulaid, Batna 2,* ***Mr Djeffal Abdelhamid****, Professeur à l'Université Mohamed Khider, Biskra, et* ***Mr Laboudi Zakaria****, Maître de conférences A à l'Université Larbi Ben M'hidi, OEB, d'avoir consenti à évaluer ce travail et pour l'honneur qu'ils me font en acceptant de faire partie du jury.*

Je souhaiterais aussi adresser mes remerciements à ***Mme Sonia Sabrina Bendib*** *pour ses précieux conseils et encouragements continuels qui me remotiver à chaque fois que j'en avais besoin.*

Je remercie particulièrement ***Mme Souheila Bouam*** *pour sa présence, sa patience et son soutien qui ne sauraient venir que d'une sœur. Il m'était parfois difficile de persévérer mais ses encouragements incessants m'ont donné la force de continuer, je lui suis très reconnaissante.*

Enfin, je ne saurais oublier de remercier et d'exprimer ma gratitude à mes parents, mon frère, mes sœurs et mes beaux-frères pour le soutien continuel qu'ils m'apportent, ainsi qu'a Nesrine, Nihed, Abla, Fadhila, Ghozlane et toutes mes amies et collègues.

A mes très chers parents, sans qui cette thése n'aurait pu voir le jour.
A moi-même.
Et à tous ceux qui me sont chers.

Table des matières

Tables des figures

Liste des tableaux

Liste des algorithmes

Liste des abréviations

ABC	**A**rtificial **B**ee **C**olony
ACO	**A**nt **C**olony **O**ptimization
BA	**B**at **A**lgorithm
BFO	**B**acterial **F**oraging **O**ptimization
CCD	**C**harge **C**oupled **D**evice
CMOS	**C**omplementary **M**etal **O**xide **S**emiconductor
CS	**C**uckoo **S**earch
DE	**D**ifferential **E**volution
FA	**F**irefly **A**lgorithm
GA	**G**enetic **A**lgorithm
GWO	**G**rey **W**olf **O**ptimization
HBMO	**H**oney **B**ees **M**ating **O**ptimization
HCOCLPSO	**H**ybrid **C**ooperative **C**omprehensive **L**earning based **PSO**
LBG	**L**ind-**B**uzo-**G**ray
MBFO	**M**odified **B**acterial **F**oraging **O**ptimization
MEABCT	**M**aximum **E**ntropy **B**ased **A**rtificial **B**ee **C**olony **T**hresholding
MFO	**M**oth **F**lame **O**ptimization
NFL	**N**o **F**ree **L**unch
OpenMP	**O**pen **M**ulti−**P**rocessing
PSNR	**P**eak **S**ignal to **N**oise **R**atio
PSO	**P**artical **S**warm **O**ptimization
QV	**Q**uantification **V**ectorielle
SA	**S**imulated **A**nnealing
SI	**S**warm **I**ntelligence
SSIM	**S**tructural **SIM**ilarity
StD	**S**tandard **D**eviation
UBFO	**U**pgraded **B**acterial **F**oraging **O**ptimization
WOA	**W**hale **O**ptimization **A**lgorithm

Chapitre 1

Introduction générale

La théorie de l'optimisation joue un rôle très important dans plusieurs domaines notamment, l'ingénierie, les mathématiques et l'économie, elle vise à déterminer l'optimum global d'un problème donné. De ce fait, plusieurs méthodes d'optimisation ont été proposées, et qui peuvent être classées en deux catégories : les méthodes exactes et les méthodes approchées (stochastiques).

Les méthodes exactes, tels que la programmation dynamique, la recherche A* où Branch & Bound garantissent l'optimalité des problèmes de petites tailles. Néanmoins, elles deviennent très couteuses en termes de temps lorsqu'il s'agit des problèmes NP-Complet ou NP-difficile (problèmes qui ne peuvent être résolus en un temps polynomial (Friedrich et al., 2010; Farshi, Drake, and Özcan, 2020); d'où l'intérêt de l'utilisation des méthodes (méta)heuristiques. Alors que les heuristiques dépendent des caractéristiques du problème à traiter et sont, de ce fait, conçues pour résoudre le problème en question, les méta-heuristiques sont, quant à elles, de nature plus générale et s'appliquent à plusieurs catégories de problèmes.

Les algorithmes méta-heuristiques sont des méthodes itératives, généralement inspirées du comportement des animaux, de la biologie ou de la physique. Ces dernières décennies, les méta-heuristiques ont montré leur efficacité dans la résolution des problèmes d'optimisation complexes grâce à leur capacité de progresser vers l'optimum global tout en essayant d'éviter d'être piégés dans les optimums locaux. Cependant, le théorème *No Free Lunch* NFL (Wolpert, Macready, et al., 1995) montre qu'il n'y a pas un algorithme méta-heuristique qualifié pour résoudre tous les problèmes d'optimisation. En effet, un algorithme méta-heuristique donné peut être performant dans la résolution d'un ensemble de problèmes d'optimisation donné et non efficace dans la résolution d'un autre type de problème. De ce fait, le développement de nouveaux

algorithmes méta-heuristiques ou l'amélioration de ceux déjà développés devient un enjeu de recherche très actif et un défi majeur à relever.

Par ailleurs, le traitement d'images occupe une place importante dans divers domaines, tels que la vision par ordinateur, la télédétection, la reconnaissance de formes, l'imagerie médicale, la modélisation de l'environnement, etc. Dans ce contexte, la compression et la segmentation d'image font partie des techniques de traitement les plus essentielles.

La compression consiste à réduire la taille physique de l'image en éliminant les données redondantes pour un gain de stockage et de transmission (Salomon and Motta, 2010). La deuxième technique partitionne une image en régions homogènes en fonction de caractéristiques particulières, telles que : la couleur, la structure de la texture et l'intensité (Fu and Mui, 1981a; Gonzalez and Woods, 2006). Le but d'une telle division est d'extraire les parties significatives qui sont faciles à analyser et à interpréter. Plusieurs méthodes de compression ont été proposées et classées en deux catégories, la compression sans perte et la compression avec perte. La première catégorie représente une compression non destructive qui permet de réduire la taille des données sans aucune perte d'information. La deuxième catégorie code l'image avec une dégradation de qualité, car cette technique entraîne une certaine perte de données permettant d'atteindre un taux de compression plus élevé que celui sans perte. La quantification vectorielle (QV) est l'une des techniques les plus efficaces utilisées dans la compression d'image avec perte, en raison de son architecture simple et de son taux de compression élevé avec une distorsion minimale. QV permet de générer un dictionnaire de codes optimal de longueur limitée qui représente l'image originale avec une fidélité plus élevée basée sur le codage par blocs. De plus, plusieurs algorithmes ont été proposés dans QV, la méthode bien connue est le *Linde-Buze-Gray* qui est un algorithme d'apprentissage itératif basé sur un ensemble d'apprentissage (Linde, Buzo, and Gray, 1980). Cependant, l'inconvénient de l'algorithme LBG est la génération d'un dictionnaire local (Gersho and Gray, 2012). Afin de remédier à ce problème, trois algorithmes méta-heuristiques ont été utilisés notamment *WOA*, *PSO* et *FA*.

Quant au second traitement, qui est la segmentation, nous avons choisi d'étudier le seuillage car il joue un rôle important dans la segmentation des images (Haralick and Shapiro, 1985). Il est utile pour l'extraction des objets de l'arrière-plan ou pour distinguer les différents objets présents dans l'image. En général, si l'histogramme de

l'image est bimodal, le choix de la valeur du seuil est facile. En effet, il suffit juste de prendre la valeur qui se trouve dans la vallée entre deux pics. Cependant, les histogrammes des images du monde réel sont multimodaux, ce qui rend le choix des valeurs des seuils exactes très difficiles. Ainsi, le multi-seuillage est considéré comme un domaine de recherche très actif. Depuis, plusieurs techniques de multi-seuillage ont été proposées dans la littérature, telles que la méthode Otsu et Kapur.

Dans ce contexte, et afin améliorer la qualité des images segmentées, nous avons utilisé la métrique de qualité *SSIM* comme fonction objective. Cependant, le temps de la recherche exhaustive des seuils augmente exponentiellement avec le nombre de seuils. Comme solution à ce problème, nous avons utilisé trois algorithmes méta-heuristiques bien connus et largement utilisés à savoir *PSO*, *FA* et *BA* et nous avons ajusté leurs paramètres de manière empirique afin de produire des solutions quasi exactes. De plus, nous avons montré que la complexité de calcul a été considérablement réduite en adoptant un paradigme de programmation parallèle à mémoire partagée pour tous les algorithmes que nous avons implémentés.

De plus, étant donné que notre travail est axé sur les algorithmes méta-heuristiques, nous avons proposé une modification de l'algorithme *BFO* afin d'améliorer ses performances. L'approche proposée, nommée *Upgraded-BFO* est testée sur 21 fonctions benchmarks et comparée avec cinq autres algorithmes méta-heuristiques, notamment, *PSO*, *WOA*, *MFO*, *GWO*, *BFO* et *MBFO*. Puis, nous avons appliqué ***UBFO*** dans la résolution du problème du multi-thresholding pour valider notre approche dans la résolution des problèmes du monde réel.

La thèse est organisée en adéquation à notre démarche, elle est constituée des parties suivantes :

Chapitre 1: Consacré à l'introduction générale présentée ci-dessus.

Chapitre 2: Ce chapitre décrit le principe de l'optimisation mono-objective, et ses différentes techniques appliquées dans la résolution des problèmes complexes. Particulièrement les méthodes méta-heuristiques, ces dernières sont décrites à travers leurs caractéristiques ainsi que leur fonctionnement. Nous abordons aussi l'application du parallélisme aux méthodes méta-heuristiques.

Chapitre 3: Dans ce chapitre, nous nous focalisons sur le seuillage des images en niveau de gris et l'utilisation des méta-heuristiques pour la résolution du problème de segmentation.

Chapitre 4: Ce chapitre décrit le principe de la quantification vectorielle et notre approche proposée pour la résolution de son problème majeur, celui de la conception du dictionnaire représentant l'image à coder. Nous avons proposé d'utiliser une méthode méta-heuristique récente pour produire un dictionnaire optimal, ainsi nous avons comparé les résultats obtenus avec ceux obtenus par d'autres algorithmes très utilisés dans la littérature.

Chapitre 5: Ce chapitre présente notre deuxième contribution, qui consiste en la proposition d'une nouvelle fonction objective pour la résolution du problème de multi-seuillage (*mutithresholding*), l'utilisation des méta-heuristiques ainsi que la parallélisation de ces dérinieres. Les expérimentations effectuées en vue de valider l'approche proposée y sont également présentées.

Chapitre 6: Notre travail étant axé sur l'optimisation et plus particulièrement les algorithmes méta-heuristiques, une approche concernant l'amélioration de l'algorithme *BFO* est présentée dans ce chapitre. Les résultats de comparaison de notre approche *UBFO* avec cinq autres algorithmes différents et de son application dans la résolution du problème de la segmentation y sont également présentés.

Chapitre 7: Ce chapitre est dédié à la *Conclusion générale et aux perspectives*. Dans cette partie, nous présentons une récapitulation de la démarche suivie ainsi que notre contribution avec ses points forts et ses points faibles. Nous finirons, enfin, par quelques perspectives de notre travail.

Chapitre 2

Optimisation et Méta-heuristiques

2.1 Introduction

L'optimisation est le processus qui consiste à trouver la ou les meilleures solutions possibles pour un problème donné. Elle permet de minimiser ou de maximiser une fonction en choisissant systématiquement les valeurs des variables dans un ensemble admissible. Cependant, les méta-heuristiques sont utilisées d'avantage que les méthodes exactes dans la résolution des problèmes d'optimisation complexes en raison de leur capacité à explorer et à trouver des régions prometteuses dans l'espace de recherche en un temps de calcul abordable.

2.2 L'Optimisation

L'optimisation vise à trouver la meilleure solution possible ou à exploiter un système de la manière la plus efficace possible. Elle peut être présentée comme un processus d'ajustement des données en entrées (variables de décisions) dans le but d'atteindre la solution optimale (minimum, maximum) du problème en question (Haupt and Haupt, 2004; Wang et al., 2009).

Un problème d'optimisation peut être décrit mathématiquement comme suit (Oliva and Cuevas, 2017) :
Étant donné une fonction $f : S \rightarrow \mathbb{R}$, nommée fonction objective (fitness ou coût),on cherche à trouver les paramétres qui minimisent f (cas de minimisation) (Oliva, Abd Elaziz, and Hinojosa, 2019)

$$x^* = \arg\min_{x \in S} f(x) \tag{2.1}$$

S définit l'espace de recheche qui représente toutes les solutions possibles pour le problème d'optimisation, les variables x sont souvent appelées solutions candidates.

La fonction f décrit le critère d'optimisation, elle permet de calculer une valeur qui indique la *qualité* d'une solution x donnée et x^* représente la meilleure solution trouvée, tel que : $f(s^*) \leq f(s), \forall s \in S$.

2.2.1 Classification des problèmes d'optimisation

Les problèmes d'optimisation varient en fonction de leur structure (mono ou multi-objective, avec ou sans contraintes) (Hussain et al., 2019) et selon le type des variables utilisées (continues ou discrètes).

Optimisation Combinatoire vs. Optimisation Continue : Tout problème comportant un grand nombre de solutions discrètes (un ensemble fini de solutions) et une fonction objective pour évaluer ces solutions les unes par rapport aux autres est un problème d'optimisation combinatoire (discrète) (Blum and Roli, 2003; Du, Swamy, et al., 2016). Tandis que l'optimisation continue traite des problèmes définis sur des domaines numériques ayant des variables de décision à valeur réelle (Nocedal and Wright, 2006).

Optimisation mono-objective /Optimisation multi-Objective : Les problèmes d' optimisation mono-objectif visent à trouver une solution unique, celle ayant le coût optimal en maximisant (minimisant) un objectif particulier. Ces problèmes peuvent aussi avoir plusieurs objectifs combinés pour former un seul objectif principal. D'autre part, l'optimisation multi-objectifs, également appelée optimisation multicritères consiste à satisfaire plusieurs critères ou objectifs contradictoires simultanément. (Yang, 2010b; Hussain et al., 2019).

Optimisation sous contrainte vs. optimisation sans contrainte: Les problèmes d'optimisation sans contrainte n'impliquent aucune restriction ou limitation, et dépendent de variables réelles. En revanche, la majorité des problèmes d'optimisation du monde réel comportent certaines contraintes/restrictions qui doivent être satisfaites afin d'aboutir à une solution optimale. Ces contraintes décrivent les dépendances entre les variables de décision (solution candidates) et les paramètres du problème (Bae et al., 2012; Hussain et al., 2019).

Une fois qu'un problème d'optimisation est correctement formulé , la tâche principale consiste à déterminer les solutions optimales par une méthode de résolution utilisant les bonnes règles mathématiques. De ce fait, plusieurs méthodes d'optimisation

ont été proposées, et qui peuvent être classées en deux catégories : les méthodes exactes et les méthodes approchées.

2.3 Méta-heuristiques pour l'optimisaiton mono-objectif

Les méta-heuristiques ont été développées à l'origine pour pallier aux limites des méthodes exactes et heuristiques. Elles font références aux méthodes de recherche approximatives utilisées dans la résolution de problèmes d'optimisation combinatoire (problèmes d'optimisation dont les variables ont une structure discrète) (Homayouni and Fontes, 2018; Radosavljević, 2018). Le terme méta-heuristique » a été inventé pour la première fois par Glover en 1986, faisant référence à la recherche Tabou (Glover, 1986).

Les méta-heuristiques construisent et/ou améliorent itérativement les solutions en appliquant certaines règles stochastiques prédéfinies sur les variables initiales (Talbi, 2009; Li et al., 2014). Ces techniques peuvent être appliquées à une variété de problèmes d'optimisation, d'où le qualificatif « méta ».

La majorité des méta-heuristiques sont inspirées de la nature (basées sur certains principes de la physique, de la biologie ou de l'éthologie), et se basent sur des composantes stochastiques (impliquant des variables aléatoires) (Mirjalili, Mirjalili, and Lewis, 2014; Hussain et al., 2019). Néanmoins , La performance de tout algorithme méta-heuristique est liée à sa capacité de maintenir un bon équilibre entre l'exploration (diversification) et l'exploitation (intensification). Ces deux dernières sont une partie fondamentale de l'algorithme, elles favorisent l'obtention d'une estimation précise de l'optimum global pour n'importe quel problème d'optimisation (Hussain et al., 2019; Birattari et al., 2001).

- *L'exploration* désigne la capacité de parcourir l'espace de recherche de manière aléatoire et aussi large que possible afin d'identifier les zones prometteuses .
- *L'exploitation* fait référence à l'intensification de la recherche dans les zones prometteuses détectées lors de l'exploration.

Les méthodes méta-heuristiques peuvent être classées selon le nombre de solutions utilisées durant la recherche en deux catégories : les méta-heuristiques à base de solution unique (trajectoire) et les méta-heuristiques basées sur une population de solution. Ce critère de classification est souvent considéré comme une distinction fondamentale dans la littérature (Mirjalili, Mirjalili, and Lewis, 2014; Birattari et al., 2001). Le processus de recherche de la première catégorie commence avec une solution candidate. Cette solution candidate unique est ensuite améliorée au fil des itérations. Tandis que les méta-heuristiques basées sur la population effectuent l'optimisation en utilisant un ensemble de solutions (population). Dans ce cas, le processus de recherche commence avec une population initiale aléatoire (plusieurs solutions) et cette population est améliorée au fil des itérations (Homayouni and Fontes, 2018; Hussain et al., 2019; Boussaïd, Lepagnot, and Siarry, 2013). Une classification détaillée des méta-heuristiques est donnée par (Birattari et al., 2001; Talbi, 2009).

Le domaine des méta-heuristiques ne cesse d'évoluer, chaque année de nouvelles méthodes sont développées et appliquées avec succès dans la résolution de divers problèmes d'optimisation. Dans ce qui suit nous allons décrire les méthodes les plus utilisées dans la littérature en se focalisant sur celles utilisées dans notre travail de recherche.

2.3.1 Algorithme de recuit simulé

Le recuit simulé (Simulated annealing) est une technique d'optimisaiton probabiliste basée sur une analogie avec le recuit des métaux. Cependant, l'algotithme de Metropolis (Metropolis et al., 1953; Hastings, 1970) est le point principal sur lequel repose la méthode du recuit simulé. En effet, quand la température est élevée, cet algorithme va permettre d'avoir comme résultats les minima locaux. Dans le cas échéant, l'algotithme de Metropolis va stocker les états les plus probables. A noter que, le recuit est un processus utilisé afin d'atteindre les états de basse énergie d'un solide en chauffant ce dernier à des températures très élevées puis en le laissant refroidir. Cette méthode provient de la métallurgie. Ainsi, la loi de décroissance de la température est le point principal de l'algorithme de Metropolis.

Le recuit simulé a été présenté la première fois par (Kirkpatrick, Gelatt, and Vecchi, 1983), son algorithme de base est illustré dans la figure 2.1.

l'Algorithme 1 et l'Algorithme 2 décrivent l'algorithme de Metropolis et l'algorithme de recuit simulé, respectivement.

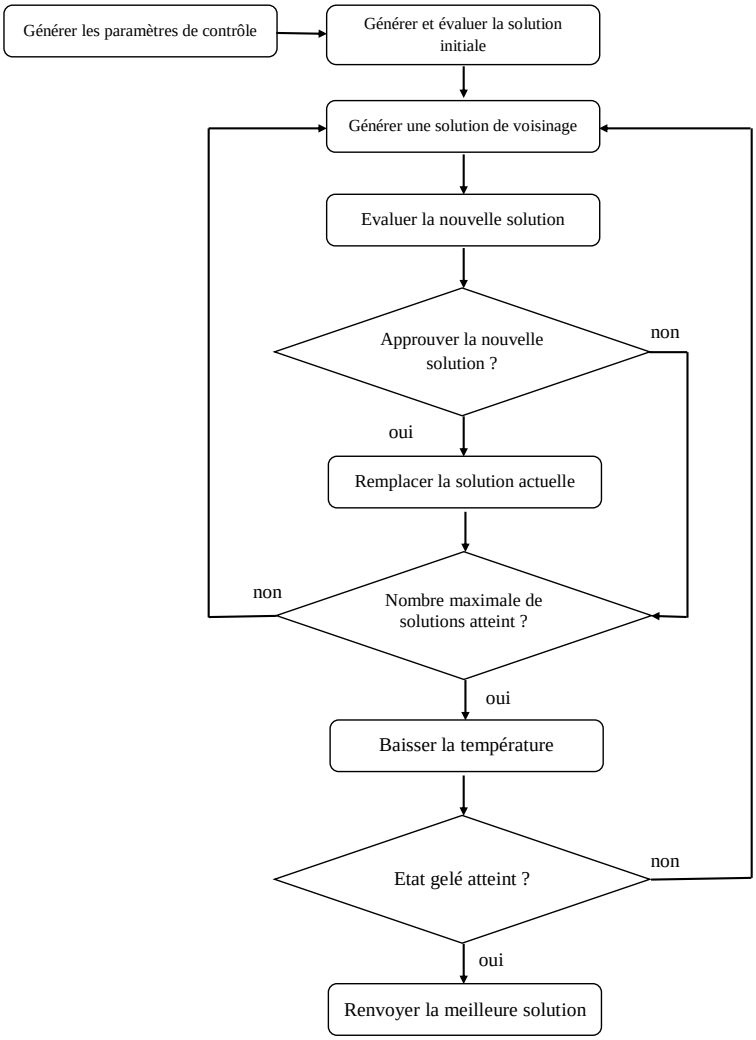


FIGURE 2.1: L'algorithme de base du récuit simulé RS (Homayouni and Fontes, 2018)

Algorithm 1 Algorithme de Metropolis

1: *Initialiser un point de départ* x_0 *et une température* T
2: **Pour** ($i = 0$ à n) **faire**
3: **Tant que** (x_i *n'est pas accepté*) **faire**
4: **si** $f(x_i) \leq f(x_{i-1})$ *: accepter* x_i
5: **si** $f(x_i) > f(x_{i-1})$ *: accepter* x_i *avec la probabilité* $e^{-\frac{f(x_i)-f(z_{i-1})}{T}}$
6: **Fin Tant que**
7: **Fin Pour**

Algorithm 2 Algorithme de recuit simulé

Déterminer une configuration aléatoire S
Choix des mécanismes de perturbation d'une configuration
Initialiser la température T
Tant que *la condition d'arrêt n'est pas atteinte* **faire**
  Tant que *l'équilibre n'est pas atteint* **faire**
    Tirer une nouvelle configuration S'
    Appliquer la règle de Metropolis
    si $f(S') < f(S)$
      $S_{min} = S'$
      $f_{min} = f(S')$
    Fin si
  Fin Tant que
  Décroître la température
Fin Tant que

2.3.2 L'algorithme de recherche Tabou

La recherche Tabou est une méta-heuristique d'optimisation globale basée sur une solution unique, developpé par (Glover, 1986). C'est une méthode d'escalade qui imite la structure de la mémoire humaine pour améliorer la prise de décision (Glover, 1990).

Le point crucial de l'algorithme recherche Tabou est sa mémoire à court terme qui va lui permettre d'éviter le retour à une configuration déjà passée. Cette configuration est maintenu dans une liste appelée ***liste tabou***, permettant ainsi d'ajouter une forme d'intelligence aux méthodes de recherche locale (Gendreau and Potvin, 2005; Glover and Laguna, 1998). La figure 2.2 illustre l'algorithme de base de la recherche Tabou (Homayouni and Fontes, 2018), qui est décrit par l'algorithme 3.

Algorithm 3 Algorithme de recherche Tabou

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

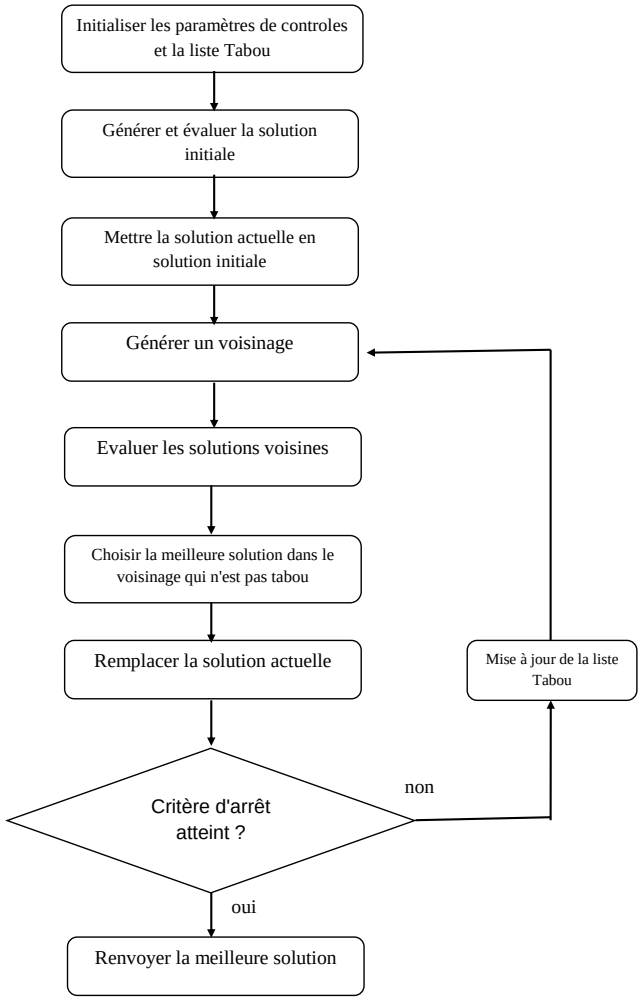


FIGURE 2.2: L'algorithme de base de la recherche Tabou (Homayouni and Fontes, 2018)

2.3.3 Optimisation par Essaim de Particules (Partical Swarm Optimization)

L'optimisation par essaim de particules est une méthode méta-heuristique basée sur l'intelligence collective; utilisée dans la résolution des problèmes d'optimisation combinatoire. Cette méthode a été initialement développée par (Kennedy and Eberhart, 1995). Elle s'inspire du comportement social des volées d'oiseaux et des bancs de poissons lorsqu'ils sont à la recherche de nourriture.

PSO (Shi and Eberhart, 1999; Zhou and Shi, 2011) est un algorithme basé sur une population de solution, il utilise un essaim de particules de sorte que chaque particule représente une solution potentielle au problème d'optimisation. Cependant, la position de chaque particule x_i est influencée par : sa propre vitesse v_i, la meilleure position visitée par elle-même *(pbest)* et par la position de la meilleure particule dans l'ensemble de la population *(gbest)*. L'algorithme 4 résume son principe de fonctionnement.

Les vitesses et les positions des particules sont mises à jour à chaque itération selon les équations suivantes (Kennedy and Eberhart, 1995) :

$$v_{id}^{t+1} = v_{id}^{t} + C_1 r_1^t \left(pbest_{id}^t - x_{id}^t\right) + C_2 r_2^t \left(gbest_d^t - x_{id}^t\right) \tag{2.2}$$

$$x_{id}^{t+1} = v_{id}^{t+1} + x_{id}^t \tag{2.3}$$

où d représente la dimension du problème; $\boldsymbol{x}_{id}^l$ est la position de la i^{me} particule de la population à l'itération t et $\boldsymbol{v}_{id}^l$ sa vitesse ; C_1 et C_2 sont les coefficients cognitifs et sociaux, respectivement ; r_1 et r_2 sont deux nombres aléatoires appartenant à l'intervalle $[0, 1]$.

Cependant, le PSO de base caractérisée par (2.2) ne possède aucune stratégie pour ajuster le compromis entre les capacités d'exploration et d'exploitation de PSO. Par conséquent, le poids d'inertie PSO est introduit pour remédier à cet inconvénient par (Shi and Eberhart, 1999). Dans le PSO à poids d'inertie, qui est la variante de PSO la plus couramment utilisée, les vitesses des particules sont multipliées par un paramètre appelé poids d'inertie w. L'équations de mise à jour de vitesse correspondante est la suivante :

$$v_{id}^{t+1} = \omega^t v_{id}^{t} + C_1 r_1^t \left(pbest_{id}^t - x_{id}^t\right) + C_2 r_2^t \left(gbest_d^t - x_{id}^t\right) \tag{2.4}$$

Le poids d'inertie donné par l'équation (2.5) ajuste le compromis entre l'exploration et l'exploitation de l'algorithme PSO, plus la valeur du poids d'inertie est importante, plus la capacité d'exploration de PSO sera importante et vice versa. Généralement, il est diminué linéairement au cours des itérations, de sorte que l'effort de recherche se concentre principalement sur l'exploration au début des itérations et se concentre davantage sur l'exploitation aux dernieres itérations (Shi and Eberhart, 1999).

$$\omega^t = \omega_{min} - \left[(\omega_{min} - \omega_{max}) \times \frac{t}{max - iteration} \right], \quad t = 1, 2, \ldots, max - iteration \tag{2.5}$$

Généralement, ω_{min} et ω_{max} sont initialisés à 0.4 et 0.9, respectivement.

En 2002 Kennedy et Clerc (Clerc and Kennedy, 2002) ont proposé d'utiliser un coefficient de construction au lieu du poids d'inertie, mais la version PSO basée sur ce dernier reste la plus utilisée.

Algorithm 4 Algorithme PSO

Initialiser aléatoirement les positions x_i et leurs vitesses correspondantes v_i
faire
  pour *(chaque particule i)* **faire**
    Calculer la valeur fitness f_i de chaque positio x_i
    si (f_i<f_{pbesti}) **alors**
      $pbest_i = x_i$
    fin
    si (f_i<f_{gbest}) **alors**
      $gbest = x_i$
    fin
    Mise à jour de x_i et de v_i par l'Eq (2.3) et l'Eq (2.5), respectivement.
  fin
tant que ($t <$ *le nombre max d'itération*)
renvoie gbest

2.3.4 Bacterial Foraging Optimization "BFO"

L'algorithme BFO (Passino, 2002), introduit en 2002 par Passino, est un algorithme d'intelligence en essaim basé sur le mécanisme individuel et collectif des bactéries Echerichia coli (E.Coli). En effet, En tant qu'organismes unicellulaires procaryotes, E.Coli présentent d'excellentes capacités d'organisation et de construction grâce à leur comportements de colonie (Panda and Naik, 2012), particuliérement leur comportement de recherche alimentaire, et précisément le processus par lequel E.Coli trouvent des nutriments, évitent les substances nocives, tout en se déplaçant simultanément vers d'autres cellules sans être extrêmement proches d'elles. Cependant, la stratégie de recherche de nourriture des bactéries E. coli est dirigée par quatre processus, à savoir la chimiotaxie, la reproduction, l'élimination et la dispersion, et l'essaimage. Ci-dessous, nous décrivons brièvement chacun de ces processus.

1. ***Chimiotaxie:*** La chimiotaxie représente le processus principal de l'algorithme BFO, selon lequel les bactéries se déplacent et s'oriente dans un gradient de nourriture tout au long de son cycle de vie. Cependant, les bactéries effectuent une marche aléatoire dans leur environnement en en alternant entre des déplacements rectilignes, selon un processus appelé la nage (cas de rencontre d'environnement riche en nutriments et sans substances nocives), et des changements d'orientation aléatoires appelés pivotements ou culbutes (cas de rencontre d'environnement défavorable) (Passino, 2002; Farshi and Orujpour, 2021).

   Le mouvement chimiotactique de la bactérie peut donc être décrit comme suit, soit S le nombre total de bactérie présent dans la population, $\theta^t(j,k,l)$ la i^{me} bactérie $(i = 1,2,\ldots,S)$ à la j^{me} étape chimiotactique, à la k^{me} étape de reproduction et à la l^{me} étape d'élimination-dispersion. Le mouvement de la bactérie à l'étape chimiotactique (j+1) est calculé par l'équation (2.6) en fonction de sa postion précédente, de la taille du pas $C(i)$ (run length unit) et d'une direction aléatoire $\Phi(j)$ permettant de décrire les changements derectionnels aléatoires.

$$\begin{aligned} &\theta^i(j+1,k,l) = \theta^i(j,k,l) + C(i)\phi(j), \\ &\phi(j) = \frac{\Delta(i)}{\sqrt{\Delta^T \cdot \Delta(i)}} \end{aligned} \tag{2.6}$$

   Où $\Delta(i) \in \Re^D$ est un vecteur dont les valeurs sont générées aléatoirement dans l'intervalle [-1,1].

   L'evaluation de chaque bactérie est effectuée en calculant son coût par la forumule suivante:

$$J(i,j+1,k,l) = J(i,j+1,k,l) + J_{cc}\left(\theta^i(j+1,k,l), P(j+1,k,l)\right) \tag{2.7}$$

   Le coût lié à une position donnée $J(i,j,k,l)$ est affecté par les forces attractives et répulsives existants entre les bactéries de la population (S), voir l'équation (2.8). Cela dit, si le coût (valeur de la fonction objective) de la i^{me} bactérie à l'étape chimiotactique $(j+1)$ est meilleur que celui à l'étape (j) alors cette bactérie va effectuer un mouvement taille $C(i)$ dans la même de direction appelé la nage (swiming) , le nombre de nage ne doit pas dépasser N_s.

2. ***Essaimage :*** Ce processus permet aux bactéries de se déplacer en groupe. Durant la recherche de nutriments, il existe à la fois une attraction et une répulsion entre

les bactéries individuelles. Ces dernières, bactéries génèrent des informations d'attraction pour permettre aux bactéries individuelles de se déplacer vers le centre de la population, les rassemblant. En même temps, les bactéries individuelles sont maintenues à distance en fonction de leurs informations de répulsion respectives (Passino, 2002; Tang et al., 2017). L'expression des forces attractives et répulsives existant entre les bactéries de la population est donné par:

$$\begin{aligned} J_{cc}(\theta, P(j,k,l)) &= \sum_{i=1}^{S} J_{cc}^{i}\left(\theta, \theta'(j,k,l)\right) \\ &= \sum_{i=1}^{S}\left[-d_{\text{attract}}\ \exp\left(-w_{\text{attract}}\ \sum_{m=1}^{D}\left(\theta_m - \theta_m^i\right)^2\right)\right] \\ &\quad + \sum_{i=1}^{S}\left[h_{\text{repellant}}\ \exp\left(-w_{\text{repellant}}\ \sum_{m=1}^{D}\left(\theta_m - \theta_m^i\right)^2\right)\right] \end{aligned} \tag{2.8}$$

Tel que: $\theta = [\theta_1, \theta_2, \ldots, \theta_{\text{D}}]^T$ est un point dans l'espace de recherche de dimension D, θ_m^i est la dimension m de la i^{me} bactérie, $P(j,k,l)$ l'ensemble des bactéries(la population). d_{attract} , w_{attract} ,$h_{\text{repellant}}$ et $w_{\text{repellant}}$ sont des coefficients différents qui doivent être choisis selon le problème à traiter.

3. ***Reproduction:*** Une les N_c étapes chimiotatactiques sont éffectuées, les bactéries rentrent de l'étape de la reproduction N_re étapes. Dans cette dernière, les bactéries sont triées dans l'ordre croissant de leur coût cumulatif comme suit (Passino, 2002):

$$J_{\text{health}}^{i} = \sum_{j=1}^{N_c+1} J(i,j,k,l) \tag{2.9}$$

Cela signifie que les bactéries faisant partie de la moitié inférieure de la liste meurent, tandis que la moitié la plus saine (les bactéries situées dans un domaine de recherche avec une concentration élevée en nutriments) se reproduit. Chaque bactérie se divise en deux bactéries identiques, placée au même endroit, De cette manière, la population de bactéries reste constante (Passino, 2002; Li et al., 2014; Tang et al., 2017).

4. ***Elimination-Disperssion:*** Au cours du processus de recherche de nutriments,

les bactéries peuvent mourir ou disperser vers d'autres zones. La dispersion permet de favoriser l'exploration de nouvelles régions riche en nutriments. Cependant, chaque bactérie est soumise à une élimination-dispersion avec une probabilité P_{ed}, si la bactérie satisfait cette P_{ed} elle sera remplacée par une autre bactérie générée aléatoirement dans l'espace de solution. Ce qui peut permettre à l'algorithme BFO de sortir de la solution optimale locale, en favorisant la recherche de solution optimal globale (Passino, 2002; Sathya and Kayalvizhi, 2011a). Sachant que le nombre de bactéries dans la population reste constant, car si une bactérie est éliminée, une autre est dispersée à une position aléatoire (Passino, 2002).

L'algorithme 5 résume le déroulement des quatre étapes décrites précédemment.

2.3.5 L'algorithme des Lucioles "Firefly Algorithm"

L'algorithme Firefly a été développé par (Yang, 2008). Il s'agit d'un algorithme d'intelligence en essaim basé sur l'emission/absorption de la luminosité et le comportement d'attraction mutuelle des lucioles. Théoriquement, l'algorithme FA est basé sur les trois règles principales suivantes (Fister et al., 2013; Yang, 2008):

- Toutes les lucioles sont unisexes, ce qui implique que chaque luciole de la population peut s'attirer mutuellement.
- L'attrait est proportionnel à la luminosité et ils diminuent tous deux à mesure que la distance augmente. Ainsi, pour deux lucioles clignotantes, la moins lumineuse se dirigera vers la plus lumineuse. S'il n'y a pas de luciole plus brillante qu'une luciole donnée, elle se déplacera au hasard.
- La luminosité d'une luciole est déterminée par la valeur de la fonction objective.

L'attractivité peut être calculée comme suit:

$$\beta(r) = \beta_0 e^{-\gamma r^2} \tag{2.10}$$

où r représente la distance entre la luciole i et la luciole j (voir l'équation : (2.11)), β_0 est l'attractivité initiale lorsque $r = 0$, et γ est un coefficient d'absorption, qui contrôle la diminution de l'intensité.

$$r_{i,j} = \sqrt{\sum_{d=0}^{D-1} \left(x_{i,d} - x_{j,d}\right)^2} \tag{2.11}$$

Algorithm 5 Algorithme BFO

Initialiser les paramétres : $D, S, N_c, N_s, N_{re}, N_{ed}, p_{ed}, C(i)(i = 1,2,\ldots,S), \theta^i(i = 1,2,\ldots,S)$
pour $(l = 1,2,\ldots,N_{ed})$ (Etapes d'élimination-disperssion) **faire**
  pour $(k = 1,2,\ldots,N_{re})$ (Etapes de reproduction) **faire**
    pour $(j = 1,2,\ldots,N_c)$ (Etapes chimiotactiques) **faire**
      pour toute bactérie $(i = 1,2,\ldots,S)$ **faire**
        Evaluation de la fonction objective $J(i,j,k,l)$ *par l'équation (2.7)*
        $J_{last} = J(i,j,k,l)$
        Pivotement (Tumble):Génératon aléatoire du vecteur $\Delta(i) \in \Re^D$
        Déplacement : Calcule de la position $\theta^i(j+1,k,l)$ *par l'équation (2.6)*
        Evaluation de la fonction objective $J(i,j+1,k,l)$ *par l'équation (2.7)*
        Nage : $m = 0$
        tant que $(m < N_s)$ **faire**
          $m = m + 1$
          si($J(i,j+1,k,l) < J_{\text{last}}$ **) alors**
            $J_{\text{last}} = J(i,j+1,k,l)$
            Déplacement : Calcule de la position $\theta^i(j+1,k,l)$ *par l'équation (2.6)*
            Evaluation de la fonction objective $J(i,j+1,k,l)$ *par l'équation (2.7)*
          sinon
            $m = N_s$
          fin si
        fin tant que
      fin pour
    fin pour
    pour $(i = 1,2,\ldots,S)$ **faire**
      Calcule de : $J^i_{\text{health}} = \sum_{j=1}^{N_c+1} J(i,j,k,l)$
    fin pour
    Trier les bactéries par ordre croissant de J_{health}
  fin pour
  pour $(i = 1,2,\ldots,S)$ **faire**
    Eliminer et disperser la i^{me} *bactérie, avec une probabilité* P_{ed}
  fin pour
fin pour

Tant dis que le mouvement de la luciole i vers la luciole j la plus attractive (plus brillante) est calculé par :

$$x_i^{t+1} = x_i^t + \beta_0 e^{-\gamma r_{i,j}} \left(x_j^t - x_i^t\right) + \alpha(\text{rand}(0,1) - 0.5) \tag{2.12}$$

Tel que : α est une valeur aléatoire entre 0 et 1. Le déroulement de l'algorithme FA est donné par l'algorithme 6.

Algorithm 6 Algorithme Firefly

Générer aléatoirement la population des Lucioles $\boldsymbol{x}_i (i = 1, 2, \ldots, n)$
Définir le coefficient d'absorption de la lumiére γ
t=0
tant que (t<$iteration - max$) **faire**
  pour *(chaque luciole i)* **faire**
    pour *(chaque luciole j)* **faire**
      si (f_i<f_j) **alors**
        Déplacement de luciole i vers la luciole j par l'équation (2.12)
      fin
      Varier l'attractivité en fonction de la distance r
      Evaluer les nouvelles solutions (calcul de la fonction fitness)
    fin
  fin
  Trier les lucioles et donner la meilleure solution globale actuelle g^*
  t=t+1
fin
renvoyer g^*

2.3.6 L'algorithme de Chauve-souris "Bat Algorithm"

L'algorithme de chauve-souris est une approche méta-heuristique récente proposée par (Yang, 2010b; Yang, 2012). L'idée de base de cet algorithme est d'imiter le comportement d'écholocation des micro-chauves-souris.

Toutes les chauves-souris de la population sont associées à un emplacement x_i et à une vitesse v_i, dans un espace de recherche à D dimensions. Cependant, le principe du déroulement de l'algorithme BA se base sur trois règles fondamentales (Yang, 2010b; Cui, Li, and Zhang, 2019) :

- Toutes les chauves-souris utilisent l'écholocation pour mesurer la distance, et elles ont une capacité incroyable à différencier entre la nourriture/proie dès barrières de fond.
- Les chauves-souris volent de façon aléatoire avec une vitesse v_i à la position x_i et une fréquence f_i, variant la longueur d'onde et le volume A_0 afin de trouver des proies. Chaque chauve-souris peut ajuster la longueur d'onde (fréquence) de son impulsion émise ainsi que le taux d'émission d'impulsion $r \in [0, 1]$ et cela en fonction de la proximité de sa proie.
- Bien que l'intensité sonore puisse varier de nombreuses façons, (Yang, 2010b) suppose que l'intensité sonore varie d'une grande valeur (positive) A_0 à une valeur minimale constante A_{min}.

Mouvement des chauves-souris : les règles présentées ci-dessus peuvent être traduites mathématiquement par les équations : (2.13), (2.14) et (2.15) afin d'obtenir les nouvelles solutions x_i et v_i à l'itération $t+1$. x^* représente la meilleure solution trouvée à l'itération t.

$$f_i = f_{\min} + (f_{\max} - f_{\min})\,\text{rand}(0,1) \tag{2.13}$$

$$v_i^{t+1} = v_i^t + \left(x_i^t - x^*\right) f_i \tag{2.14}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{2.15}$$

D'autre part, Yang a conçu une stratégie de recherche locale pour chaque chauve-souris dans le but d'améliorer la capacité de recherche locale de BA ; l'idée est d'utiliser une marche aléatoire comme montré par l'équation suivante (Yang and He, 2013; Yang, 2010b) :

$$x_{new} = x^* + \varepsilon \overline{A^t} \tag{2.16}$$

Tel que : ε est un nombre aléatoire uniformément distribué dans $[-1,1]$, et $\overline{A^t}$ est l'intensité sonore moyenne de toutes les chauves-souris à l'itération t.

Variations de l'intensité sonore et des fréquences d'impulsion : pour chaque chauve-souris i, l'intensité sonore A_i^{t+1} et le taux r_i^{t+1} de l'émission d'impulsions sont mis à jour comme suit (Yang, 2010b):

$$A_i^{t+1} = \alpha A_i^t \tag{2.17}$$

$$r_i^{t+1} = r_i^0[1 - \exp(-\gamma t)] \tag{2.18}$$

où α est un facteur croissant de l'intensité sonore, et γ est un facteur décroissant du taux d'émission d'impulsion.

(Yang, 2010b) a résumé sa proposition par l'algorithme 7.

2.3.7 L'algorithme Grey Wolf Optimizer

Grey Wolf Optimizer est un algorithme méta-heuristique qui s'inspire des loups gris, dévelopé par (Mirjalili, Mirjalili, and Lewis, 2014). En effet, en se basant sur leur

Algorithm 7 Algorithme Bat

Générer aléatoirement la population des Chauves-souris et leurs vitesses $\boldsymbol{x}_i (i = 1, 2, \ldots, n)$ $\boldsymbol{v}_i (i = 1, 2, \ldots, n)$
Initialiser les paramétres : A_i^0, r_i^0, *et* f_{min}, f_{max}
t=0
tant que (t<$iteration - max$) **faire**
  pour *(chaque chauve-souris i)* **faire**
    Mise à jour de v_i *et de* x_i *par les équations: (2.13), (2.14) et (2.15)*
    si(rand <r_i^t) **alors**
      Recalculer la position de la chauve-souris i autour de x^* *avec l'équation: (2.16)*
    fin
    si(rand <A_i^t&$f\left(x - new_i^t\right) < f\left(x^*\right)$) **alors**
      Accepter la nouvelle solution x_i^{t+1} *et la vitesse* v_i^t.
      Diminuer A_i^t *avec l'équation: (2.17) et augmenter* r_i^t *avec l'équation: (2.18)*
    fin
  fin
  Trier les chauves-souris et trouver la meilleure solution x^* *courante.*
  t=t+1
fin
renvoyer x^*

hiérarchie ainsi que sur leur méthodes de chasser et d'attaquer la cible que des formules mathématiques ont été modélisées afin de représenter ce mode de fonctionnement.

Avant de présenter les formules mathématiques, nous allons tout d'abord résumer en ce qui suit le mode de fonctionnement des loups gris (Canis lupus) :

Mode de fonctionnement des loups gris

1. ***La hiérarchie :***

   - *les loups Alpha :* se sont les loups responsables du groupe, ils prennent les décisions sur la chasse, les lieux de campement, le temps du réveil, etc. Tous les autres membres du groupe doivent non seulement respecter les loups Alpha mais aussi suivre à la lettre leurs ordres. A noter que, les loups alpha ne sont pas forcément les plus fort mais plutôt les meilleurs en terme d'organisation et de gestion du groupe (Mirjalili, Mirjalili, and Lewis, 2014; Hassanien and Oliva, 2017).

   - *Les loups Beta :* ces loups ont comme fonction d'aider les loups Alpha à prendre des décisions ainsi que de s'assurer que le reste du groupe applique bien les ordres, ils sont aussi les mieux qualifier de prendre la place d'un

loup alpha en cas de décès (Mirjalili, Mirjalili, and Lewis, 2014; Hassanien and Oliva, 2017).

- *Les loups Omega :* c'est le dernier rang des loups, leurs rôles est de protéger et respecter l'ensemble de la meute (Mirjalili, Mirjalili, and Lewis, 2014; Hassanien and Oliva, 2017).
- *Les loups Delta :* ils sont en dessous des loups Beta mais dominent les loups Omega. Leurs rôles c'est de surveiller le territoire, de protéger la meute, d'aider les Alphas et les Betas en cas de chasse et fournir de la nourriture à la meute et enfin, de soigner et prendre soin des loups faibles, malades et blessés (Mirjalili, Mirjalili, and Lewis, 2014; Hassanien and Oliva, 2017).

2. ***La chasse :*** Les techniques de chasse suivies par les loups gris se basent principalement sur l'une des trois phases : **a)** Poursuivre, chasser et approcher la proie, **b)** Poursuivre, encercler et harceler la proie jusqu'à ce qu'elle s'arrête de bouger, ou **c)** Attaquer en direction de la proie (Mirjalili, Mirjalili, and Lewis, 2014; Hassanien and Oliva, 2017).

Formulation mathématique

Dans cette partie, les auteurs ont formulé mathématiquement, non seulement la hiérarchie sociale des loups gris mais aussi la technique de chasse de ces derniers.

- la hiérarchie sociale a été formulée de telle sorte à ce que la solution la plus pertinente est considérée comme la solution Alpha, juste après vient la solution Beta et Delta et en fin la solution appelé Omega.
- Afin de représenter mathématiquement l'encerclement de la proie, les auteurs ont mis en place l'équation suivante :

$$\begin{aligned} \vec{D} &= \left| \vec{C} \cdot \vec{X}_p(t) - \vec{X}(t) \right| \\ \vec{X}(t+1) &= \vec{X}_p(t) - \vec{A} \cdot \vec{D} \end{aligned} \tag{2.19}$$

où t indique l'itération courante,$\vec{A}$ et $\vec{C}$ sont des vecteurs de coefficients, $\vec{X}_p$ est le vecteur de position de la proie, et $\vec{X}$ indique le vecteur de position d'un loup. Les vecteurs $\vec{A}$ et $\vec{C}$ sont calculés comme suit :

$$\begin{aligned} \vec{A} &= 2\vec{a} \cdot \vec{r}_1 - \vec{a} \\ \vec{C} &= 2 \cdot \vec{r}_2 \end{aligned} \tag{2.20}$$

où les composantes de $\vec{a}$ sont linéairement réduites de 2 à 0 au cours des itérations, tandis que $\vec{r_1}$ et $\vec{r_2}$ sont des vecteurs aléatoires dans l'intervalle $[0, 1]$.

- Quant à la chasse, les solutions Alpha, Beta et Delta représentent une meilleure prédiction de la position de la proie. Ensuite et en se basant sur ces solutions, ils obligent les autres solutions à mettre à jour leur positions. La formulation mathématique dans cette étape est la suivante :

$$\begin{aligned} &\vec{D}_\alpha = \left|\vec{C}_1 \cdot \vec{X}_\alpha - \vec{X}\right|, \vec{D}_\beta = \left|\vec{C}_2 \cdot \vec{X}_\beta - \vec{X}\right|, \vec{D}_\delta = \left|\vec{C}_3 \cdot \vec{X}_\delta - \vec{X}\right| \\ &\vec{X}_1 = \vec{X}_\alpha - \vec{A}_1 \cdot \left(\vec{D}_\alpha\right), \vec{X}_2 = \vec{X}_\beta - \vec{A}_2 \cdot \left(\vec{D}_\beta\right), \vec{X}_3 = \vec{X}_\delta - \vec{A}_3 \cdot \left(\vec{D}_\delta\right) \\ &\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3} \end{aligned} \tag{2.21}$$

- Finalement et afin de modéliser l'attaque, les auteurs ont choisi de décroitre la valeur de $\vec{a}$.

- Pour chercher une proie, les loups gris divergent puis ils convergent afin d'attaquer. Les auteurs ont modélisé la divergence en utilisant le vecteur $\vec{A}$ avec des valeurs aléatoires supérieures à 1 ou inferieures à -1. cet intervalle va obliger les loups a diverger. En outre, afin d'éviter la stagnation, les auteurs affectent des valeurs aléatoires à tout moment à C afin de mettre l'accent sur l'exploration. Ce paramètre peut aussi représenter les obstacles naturels que les loups rencontrent lors de la phase d'approche.

L'algorithme 8 résume la proposition des auteurs.

2.3.8 L'algorithme Moth-flame optimization

L'algorithme Moth-Flame Optimization (Mirjalili, 2015) est une méta-heuristique inspirée de la façon dont les papillons de nuit naviguent en se basant sur la lumière de la lune. En effet, la technique utilisée va leur permettre de se déplacer en ligne droite sur une longue distance. Cependant, ces papillons sont piégés dans un parcours en spirale inutile et/ou mortel autour des lumières artificielles (Mirjalili, 2015; Shehab et al., 2020). Ce comportement a été formulé mathématiquement en passant par les étapes suivantes :

1. **La position des papillons de nuit :** dans cette étape, les auteurs ont supposé que les papillons peuvent se déplacer en $1-D$, $2-D$ et $3-D$ ou dans un espace hyperdimensionnel. Supposons que n est le nombre de papillons et d est

Algorithm 8 Algorithme de Grey Wolf Optimizer

Initialiser la population de loups gris $X_i (i = 1, 2, \ldots, n)$
Initialiser a, A, *et* C
Calculer le rendement de chaque agent de recherche
$X_a =$ *le meilleur agent de recherche*
$X_\beta =$ *le deuxième meilleur agent de recherche*
$X_\delta =$ *le troisième meilleur agent de recherche*
tant que ($t <$ Nombre maximal d'itérations) **faire**
  pour (chaque agent de recherche) **faire**
    Mettre à jour la position de l'agent de recherche actuel par l'équation (2.21)
  fin
  Mettre à jour a, A, *and* C
  Calculer le rendement de tous les agents de recherche
  Mettre à jour X_α, X_β, *and* X_δ
  $t = t + 1$
fin
renvoi X_a

le nombre de dimension, la matrice suivante représente l'ensemble de papillons de nuit (Mirjalili, 2015):

$$M = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & \cdots & m_{1,d} \\ m_{2,1} & m_{2,2} & \cdots & \cdots & m_{2,d} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m_{n,1} & m_{n,2} & \cdots & \cdots & m_{n,d} \end{bmatrix} \tag{2.22}$$

Ainsi, le tableau suivant stocke les valeurs de qualité correspondantes pour tous les papillons (Mirjalili, 2015):

$$OM = \begin{bmatrix} OM_1 \\ OM_2 \\ \vdots \\ OM_n \end{bmatrix} \tag{2.23}$$

2. **La position des flammes :** similairement à la position des papillons, la position des flammes est aussi représentée par une matrice et un tableau de stockage des valeurs de qualité comme suit (Mirjalili, 2015) :

$$F = \begin{bmatrix} F_{1,1} & F_{1,2} & \cdots & \cdots & F_{1,d} \\ F_{2,1} & F_{2,2} & \cdots & \cdots & F_{2,d} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ F_{n,1} & m_{n,2} & \cdots & \cdots & F_{n,d} \end{bmatrix} \tag{2.24}$$

$$OF = \begin{bmatrix} OF_1 \\ OF_2 \\ \vdots \\ OF_n \end{bmatrix} \tag{2.25}$$

3. **Le problème d'optimisation :** pour se rapprocher de l'optimum global, l'algorithme MFO définit le problème d'optimisation comme suit :

$$MFO = (I, P, T) \tag{2.26}$$

   - I est une fonction qui génère une population aléatoire de papillons de nuit et les valeurs de fitness correspondantes : $I : \emptyset \rightarrow \{M, OM\}$.
   - P est une fonction principale, qui déplace les papillons dans l'espace de recherche. Cette fonction reçoit la matrice de M et renvoie éventuellement sa matrice mise à jour : $P : M \rightarrow M$.
   - Quant à la fonction T, elle renvoie vrai si le critère de terminaison est satisfait et faux si le critère de terminaison n'est pas satisfait : $T : M \rightarrow \{true, false\}$.

4. **L'orientation transversale :** l'inspiration principale de cet algorithme est l'orientation transversale des papillons en fonction de la flamme. De ce fait, pour représenter mathématiquement ce comportement (Mirjalili, 2015; Abd El Aziz, Ewees, and Hassanien, 2017), la position de chaque papillon doit être mise à jour en fonction de la position de la flamme comme suit : $M_i = S(M_i, F_i)$ où M_i indique le i^{me} papillon, F_j indique la j-ème flamme, et S est la fonction spirale.

   Les trois conditions à respecter lors de l'utilisation d'une spirale logarithmique, sont les suivantes :

   - Le point initial de la spirale doit commencer du papillon.
   - Le point final de la spirale doit être la position de la flamme.

- La fluctuation de la portée de la spirale ne doit pas dépasser l'espace de recherche.

Ainsi, la spirale logarithmique peut être représentée comme suit :

$$S\left(M_i, F_j\right) = D_i \cdot e^{bt} \cdot \cos(2\pi t) + F_j \tag{2.27}$$

où D_i indique la distance du i^{eme} papillon pour la j^{eme} flamme, b est une constante pour définir la forme de la spirale logarithmique, et t est un nombre aléatoire dans $[1, 1]$. D est calculé comme suit :

$$D_i = \left|F_j - M_i\right| \tag{2.28}$$

Pour éviter de stagner dans un optimum local, les solutions optimales ont été stockées à chaque répétition, et les papillons volent autour de la flamme la plus proche en utilisant les matrices OF et OM (Mirjalili, 2015).

5. **Le problème du nombre de flammes :** en effet, la mise à jour de la position des papillons par rapport à n emplacements différents de flammes dans l'espace de recherche peut dégrader l'exploitation des meilleures solutions (Mirjalili, 2015). Par conséquent, réduire le nombre de flammes est la meilleure solution pour résoudre ce problème en utilisant la formule suivante :

$$\text{flame no} \;=\; \text{round}\left(N - l * \frac{N-1}{T}\right) \tag{2.29}$$

où l est le nombre actuel d'itération, N est le nombre maximum de flammes, et T indique le nombre maximum d'itérations.

L'algorithme 9 résume la proposition des auteurs.

2.3.9 L'algorithme Whale Optimization

Whale optimization (Mirjalili and Lewis, 2016) est un algorithme metaheuristique inspiré de baleines à bosse (humpback whales), particuliérement en se basant sur leur méthode de chasse par filet à bulles. Avant d'entrer dans les détails des formules mathématiques, nous allons tout d'abord expliquer brièvement le comportement des baleines à bosse lors de la chasse.

Algorithm 9 Algorithme de Moth-flame optimization

Initialiser les parametres de MFO $X_i(i = 1, 2, \ldots, n)$
Initialiser la position des papillons M_i *aléatoirement*
Calculer le rendement de chaque agent de recherche
pour (i = 1 to n) **faire**
    Calculer la fonction f_i
fin
tant que (*itérations* $\leq$ *Max itération*) **faire**
    Mettre à jour M_i
    Calculer le nombre de flammes
    Evaluer la fonction f_i
    si *itération* == *1* **alors**
        $F = sort(M)$ *et* $OF = sort(OM)$
    sinon
        $F = sort(M_{t-1}, M_t)$ *et* $OF = sort(M_{t-1}, M_t)$
    fin
    pour (i = 1 to n) **faire**
        pour (j = 1 to d) **faire**
            Mettre à jour r et t
            Calculer la valeur de D
            Mettre à jour $M(i, j)$
        fin
    fin
fin
Affiche la meilleur solution

Lors de la recherche de la nourriture, les baleines à bosse commencent par créer des bulles distinctes le long d'un cercle ou d'une trajectoire en forme de 9. Deux manœuvres associées aux bulles ont été observées: **a)** spirales ascendantes et **b)** doubles boucles. Les auteurs se sont focalisés sur la première manœuvre, où les baleines à bosse plongent à environ $12m$ de profondeur, puis commencent à créer des bulles en forme de spirale autour de leur proie et remontent vers la surface (Mirjalili and Lewis, 2016). Nous allons maintenant présenter le modèle mathématique de leur méthode de fonctionnement (i.e., l'encerclement de la proie, la manœuvre d'alimentation par filet à bulles en spirale et la recherche de la proie):

1. Pour l'encerclement de la proie et la mise à jour de la position des agents de recherche, les auteurs ont mis en place les deux équations suivantes (Mirjalili and Lewis, 2016):

$$\vec{D} = \left| \vec{C} \cdot \vec{X}^*(t) - \vec{X}(t) \right| \tag{2.30}$$

$$\vec{X}(t+1) = \vec{X}^*(t) - \vec{A} \cdot \vec{D} \tag{2.31}$$

où t indique l'itération courante, $\vec{A}$ et $\vec{C}$ sont des vecteurs de coefficients, $\vec{X}^*$ est le vecteur de position de la meilleure solution, et $\vec{X}$ est le vecteur de position. A noter que, $\vec{X}^*$ est mis à jour à chaque itération. Les vecteurs $\vec{A}$ et $\vec{C}$ sont calculés comme suit :

$$\begin{aligned} \vec{A} &= 2\vec{a} \cdot \vec{r}_1 - \vec{a} \\ \vec{C} &= 2 \cdot \vec{r}_2 \end{aligned} \tag{2.32}$$

où les composantes de $\vec{a}$ sont linéairement réduites de 2 à 0 au cours des itérations, tandis que $\vec{r}$ est un vecteur aléatoire dans l'intervalle $[0, 1]$.

2. La manœuvre d'alimentation par filet à bulles a été modélisée suivant deux approches :

   - **Mécanisme d'encerclement rétrécissant :** en diminuant la valeur de $\vec{a}$, l'intervalle de $\vec{A}$ va aussi diminuer, ce qui va permettre à mettre à jour la position de l'agent de recherche actuel vers l'agent de recherche avec la meilleure position (Mirjalili and Lewis, 2016).
   - **Mise à jour de la position spiral :** une équation en spirale est créée entre la position de la baleine et de la proie pour imiter le mouvement en forme hélicoïdale des baleines (Mirjalili and Lewis, 2016) :

   $$\vec{X}(t+1) = \overrightarrow{D'} \cdot e^{bl} \cdot \cos(2\pi l) + \overrightarrow{X^*}(t) \tag{2.33}$$

   où $\overrightarrow{D'} = \left|\vec{X}^*(t) - \vec{X}(t)\right|$ indique la distance de la i-ème baleine à la proie (meilleure solution obtenue jusqu'à présent), b est une constante permettant de définir la forme de la spirale logarithmique, l est un nombre aléatoire dans $[-1, 1]$.

3. Passant maintenant à la recherche de proies. Vu que les baleines à bosse cherchent leurs proies de façon aléatoire en fonction de la position des autres baleines, les auteurs ont choisi d'affecter une valeur aléatoire à $\vec{A}$ a condition qu'elle soit supérieur à 1 ou inféreirur à -1 afin de forcer l'agent de recherche à s'éloigner de la baleine référence (Mirjalili and Lewis, 2016; Abd El Aziz, Ewees, and Hassanien, 2017). Dans cette étape le modèle mathématique est le suivant :

$$\vec{D} = \left|\vec{C} \cdot \overrightarrow{X_{\text{rand}}} - \vec{X}\right| \tag{2.34}$$

$$\vec{X}(t+1) = \overrightarrow{X_{\text{rand}}} - \vec{A} \cdot \vec{D} \tag{2.35}$$

où $\vec{X_{rand}}$ est un vecteur de position aléatoire (une baleine aléatoire) choisi dans la population actuelle.

L'algorithme 10 résume la proposition des auteurs.

Algorithm 10 Algorithme de Whale Optimization

Initialiser la population des baleines $X_i (i = 1, 2, \ldots, n)$
Calculer la fonction fitness pour chaque agent de recherche
$X^* =$ *le meilleur agent de recherche*
tant que ($t <$ *le nombre max d'itération*) **faire**
  pour *(chaque agent de recherche)* **faire**
    Mettre à jour a, A, C, l, *et* p
    si $(p < 0.5)$ **alors**
      si $(|A| < 1)$ **alors**
        Mettre à jour la position de la recherche actuelle par l'équation (2.30)
      sinon
        Sélectionner un agent de recherche aléatoire (X_{rand})
        Mettre à jour la position de la recherche actuelle par l'équation (2.35)
      fin
    sinon
      Mettre à jour la position de la recherche actuelle par l'équation (2.33)
    fin
  fin
  Vérifier si un agent de recherche dépasse l'espace de recherche et le modifier
  Calculer la fonction fitness de chaque agent de recherche
  Mettre à jour X^* *s'il existe une meilleure solution*
  $t = t + 1$
fin
renvoie X^*

Bien que les algorithmes méta-heuristiques aient connu un essor considérable dans la résolution des problèmes d'optimisation combinatoire, leur application aux problèmes du monde réel demeure couteuse en termes de temps de calcul et de quantité de mémoire. Ceci est dû au fait qu'ils soient étroitement liés à la taille du problème et au désir d'obtention d'une certaine qualité de solution. Afin d'améliorer la performance des méta-heuristiques, on a eu recours au calcul parallèle.

2.4 Les méta-heuristiques paralléles

Le développement de méta-heuristiques parallèle vise à traiter des instances de problèmes plus importantes que ce qui est réalisable par des méthodes séquentielles, et

ce dans des temps de calcul raisonnables, et à améliorer la qualité des solutions par l'échange d'informations entre méta-heuristiques (Rego and Alidaee, 2006; Crainic and Hail, 2005). Le parallélisme tend aussi à améliorer la robustesse de la méthode, c'est-à-dire sa capacité à offrir un niveau de performance élevé et constant sur une grande variété de problèmes et de caractéristiques d'instances (Rego and Alidaee, 2006; Crainic and Hail, 2005; Crainic, 2019).

Le calcul parallèle est une stratégie utilisée en informatique dans le but de réduire le temps nécessaire à la résolution d'un problème séquentiel. Il consiste à décomposer la charge de calcul totale en sous tâches qui seront exécutées simultanément sur les processeurs disponibles.

Plusieurs méta-heuristiques parallèles ont été proposées et qui peuvent être classées de différentes manières; (Crainic and Toulouse, 1998; Cung et al., 2002; Alba, 2005; Gras et al., 2003) ont établi une classification des différentes implémentations parallèles retrouvées dans le monde des méta-heuristiques. Cependant, l'implémentation parallèle utilisée dans notre travail fait partie de la classification de (Crainic and Toulouse, 1998).

La classification de Crainic et Toulouse (1998) contient trois types de parallélisme :

1. Le premier type représente le parallélisme fonctionnel, une stratégie de parallélisation des opérations à l'intérieur d'une itération de la méthode méta-heuristique. Il est considéré comme étant une parallélisation de bas niveau dont le but est d'accélérer le calcul sans chercher à effectuer une meilleure exploration. Cette méthode de résolution parallèle est la même que son équivalent séquentiel mais plus rapide (Crainic, 2019).

2. L'idée générale du second type est de décomposer le domaine du problème, ou l'espace de recherche en sous-problèmes et à utiliser la puissance de calcul afin de résoudre ces sous-problèmes en parallèle. La méthode de recherche de l'approche parallèle résultante est cependant différente de celle de la méthode séquentielle correspondante. La mise en application de ce type de parallélisme est généralement basée sur un modèle maître-esclave ; le maitre se charge de déterminer les partitions qui doivent être explorées par les esclaves de manière concurrente et indépendante. Une fois le traitement terminé, le maitre assemble les solutions partielles trouvées par les esclaves en une solution complète au

problème. Plusieurs travaux ont été menés en se basant sur cette stratégie de parallélisme (Alba, 2005; Crainic, 2008; Crainic, Davidović, and Ramljak, 2014; Talbi, 2009; Pedemonte, Nesmachnow, and Cancela, 2011).

3. La recherche multiple représente le dernier type de parallélisme de cette classification, elle consiste à effectuer une recherche quasi-exhaustive en utilisant plusieurs processus de recherche « thread » de manière simultanée et avec différents degrés de synchronisation et de coopération. Ces processus peuvent communiquer entre eux soit lors de la recherche ou bien à la fin dans le but de déterminer la meilleure solution. Cette stratégie offre aussi la possibilité d'utiliser plusieurs méta-heuristiques de façon concurrente, soit par des approches *indépendantes* ou bien les approches *coopératives* (Alba, 2005; Crainic and Toulouse, 1998).

L'implémentation d'un algorithme parallèle dépend de l'environnement de programmation utilisé, qui est lié à l'architecture matérielle sur laquelle il va être exécuté. Selon la classification de Flynn (Flynn, 1972), il existe quatre architectures parallèles; l' architecture SISD (Single Instruction, Single Data) qui correspond aux ordinateurs monoprocesseur permettant d'exécuter une seule unité d'instructions à la fois sur un ensemble unique de données. L'architecture SIMD (Single Instruction, Multiple Data), permet aussi d'exécuter un seul flux d'instructions à la fois, mais sur plusieurs ensembles donnés. Tandis que l'architecture MISD (Multiple Instruction, Single Data) permet d'effectuer différentes opérations sur les mêmes données. L'architecture MIMD (Multiple Instruction, Multiple Data) est celle la plus répandue et la plus puissante de Flynn, elle permet à plusieurs processeurs d'exécuter simultanément différentes unités d'instructions sur différents ensembles de données. Cette catégorie se divise en trois classes fondamentales : les architectures MIMD à mémoire partagée, les architectures MIMD à mémoire distribuée et celles à mémoire hybride (Tanenbaum, 2012).

2.4.1 Les mesures de performances

Plusieurs métriques ont été proposées pour évaluer les performances d'algorithmes parallèles. Celles les plus couramment utilisées sont l'accélération et l'efficacité.

- ***L'accélération (speed-up)***: permet d'évaluer le gain en temps de calcul obtenu par l'algorithme parallèle par rapport à sa version séquentielle. Il est calculé comme le rapport entre le temps d'exécution de l'algorithme séquentiel (T_1) et le temps d'exécution de la version parallèle utilisant p processeurs (T_p), comme

suit :

$$S_p = \frac{T_1}{T_p} \tag{2.36}$$

On distingue trois types d'accélérations : sous-linéaire (inférieure au nombre de processeurs $S_p < p$), linéaire (égale au nombre de processeurs $S_p = p$) et super-linéaire (supérieure au nombre de Processeurs $S_p > p$). Le cas idéal pour un algorithme parallèle est d'atteindre une accélération linéaire, cela signifie que le temps nécessaire pour l'exécuter diminue proportionnellement à l'ensemble de données utilisées, mais les accélérations sous-linéaires sont les plus répandues en raison de temps nécessaires à la communication et à la synchronisation entre processus.

- ***L'efficacité (efficiency)*** : L'efficacité de calcul donnée par l'équation (2.37) est la valeur normalisée de l'accélération, concernant le nombre de processeurs utilisés lors d'exécution d'un algorithme parallèle. Cette métrique permet de comparer différents algorithmes, éventuellement exécutés sur des plates-formes informatiques non identiques. Plus la valeur de l'efficacité est proche de 1, meilleures sont les performances. Une efficacité égale à 1 correspond à une accélération linéaire.

$$E_p = \frac{S_p}{p} = \frac{T_1}{p \times T_p} \tag{2.37}$$

2.5 Conclusion

Dans ce chapitre, nous avons présenté le concept de l'optimisation mono-objective basé sur l'utilisation des algorithmes méta-heuristiques en se focalisant particulièrement sur les méta-heuristiques à base de population de solutions, celles-ci étant l'objet de notre travail. Nous avons ensuite abordé quelques notions de la parallélisation des méta-heuristiques. Durant ces quatre décennies, de nombreuses méthodes méta-heuristiques ont été développées vu leur efficacité à résoudre des problèmes complexes d'optimisation n'ayant pas été efficacement résolus par les heuristiques et les méthodes d'optimisation classiques. Parmi ces problèmes, figure celui du traitement d'image, qui fera l'objet du chapitre suivant.

Chapitre 3

Traitement d'image

3.1 Introduction

Depuis que le traitement de l'image analogique a été remplacé par le traitement de l'image numérique, les technologies d'information n'ont pas cessé d'évoluer. Faisant ainsi de l'image numérique la source principale de l'information, elle est devenue présente dans différents domaines de la vie (les loisirs, la médecine, les véhicules autonomes, la surveillance, etc). Néanmoins, l'analyse de toutes les scènes contenues dans l'image n'est pas forcement nécessaire, d'où l'avènement des différentes techniques du traitement d'images. Ce dernier a pour but d'extraire les caractéristiques permettant d'identifier les objets de la scène en employant différentes méthodes. La segmentation d'image, à laquelle nous nous intéressons dans ce travail, est parmi ces méthodes qui représente une tâche fondamentale pour l'analyse et l'interprétation de la scène. Elle consiste à diviser l'image numériques en régions homogénes plus significatives.

3.2 Généralités sur le traitement d'images numériques

De nos jours, les techniques de traitement de l'image numérique sont largement répandues dans de multiples domaines : l'imagerie médicale, la post-production de films, les jeux,la télédétection, la vision par ordinateur, etc. Cependant, l'intérêt porté à ces techniques résulte de deux domaines d'application fondamentaux : l'amélioration des informations contenues dans l'image pour une bonne interprétation humaine, et le traitement des données pour une meilleure utilisation ultérieure (le stockage, la transmission, la perception machine, l'extraction d'information, etc).

Mais qu'est-ce qu'une image numérique ?

3.2.1 L'image numérique

Une image peut être définie comme une fonction bidimensionnelle F(x,y) où x et y sont les coordonnées spatiales. L'amplitude F associée aux coordonnées x et y représente l'intensité où le niveau de gris à ce point là (Gonzalez and Woods, 2018).
En réalité, F est continue sur x et y mais en pratique elle a une valeur discrète dans une image numérique; chaque élément référé par x et y est appelé pixel (Gonzalez and Woods, 2018) .

L'image numérique est aussi définie comme étant une matrice à deux dimenssions (2D) de valeurs numériques discretes (Russ, 2011). La Figure 3.1 et l'équation (3.1) montrent un exemple d'une représentation matricielle d'une image.

Chaque élément de cette matrice est appelé un élement d'image (*pictur element)*, pixel ou pel (Bovik, 2009). Autrement dit : *un pixel* représente l'unité de mesure qui définit l'image.

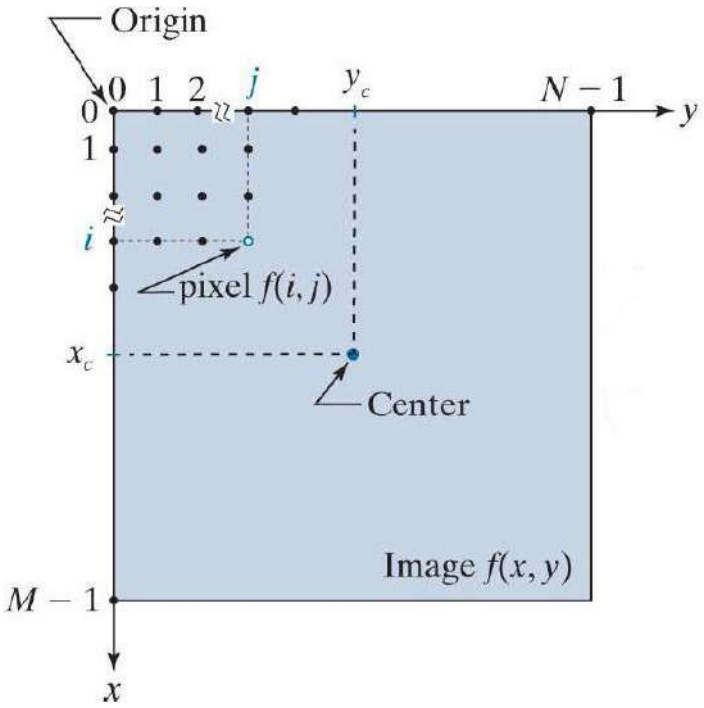


FIGURE 3.1: Représentation matricielle d'une image (Gonzalez and Woods, 2018)

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix} \tag{3.1}$$

3.2.2 Résolution d'une image numérique

La résolution définit la qualité de l'image (nombre total de pixels), elle est calculée par le produit des dimensions de l'image (Nombre de lignes x Nombre de colonnes). Plus l'image contient de pixels plus elle est de haute qualité (Gonzalez and Woods, 2018). Une image avec une faible résolution montre l'effet des escaliers ou du damier (chessboard) (Gonzalez and Woods, 2018). La Figure 3.2 montre la même image avec différentes résolutions.

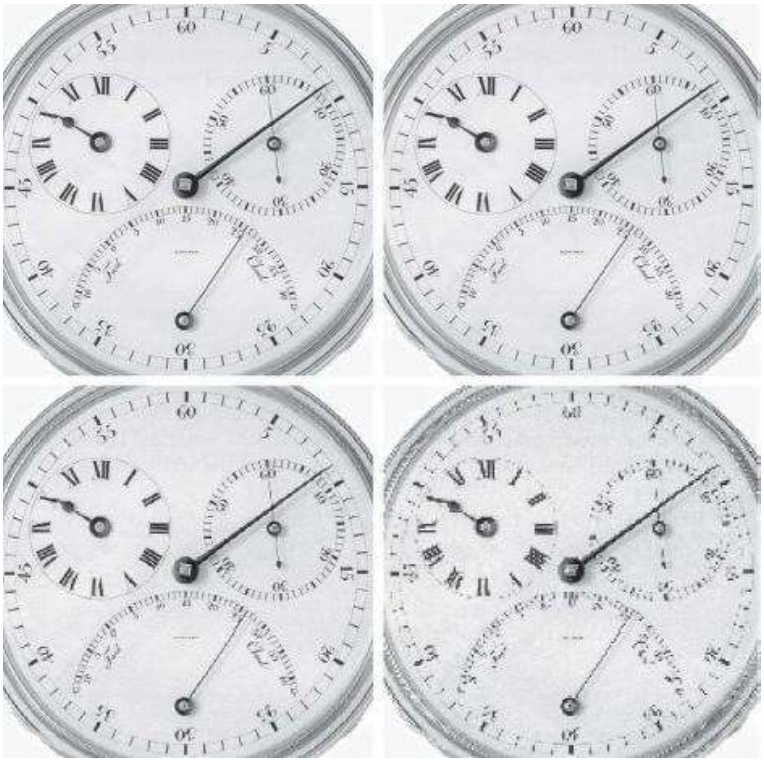


FIGURE 3.2: L'effet de la réduction de la résolution spatiale (Gonzalez and Woods, 2018)

3.2.3 Profondeur d'une image numérique

La profondeur représente la quantité d'information que peut prendre un pixel, elle est définie en nombre de bits par pixel. Ainsi, une image numérique d'une profondeur

de K bits par pixel compte 2^k valeurs (couleurs) possibles.

Les images, selon leur profondeur, peuvent être classifiées en trois catégories :

1. **Images binaires** : Se sont des images particulières pour lesquelles un pixel ne prend que deux valeurs soit le 0 (noir) ou le 1 (blanc). Leur profondeur est de 1 bit par pixel. Une image binaire est généralement issue d'une opération de segmentation (la binarisation). La Figure 3.3 représente une image binarisée par la méthode d'OTSU.

FIGURE 3.3: Binarisation de l'image Boboon par la méthode d'OTSU

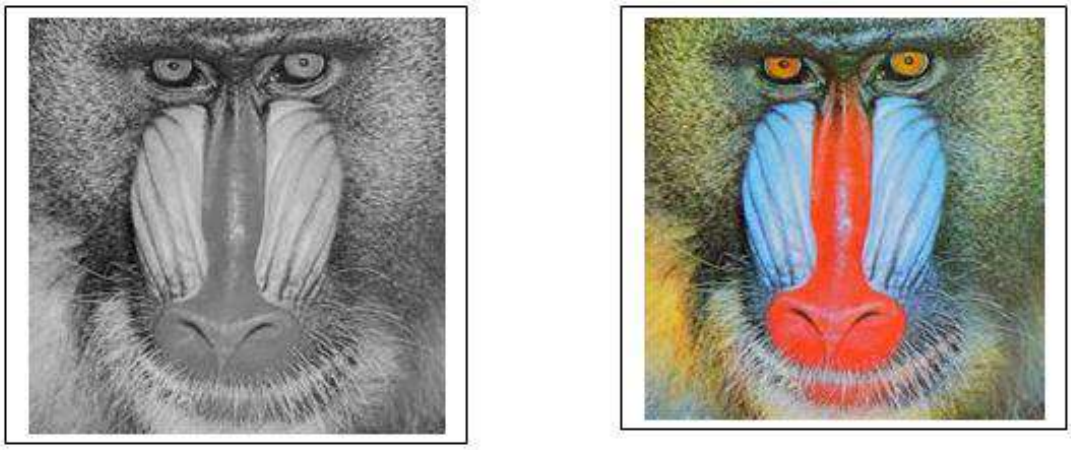
2. **Images en niveau de gris** : Se sont des images monochromes avec 8 bits pour chaque pixel, ce qui fait que l'image contient 256 (2^8) niveaux de gris.

3. **Images en couleur** : Ce type d'image est constitué de trois composantes, chacune d'elle est de taille NxM (nombre de lignes x nombre de colonnes). La combinaison de ces trois composantes forme une couleur, donc, un pixel est représenté par trois valeurs sur trois matrices (Koschan and Abidi, 2008). Leur profondeur est de 24 bits par pixel.

La Figure 3.4 représente la même image dans ces deux derniers types abordés.

3.2.4 Capture, acquisition et stockage de l'image

Dans les systèmes d'acquisition des images numériques, un capteur capte l'image naturelle. Cette dernière est numérisée afin qu'elle puisse être traitée par la machine. L'image est divisée en plusieurs petits élements (pixels), voir Figure 3.5. Parmi les



(a) image en niveau de gris (b) image en couleur

FIGURE 3.4: Représentation de l'image " Baboon " dans l'espace de couleur RGB et en niveau de gris

capteurs d'images existants on retrouve CCD[1] et CMOS[2] (Gonzalez and Woods, 2018; Nixon and Aguado, 2019). Les images numériques sont obtenues en convertissant un signal continu en un signal numérique qui ne contient que des 0 et des 1 (Bovik, 2009).

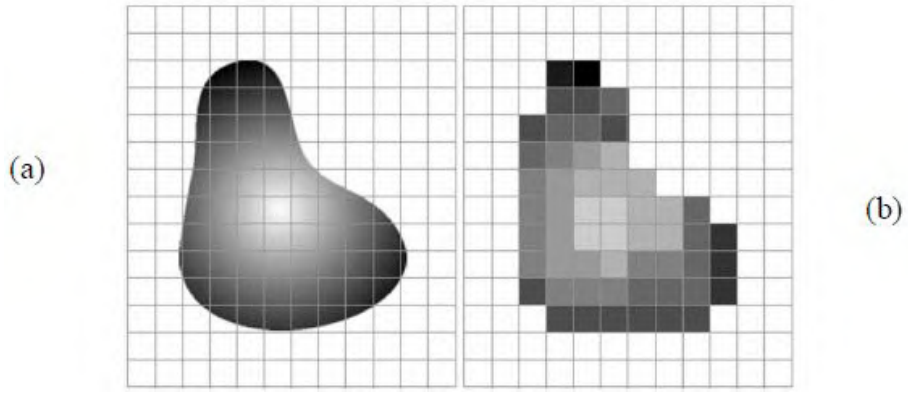


FIGURE 3.5: (a) L'image continue projetée sur une matrice de capteurs. (b) Résultat de l'échantillonnage et la quantification d'image. (Gonzalez and Woods, 2018)

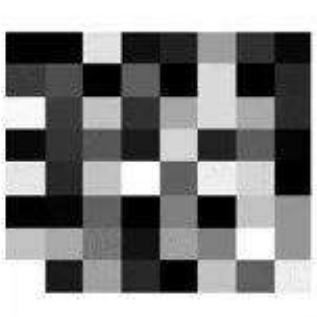
Les deux principales fonctions de la numérisation sont : l'échantillonnage et la quantification (Bovik, 2009).

- ***L'echantillonnage*** : Consiste à choisir le nombre nécessaire de points représentants l'image (Crane, 1996).

[1]Charge Coupled Device
[2]Complementary Metal Oxide Semiconductor

- ***La quantification*** : Consiste à transformer le signal continu en un signal discret puis l'affecter à une case bien déterminée. Le convertisseur analogique numérique (CAN) est le dispositif qui se charge de cette opération (Crane, 1996; Chan and Shen, 2005). Ce dispositif quantifie le signal continu afin d'obtenir un numéro qui représente la valeur d'un pixel.

La Figure 3.6 montre une image numérique de taille 8x8 avec 8 bits par pixel, et sa matrice correspondante.



1	8	219	51	69	171	81	41
94	108	20	121	17	214	15	74
233	93	197	83	177	215	183	78
41	84	118	62	210	71	122	38
222	73	197	248	125	226	210	5
35	36	127	5	151	2	197	165
196	180	142	52	173	151	243	164
254	62	172	75	21	196	126	224

FIGURE 3.6: Numérisation d'image (Shih, 2010)

3.2.5 Traitement d'image numérique

Au début, le traitement d'images n'était utilisé que dans la science spatiale. D'ailleurs, cette dernière est la raison directe de l'avènement et de l'évolution des méthodes du traitement d'images (Gonzalez and Woods, 2018), dès lors, elles sont appliquées dans presque tous les domaines : domaine médecale, transmission et encodage, sismographie, télédétection, vision par ordinateur, robotique, reconnaissance de forme, imagerie microscopique, etc. Cependant, le traitement d'image consiste à appliquer des opérations mathématiques ou bien des algorithmes sur des images numérisées. Son utilisation permet de restaurer les originales, d'améliorer le contenu visuel et d'extraire des informations pertinentes.

Les opérations du traitement d'image peuvent être regroupées sous un cadre général de l'ingénierie de l'image (IE : Image Engineering) (Tyagi, 2018), qui se compose de trois couches fondamentales (figure 3.7). Le traitement d'image (couche basse), l'analyse d'image (couche intermédiaire) et la compréhension et l'interprétation de l'image (couche haute) (Zhang, 2006) :

- **Traitement de bas niveau (Couche basse) :** un prétraitement des images, généralement des opérations initales telles que : la réduction de bruit, l'amélioration du contraste, etc.
- **Traitement intermédiaire :** ce niveau implique des operations essentielles : la segmentaiton, la compression, la détection de contours, etc.
  La qualité des résultats obtnus à ce niveau influence la pertinence des systémes de haut niveau.
- **Traitement de haut niveau (Couche haute) :** ce traitement est souvent lié à la vision par ordinateur. Il traite des entités de nature symbolique pour la compréhension et l'interprétation de l'image.

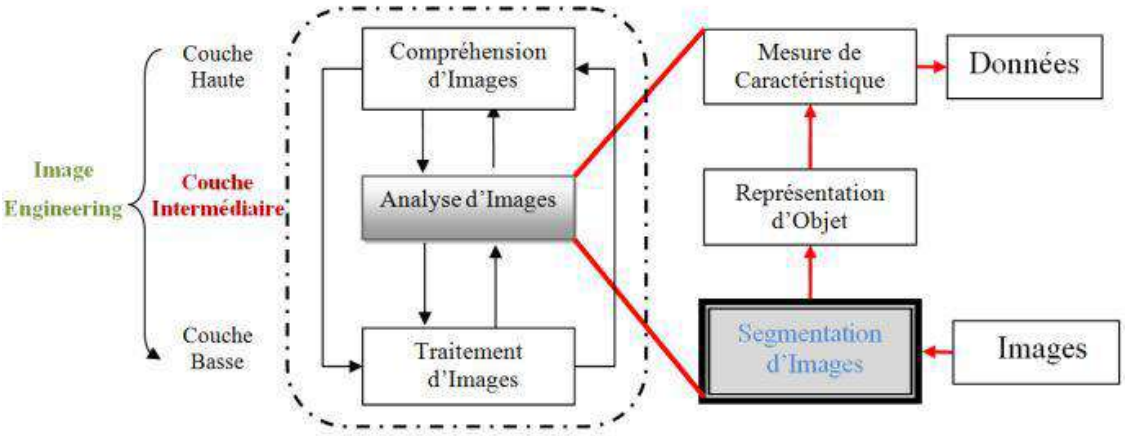


FIGURE 3.7: Ingéneirie de l'image et segmentation d'image (Zhang, 2006)

La segmentation d'image est l'une des tâches les plus critiques de l'analyse automatique d'images. Elle consiste à subdiviser une image en ses parties constitutives et à extraire ces parties d'intérêt (objets). Ainsi, les résultats de la segmentation ont une influence considérable sur la précision de l'évaluation des caractéristiques (mesure de caractéristiques) (Zhang and Gerbrands, 1994).

Dans ce qui suit nous allons aborder les differentes méthodes de la segmentation d'images, Toutefois, l'accent est mis sur les méthodes de segmentation basées sur les méthodes de seuillage (Thresholding) .

3.3 Segmentation d'image

La segmentation est un processus qui divise une image en régions ou objets ayant des caractéristiques similaires (Haralick and Shapiro, 1985; Zhang, Fritts, and Goldman, 2008).

Formellement, soit I une image et soient $R_i(i = 1, 2, \ldots, n)$ des régions disjointes non vides. La segmentation d'image doit satifaire les conditions suivantes (Horowitz and Pavlidis, 1976; Monga and Wrobel, 1987; Fu and Mui, 1981b):

1. $\cup_{i=1}^{n} R_i = I$ (L'union des régions ségmentées = l'image entière).
2. Pour tout i et j, $i \neq j : R_i \cap R_j = \emptyset$ (pas de chevauchement entre les régions).
3. Pour tout $i = 1, 2, \ldots, n$, il doit y avoir $P(R_i) = Vrai$ (similarité entre les pixels de la même région).
4. Pour tout $i \neq j$, il exite $P\left(R_i \cup R_j\right) = Faux$ (dissimilarité entre les pixels des régions distinctes).
5. Pour tout $i = 1, 2, \ldots, n$, R_i est une composante connexe.

tel que: $P(R_i$ est un prédicat d'uniformité pour touts les éléments de l'ensemble R_i et $\emptyset$ représente l'ensemble vide.

Une grande variété de méthodes de segmentation a été développée au cours de ces dernières décennies et ce nombre ne cesse de croître chaque année. Ces méthodes peuvent être classées, essentiellement, en deux catégories fondamentales la discontinuité et la similarité (Bovik, 2009; Gonzalez and Woods, 2006). La première regroupe les méthodes de segmentation basées sur l'approche *contour* (Frontière), tandis que la deuxième regroupe les méthodes de segmentation basées sur l'approche *région* (voir figure 3.8).

3.3.1 Approches contour

L'approche contour consiste à transformer l'image originale en une image contour en identifiant les changements entre les régions. En général, un contour est défini par une variation brusque dans l'intensité lumineuse de l'image et qui se produit sur les frontières entre deux régions (Thakare, 2011; Muthukrishnan and Radha, 2011).

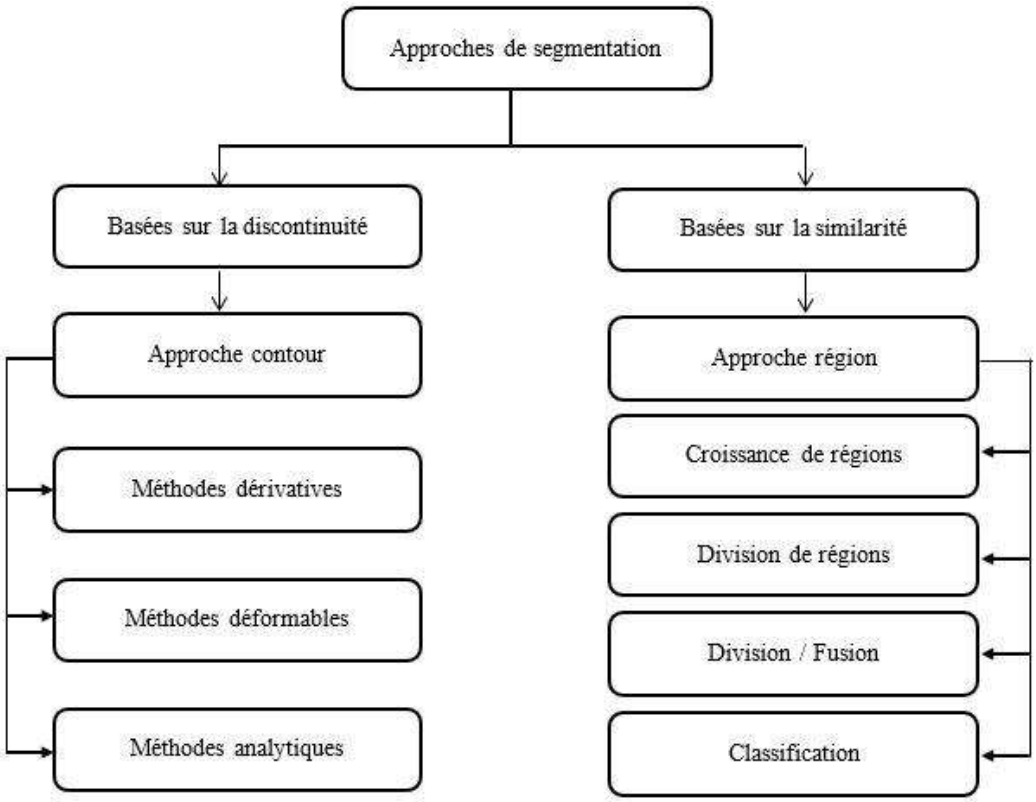


FIGURE 3.8: Téchniques de segmentation d'image

Dans l'approche de segmentation basée contour Il existe plusieurs méthodes qui sont regroupées en trois catégories : les méthodes *dérivatives*, les méthodes *Analytiques* et les méthodes *déformables*.

- ***Les méthodes dérivatives*** consistent à calculer la dérivée en chaque point de l'image afin de détecter et d'identifier les variations d'intensité. Ces méthodes peuvent être classées en deux groupes : approche *Gradient* basée sur la dérivée première et l'approche *Laplacien* basée sur la dérivée seconde.

  Il existe plusieurs opérateurs de gradient, les plus connus sont les masques de Robert (Roberts, 1963), de Prewit (Prewitt, 1970) et de Sobel (Sobel, 1978). Ces techniques sont rapides et donnent de bons résultats, seulement leur efficacité diminue face aux images bruitées (discontinuité des contours) d'où l'apparition des filtres optimaux (méthodes analytiques).

- ***Les méthodes analytiques*** se basent sur l'utilisation des filtres optimaux afin d'améliorer la précision et la localisation des contours. Parmi ces filtres, nous citons le filtre de *Canny* (Canny, 1986) et de *Deriche* (Deriche, 1987).

- ***Les méthodes déformables*** connues aussi sous le nom « Sneakes » ou « Contours actifs » visent à extraire les contours ou les surfaces fermées. Ces méthodes se basent sur l'utilisation d'une courbe paramétrique déformable successivement jusqu'à ce qu'elle coïncide avec la forme de l'objet à détecter. L'évolution de cette courbe menée par : une force interne liée à la surface elle-même visant à garder cette dernière aussi lisse que possible et une force externe qui dirige la courbe vers les contours souhaités (Kass, Witkin, and Terzopoulos, 1988; McInerney and Terzopoulos, 2000; Sethian, 1996; Cremers, Rousson, and Deriche, 2007; Sum and Cheung, 2007).

3.3.2 Approches région

L'approche région se base sur la continuité, elle consiste à diviser l'image entière en sous-régions ou en groupes en fonction d'un critère d'homogénéité (par exp : regrouper les pixels ayant le même niveau de gris en une seule région) (Gonzalez and Woods, 2006; Jain, Duin, and Mao, 2000). Les méthodes de segmentation, basées sur l'approche région, peuvent être divisées en quatre classes : croissance de régions, division de régions, division/fusion et classsificaiton.

Segmentaiton par croissance de régions (Region groiwing)

La segmentation par croissance de régions est une technique ascendante basée sur la notion de *germe* (seed), qui représente un pixel ou une région. Cela dit, les régions sont formées par agrégation des pixels voisins satisfaisant un critère d'homogénéité choisi. La croissance de régions s'arrête lorsque chaque pixel de l'image est attribué à un segment (Gao et al., 2011; Tilton et al., 2012).

Toutefois, le choix des germes initiaux et du critère d'homogénéité est critique, car la qualité des segments obtenus dépend de l'emplacement et de l'ordre de ces germes. Dans ce contexte, (Wan and Higgins, 2003) ont proposé des méthodes de croissance de régions symétriques (symmetric region growing), tandis que, (Chang and Li, 1994) ont développé une méthode de croissance de régions adaptatif.

Segmentation par division de régions (Split)

La division de régions est une approche de segmentation descendante qui considère l'image entière comme région initiale. Cette dernière est divisée en sous régions de manière récursive tant qu'un critère d'homogénéité n'est pas satisfait. Ce processus

est répété pour chaque région nouvellement crée et ne s'arrête que lorsque toutes les régions soient homogènes ou bien leur taille est en dessous d'un seuil préalablement fixé (Ohlander, Price, and Reddy, 1978).

Comme la division de l'image est un processus récursif, elle est généralement représentée par une structure géométrique particulière tel que l'arbre quartenaire (*Quadtree*) (Goshtasby and Nikolov, 2007).

Néanmoins, l'inconvénient de cette approche est la sur-segmentation (l'image finale peut contenir des régions ayant des propriétés identiques). Cette dérnière peut être résolue en utilisant la méthode de division-fusion (*Split and Merge*).

Segmentation par division-fusion (Split and Merge)

La segmentation par division-fusion est une méthode hybride proposée par Horowitz et Pavlidis (Horowitz and Pavlidis, 1976). Cette téchnique se déroule en deux partie :

- la *division* de l'image en de petites régions homogènes en utilisant la technique de segmentation par division décrite précedemment.
- La *fusion* des régions adjacentes et similaires. Dans ce cas, la similarité peut être mesurée en comparant le niveau de gris moyen ou en utilisant des tests statistiques. Ce processus est répété suivant une structure pyramidale jusqu'à ce qu'aucune autre fusion ne soit possible.

Segmentation par classification

La classification est le processus qui permet de partitionner un ensemble de données multidimensionnelles en un ensemble de k classes disjointes (*cluster*). En segmentation d'image, elle permet d'identifier les pixels ayant des attributs similaires afin de les regrouper dans une même classe (les pixels de deux classes distinctes ont des attributs très différents) (Jain, Murty, and Flynn, 1999). Parmi les méthodes proposées dans la littérature, nous citons : l'algorihtme de *K-means* et sa version floue *Fuzzy C-means* et l'algorithme *Mean-Shift*.

3.3.3 Segmentation par seuillage

Le seuillage (Thresholding) est une méthode largement utilisée dans la segmentation d'image. Il est facile à utiliser et utile pour séparer les objets de l'arrière-plan, ou pour

discriminer les objets qui ont des niveaux de gris distincts (Sahoo, Soltani, and Wong, 1988).

Les techniques de seuil peuvent être divisées en deux catégories selon le nombre de seuils requis pour l'image : le seuillage à deux niveaux et le multi-seuillage (a plusieurs niveaux). Dans l'approche à deux niveaux, les pixels qui ont une valeur d'intensité supérieure au seuil *(th)* sont étiquetés comme un objet, et les autres pixels font partie du fond. Pour une sélection appropriée des classes dans une image, la valeur de seuil optimale *(th)* doit être sélectionnée de la manière suivante :

$$\begin{aligned} C_1 &\leftarrow p \quad \text{si} \quad 0 \leq p < th \\ C_2 &\leftarrow p \quad \text{si} \quad th \leq p < L-1 \end{aligned} \tag{3.2}$$

où, p indique l'un des mxn pixels qui appartient à l'image en niveaux de gris I, qui peut être noté en L nuances de gris $L = 0, 1, 2, \ldots, L-1$. C_1 et C_2 représentent les deux classes différentes, qui sont séparées par le pixel p, alors que th indique le seuil.

Le concept de seuillage à deux niveaux peut être appliqué au multi-seuillage de la manière suivante :

$$\begin{aligned} C_1 &\leftarrow p \quad \text{si} \quad 0 \leq p < th_1 \\ C_2 &\leftarrow p \quad \text{si} \quad th_1 \leq p < th_2 \\ C_i &\leftarrow p \quad \text{si} \quad th_{i-1} \leq p < th_i \\ C_n &\leftarrow p \quad \text{si} \quad th_n \leq p < L-1 \end{aligned} \tag{3.3}$$

où, $\{th_1, th_2, \ldots, th_i, th_{i+1}, th_k\}$ représentent les différents seuils.

Les seuils (th) peuvent être déterminés de manière locale ou globale. En seuillage local, un seuil différent est attribué à chaque partie de l'image. Tandis qu'en seuillage global, une seule valeur de seuil globale est fixée pour l'ensemble de l'image. Cependant, la sélection de ces seuils peut être effectuée soit par des approches paramétriques ou non paramétriques (Ballard and Brown, 1982; Chehdi and Coquin, 1991; De Albuquerque, Esquef, and Mello, 2004).

Les méthodes paramétriques supposent que l'histogramme peut être approximé en utilisant, par exemple, une combinaison linéaire de fonctions de densité de probabilité (PDF) (Snyder et al., 1990) dont le médèle est connu à priori. Autrement dit, les

paramètres statistiques des classes dans l'image sont estimés (Cheriet, Said, and Suen, 1998; Reddi, Rudin, and Keshavan, 1984). Alors que les méthodes non paramétriques sont basées sur l'optimisation d'une fonction objective pour déterminer les valeurs des seuils optimaux (Kittler and Illingworth, 1986), telles que la variance entre les classes (Otsu, 1979) ou les mesures d'entropie. Nous présentons dans ce qui suit quelques une de ces méthodes.

La méthode d'Otsu

En 1975, Otsu (Otsu, 1979) a proposé une technique non paramétrique pour segmenter l'image en différentes classes de manière à maximiser la variance des différentes classes. Otsu décrit trois critères discriminants possibles qui sont la variance inter classes σ_B^2, intra classes σ_W^2, et la variance totale σ_T^2.

En utilisant la notation d'Otsu, une image peut être composée de L niveaux de gris. Le nombre de pixels avec un niveau de gris donné i est noté par n_i, et le nombre total de pixels de l'image est donné par $N = n_0 + n_2 + n_3 + \ldots + n_{L-1}$. La probabilité d'un pixel ayant un certain niveau de gris est calculé comme suit :

$$p_i = \frac{n_i}{N} \text{ Où } : p_i \geq 0, \sum_{i=1}^{L} p_i = 1 \tag{3.4}$$

Le but de cette méthode est de trouver le seuil qui maximise l'un de ces rapports :

$$\eta = \sigma_B^2/\sigma_T^2, \lambda = \sigma_B^2/\sigma_W^2, K = \sigma_T^2/\sigma_W^2 \tag{3.5}$$

La plupart des chercheurs choisissent à maximiser le rapport η tel que :

$$\sigma_B^2 = \omega_0 \left(\mu_0 - \mu_T\right)^2 + \omega_1 \left(\mu_1 - \mu_T\right)^2 \tag{3.6}$$

Où en utilisant sa formule simplifier suivante :

$$\sigma_B^2 = \omega_0 \omega_1 \left(\mu_0 - \mu_1\right)^2 \tag{3.7}$$

La variance totale σ_T^2 est calculée par :

$$\sigma_T^2 = \sum_{i=0}^{L-1} \left(i - \mu_T\right)^2 p_i \tag{3.8}$$

Etant donné que la variance totale est constante pour l'histogramme de l'image, il suffit donc juste de maximiser la variance entre les classes (σ_B^2). Cependant, Otsu a défini la variance entre les classes (interclasse) comme la somme des fonctions sigma de chaque classe de sorte que :

$$\sigma_0 = \omega_0 \left(\mu_0 - \mu_T\right)^2, \quad \sigma_1 = \omega_1 \left(\mu_1 - \mu_T\right)^2 \tag{3.9}$$

$$\omega_0 = \sum_{i=0}^{th-1} p_i, \quad \omega_1 = \sum_{i=th}^{L-1} p_i = 1 - \omega_0 \tag{3.10}$$

Où μ_T est l'intensité moyenne de l'image originale. Dans le seuillage à deux niveaux le niveau moyen de chaque classe peut être calculé par :

$$\mu_0 = \sum_{i=0}^{th-1} \frac{ip_i}{\omega_0}, \quad \mu_1 = \sum_{i=th}^{L-1} \frac{ip_i}{\omega_1} \tag{3.11}$$

Le seuil optimal th^* peut être déterminé en maximisant la fonction de variance entre les classes par l'équation suivante :

$$th^* = \arg\max(\sigma_0 + \sigma_1) \tag{3.12}$$

Le seuillage à deux niveaux basé sur la variance entre classes peut être étendu au seuillage à plusieurs niveaux comme suit :

$$\begin{aligned} \sigma_0 &= \omega_0 \left(\mu_0 - \mu_T\right)^2 \\ \sigma_1 &= \omega_1 \left(\mu_1 - \mu_T\right)^2 \\ \sigma_2 &= \omega_2 \left(\mu_2 - \mu_T\right)^2 \\ \sigma_k &= \omega_k \left(\mu_k - \mu_T\right)^2 \\ \sigma_m &= \omega_m \left(\mu_m - \mu_T\right)^2 \end{aligned} \tag{3.13}$$

$$\mu_0 = \sum_{i=0}^{th_1-1} \frac{ip_i}{\omega_i}, \quad \mu_1 = \sum_{i=th_1}^{th_2-1} \frac{ip_i}{\omega_i}, \quad \mu_2 = \sum_{i=th_2}^{th_3-1} \frac{ip_i}{\omega_i}, \quad \mu_j = \sum_{i=th_j}^{t_{i+1}-1} \frac{ip_i}{\omega_i} \quad \mu_m = \sum_{i=th_m}^{L-1} \frac{ip_i}{\omega_i} \tag{3.14}$$

Les seuils optimaux sont déterminés en maximisant la fonction de variance entre les classes suivante :

$$th^* = \{th_1^*, th_2^*, \cdots, th_k^*, \cdots, th_{m-1}^*\} = \underset{0 \leq th_k \leq (L-1)}{\operatorname{argmax}} \sum_{i=0}^{m} \sigma_i \tag{3.15}$$

La méthode de Kapur

La méthode de la somme d'entropie proposé par Kapur (Kapur, Sahoo, and Wong, 1985) est une technique de segmentation efficace basée sur la distribution de probabilité de l'histogramme des niveaux de gris. L'entropie est maximale lorsque les seuils optimaux séparant les classes sont correctement attribués. Par conséquent, le but est de trouver les seuils optimaux qui produisent l'entropie maximale. Cependant, L'entropie d'une source discrète est souvent obtenue à partir de la distribution de probabilité $p = p_i$, où p_i est la probabilité du système dans l'état possible i (Bhandari et al., 2016) La probabilité de chaque niveau de gris i est la fréquence d'occurrence relative à ce niveau de gris, normalisée par le nombre total de niveaux de gris.

L'entropie de Kapur mesure la densité et la séparabilité des différentes classes. Pour le seuillage à deux niveaux, l'entropie de Kapur peut être décrite par l'équation suivante :

$$\begin{aligned} H_0 &= -\sum_{i=0}^{th-1} \frac{p_i}{\omega_0} \ln \frac{p_i}{\omega_0}, \quad \omega_0 = \sum_{i=0}^{th-1} p_i \\ H_1 &= -\sum_{i=th}^{L-1} \frac{p_i}{\omega_1} \ln \frac{p_i}{\omega_1}, \quad \omega_1 = \sum_{i=th}^{L-1} p_i \end{aligned} \tag{3.16}$$

Le seuil est optimal (th^*) lorsque la somme des entropies de classe est maximale, équation (3.17).

$$th^* = \arg\max(H_0 + H_1) \tag{3.17}$$

L'entropie de Kapur peut être étendue au multi-seuillage (Pare et al., 2016) comme suit :

$$
\begin{aligned}
H_0 &= -\sum_{i=0}^{th_1-1} \frac{p_i}{\omega_0} \ln \frac{p_i}{\omega_0}, \quad & \omega_0 &= \sum_{i=0}^{th_1-1} p_i \\
H_1 &= -\sum_{i=th_1}^{th_2-1} \frac{p_i}{\omega_1} \ln \frac{p_i}{\omega_1}, \quad & \omega_1 &= \sum_{i=th_1}^{th_2-1} p_i \\
H_2 &= -\sum_{i=th_2}^{th_3-1} \frac{p_i}{\omega_2} \ln \frac{p_i}{\omega_2}, \quad & \omega_2 &= \sum_{i=th_2}^{th_3-1} p_i \\
H_k &= -\sum_{i=th_k}^{th_{k+1}-1} \frac{p_i}{\omega_j} \ln \frac{p_i}{\omega_j}, \quad & \omega_k &= \sum_{i=th_j}^{L_{k+1}^{L-1}} p_i \\
H_m &= -\sum_{i=th_m}^{L-1} \frac{p_i}{\omega_m} \ln \frac{p_i}{\omega_m}, \quad & \omega_m &= \sum_{i=th_m}^{L-1} p_i
\end{aligned}
\tag{3.18}
$$

Les seuils optimaux sont obtenus en utilisant l'équation (3.19).

$$
th^* = \{th_0^*, th_1^*, \cdots, th_k^*, \cdots, th_{m-1}^*\} = \underset{0 \le th_k \le (L-1)}{\operatorname{argmax}} \sum_{i=0}^{m} H_i \tag{3.19}
$$

La méthode d'entropie de Tsallis

Tsai (Tsai, 1985) a développé le concept d'entropie non extensive de Tsallis, qui est similaire à la méthode de la somme d'entropie maximale de Kapur vu précédemment. L'entropie de Tsallis a été étudiée pour une extension possible de l'entropie de Shannon (la théorie de l'information). Elle est définie comme suit :

$$
S_q = \frac{1 - \sum_{i=1}^{k} (p_i)^q}{q-1} \tag{3.20}
$$

où k indique le nombre total de possibilités du système et q désigne la mesure du degré de non-extensivité du système, connue sous le nom d'indice entropique ou de paramètre de Tsallis.

L'utilisation de cette méthode consiste à maximiser l'entropie de Tsallis des classes (*fond*) et (*objet*) comme suit :

$$
S_1^q(th) = \frac{1 - \sum_{i=1}^{th-1} \left(\frac{p_i}{P_1}\right)^q}{q-1} \text{ et } S_2^q(th) = \frac{1 - \sum_{i=th}^{L-1} \left(\frac{p_i}{P_2}\right)^q}{q-1} \tag{3.21}
$$

L'entropie totale de Tsallis est décrite comme suit :

$$
S^q(th) = \left[S_1^q + S_2^q + (1-q) S_1^q(t) S_2^q(t)\right] \tag{3.22}
$$

Le seuil optimal (th^*) est déteminé par l'équation suivante :

$$th^* = \arg \max_{0 \leq th \leq L-1} \{S^q(th)\} \tag{3.23}$$

L'utilisation de cette méthode dans le cas du multi-seuillage, consiste à détérmnier l'ensemble des seuils qui maximise la fonction donnée par l'équaiton (3.25)

$$\begin{aligned} S_1^q(th) &= \frac{1 - \sum_{i=1}^{th_1-1} \left(\frac{p_i}{P_1}\right)^q}{q-1}, P_1 = \sum_{i=0}^{th_1-1} p_i \\ S_2^q(th) &= \frac{1 - \sum_{i=th_1}^{th_2-1} \left(\frac{p_i}{P_2}\right)^q}{q-1}, P_2 = \sum_{i=th_1}^{th_2-1} p_i \\ S_m^q(th) &= \frac{1 - \sum_{i=th_m}^{l-1} \left(\frac{p_i}{P_m}\right)^q}{q-1}, P_m = \sum_{i=th_m}^{L-1} p_i \end{aligned} \tag{3.24}$$

$$th^* = \{th_1^*, th_2^*, \cdots, th_k^*, \cdots, th_{m-1}^*\} = \underset{0 \leq th_k \leq (L-1)}{\operatorname{argmax}} (S_1^q + S_2^q + \ldots + S_{L-1}^q + (1-q) S_1^q S_2^q \ldots S_{L-1}^q) \tag{3.25}$$

La méthode de corrélation entropique

La méthode de corrélation entropique proposée par Yen (Yen, Chang, and Chang, 1995) consiste à maximiser la corrélation entropique entre les classes C_1 (*objet*) et C_2 (*fond*). La corrélation de ces deux classes est calculée par la formule suivante :

$$Cor_1(t) = -\ln \sum_{i=0}^{th} \left(\frac{p_i}{P_1}\right)^2 \text{ et } Cor_2(t) = -\ln \sum_{i=t+1}^{l-1} \left(\frac{p_i}{P_2}\right)^2 \tag{3.26}$$

La corrélation totale associée aux deux classes est alors claculée par :

$$J(th) = Cor_1(th) + Cor_2(th) = -\ln \sum_{i=0}^{th} \left(\frac{p_i}{P_1}\right)^2 - \ln \sum_{i=th+1}^{l-1} \left(\frac{p_i}{P_2}\right)^2 \tag{3.27}$$

Qui peut être simplifiée par :

$$J(th) = -\ln \left(\frac{G_1(th) G_2(th)}{P_1^2 P_2^2}\right) = -\ln \left(G_1(th) G_2(th)\right) + 2 \ln \left(P_1 P_2\right) \tag{3.28}$$

Tel que :

$$G_1(th) = \sum_{i=0}^{t} p_i^2, G_2(th) = \sum_{i=t+1}^{l-1} p_i^2, P_1 = \sum_{i=0}^{th} p_i \text{ et } P_2 = \sum_{i=th+1}^{l-1} p_i \tag{3.29}$$

Le seuil optimal th^* est obtenu en maximisant la fonciton $J(th)$:

$$th^* = \arg \max_{0 \leq t \leq L-1} J(th) \tag{3.30}$$

La méthode de Kittler

Dans cette méthode (Kittler and Illingworth, 1986), l'histogramme est considéré comme une estimation de la fonction de densité de probabilité d'un mélange de population formées des niveaux de gris de l'objet et de l'arrière-plan. Chacun de ces derniers est distribué avec une moyenne μ_i , un écart type σ_i et une probabilité P_i. Le seuil optimal th^* est calculé par la minimisation de la fonction objective suivante:

$$J\left(th^*\right) = \arg\min_{th \in 0..L-1} J(th) \tag{3.31}$$

Tel que:

$$J(th) = 1 + 2\left[P_0(th)\log\sigma_0(th) + P_1\log\sigma_1(th)\right] - 2\left[P_0(t)\log P_0(t) + P_1(th)\log P_1(th)\right] \tag{3.32}$$

$$P_1(th) = \sum_{i=0}^{th} p(i), \quad \text{et} \ : P_2(th) = 1 - P_1 = \sum_{i=th+1}^{L-1} p(i), \tag{3.33}$$

$$\mu_1(th) = \frac{1}{P_1}\sum_{i=0}^{th} i * p(i), \quad \text{et: } \mu_2(th) = \frac{1}{P_2}\sum_{i=th+1}^{L-1} i * p(i) \tag{3.34}$$

$$\sigma_1(th) = \frac{1}{P_1(th)}\sum_{i=0}^{\infty}\left[i - \mu_1(th)\right]^2 * p(i), \text{ et: } \sigma_2(th) = \frac{1}{P_2(th)}\sum_{i=th+1}^{\infty-1}\left[i - \mu_2(th)\right]^2 * p(i) \tag{3.35}$$

La méthode de Ridler

Ridler et Calvard (Ridler, Calvard, et al., 1978) ont développé l'algorithme Isodata qui est une technique simple et itérative. Initialement, une estimation est faite sur une

valeur possible pour le seuil. Après, les moyennes des valeurs de pixels dans les deux classes (*objet* et *fond*) produites à l'aide de ce seuil sont calculées, équation (3.36). Le seuil est repositionné au centre des deux moyennes :

$$th_{\text{Ridler}} = \frac{1}{2}(\mu_1 + \mu_2) \tag{3.36}$$

Les valeurs moyennes μ_1 et μ_2 sont recalculées à nouveau, un nouveau seuil est alors obtenu, ce processus est répété jusqu'à ce que la valeur du seuil devient stable.

3.4 Amélioration des méthodes de seuillage par les métaheuristiques

Bien que, les méthodes de seuillage abordées précédemment ont montré leur efficacité dans la segmentation d'image; elles deviennent très couteuses en termes de temps de calcul avec l'augmentation du nombre de seuils (la recherche exhaustive). Pour surmonter ce problème de complexité et afin d'obtenir des valeurs de seuils optimaux, de nombreuses techniques d'optimisation méta-heuristiques ont été appliquées au fil des années aux méthodes de seuillage classiques susmentionnées.

Parmi les travaux présentés dans la littérature nous citons : (Hammouche, Diaf, and Siarry, 2008; Tang et al., 2011; Zhang et al., 2014; Pruthi and Gupta, 2016; Abd Elaziz, Ewees, and Oliva, 2020) ont présenté des travaux basés sur l'utilisation des algorithmes génétiques (GA) et sa combinaison afin de surmonter les inconvénients liés à la segmentation. De même, PSO est appliqué pour résoudre le problème du multi-seuillage pour la segmentation (Yin, 2007; Maitra and Chatterjee, 2008; Gao et al., 2009; Ghamisi et al., 2012; Gao, Pun, and Kwong, 2016; Naik, Sadhu, and Gopal, 2016). Il y'a aussi l'utilisation de l'algorithme Firefly par (Erdmann et al., 2015), tandis que, (Chen et al., 2016) ont proposé l'algorithme Improved Firefly Algorithm afin d'améliorer les performances de FA pour résoudre le problème de multi-seuillage. D'autre part, (Fayad, Hatt, and Visvikis, 2015) ont proposé un modèle de segmentation basé sur l'algorithme de colonies de fourmis (ACO), ils ont obtenu de bons résultats et de faibles erreurs par rapport à la situation réelle. L'algorithme d'optimisation de la colonie d'abeilles artificielle (ABC) est utilisé par (Cuevas et al., 2012; Bhandari, Kumar, and Singh, 2015) pour calculer les seuils.

En outre, il existe de nombreux autres algorithmes métaheuristiques, qui ont été utilisés pour la segmentation d'images, notamment : CS (Agrawal et al., 2013), BFO (Sathya and Kayalvizhi, 2011a; Sathya and Kayalvizhi, 2011b; Bakhshali and Shamsi, 2014), DE (Cuevas, Zaldivar, and Pérez-Cisneros, 2010; Ali, Siarry, and Pant, 2012), CSA (Oliva and Cuevas, 2017), WOA (Abd El Aziz, Ewees, and Hassanien, 2018) et FFO (Chai, 2021).

3.5 Conclusion

Nous avons, au cours de ce chapitre, abordé quelques notions sur le traitement d'image en se focalisant sur le domaine de la segmentation. Ce dernier est tellement vaste et diversifié qu'il est difficile de présenter une étude exhaustive de ses approches. Nous nous sommes particulièrement intéressés à l'approche de segmentation par seuillage. Les méthodes de seuillage sont nombreuses, nous avons détaillé quelques-unes d'entre elles.

Par ailleurs, ces méthodes se caractérisent par la complexité du temps de calcul lors de la recherche exhaustive, ce qui fait d'elles un problème d'optimisation complexe. La résolution d'un tel problème fait appel au domaine de l'optimisation mono-objective basé sur les métaheuristiques, quelques travaux dans ce contexte ont été également cités dans ce chapitre.

Chapitre 4

L'application des méta-heuristiques à la quantification vectorielle pour la compression d'image

4.1 Introduction

En raison du développement de la technologie numérique, les besoins en espace de stockage et en temps de transmission des données numériques augmentent à une vitesse incroyable, mais la bande passante de transmission, quant à elle, reste limitée. Afin de remédier à ce problème et atteindre une exploitation optimale, une technique de compression de données multimédia (image, audio et vidéo) est la solution la plus adaptée. Les techniques de compression visent à réduire le nombre de bits nécessaires pour représenter un signal donné, tout en maintenant une fidélité ou une qualité satisfaisante. Par conséquent, plusieurs méthodes de compression ont été proposées et classées en deux catégories, la compression sans perte et la compression avec perte. La première est une compression non destructive qui permet de réduire la taille des données sans perte d'information. Alors que la seconde catégorie, code l'image avec une dégradation de la qualité, puisque cette technique entraîne une certaine perte de données permettant d'atteindre un taux de compression plus élevé que la compression sans perte.

La quantification vectorielle (QV) est l'une des techniques les plus efficaces utilisées dans la compression d'images avec perte en raison de son architecture simple et de son taux de compression élevé avec une distorsion minime. Les trois principales étapes de la QV sont le processus de codage, le processus de décodage et la génération du dictionnaire. Cependant, la qualité de la QV est fortement conditionnée par l'algorithme de construction de ce dictionnaire. La technique *LBG* est largement utilisée pour la

conception du dictionnaire. Néanmoins, cette technique est très sensible au dictionnaire initial utilisé et a tendance à se piéger au minimum local. Par conséquent, de nombreux travaux de recherche basés sur l'optimisation ont été développés pour la génération d'un livre de codes global.

Dans notre travail, nous avons proposé l'uitlisation d'un nouvel algorithme méta-heuristique d'optimisation des baleines, notamment, **WOA** pour la conception de dictionnaire optimal dans le but d'améliorer la qualité de la quantification vectorielle. L'approche proposée a été comparée avec les quatre algorithmes les plus utilisés, LBG-Rand, LBG-Split, PSO et FA (Boubechal, Seghir, and Benzid, 2018).

4.2 Quantification vectorielle pour la compression des images

La QV dans son sens le plus général est l'approximation d'un signal de valeurs continues par un signal de valeurs discrètes. Elle est l'une des méthodes les plus utilisées en compression d'image (Gersho and Gray, 2012). Le processus de la QV illustré à la figure 4.1, peut être divisé en trois phases principales : la génération du dictionnaire *(livre de codes)*, le codage et décodage de l'image.

L'objectif de la 1^{er} phase est de générer un ensemble de mots de code représentatifs pour former le dictionnaire. Tandis que dans la 2^{me} phase, pour tout vecteur $\mathbf{x}$ du signal source en entrée *(bloc de l'image de taille 4x4 ou 8x8)*, il s'agit de trouver le code-vecteur c du dictionnaire $\mathbf{C}$ le plus ***proche*** du vecteur source $\mathbf{x}$. La notion de similarité utilisée ici est la distance euclidienne. Ainsi, seule l'adresse du code-vecteur $\mathbf{c}$ sélectionné sera stockée/transmise, c'est donc à ce niveau là que s'effectue la compression. Cependant, lors du décodage, le même dictionnaire est consulté pour fournir le code-vecteur de l'adresse reçue afin de reconstruire l'image compressée (Karayiannis and Liu, 2000; Linde, Buzo, and Gray, 1980).

4.2.1 Formulation mathématique de la quantification véctorielle

La ***performance*** de la QV est fortement conditionnée par le *dictionnaire* utilisé. Donc, la conception du dictionnaire *(codebook)* est la partie la plus essentielle de la QV, et elle peut être décrite mathématiquement comme suit (Linde, Buzo, and Gray, 1980; Hang, Woods, et al., 1995; Fonseca, Ferreira, and Madeiro, 2018) :

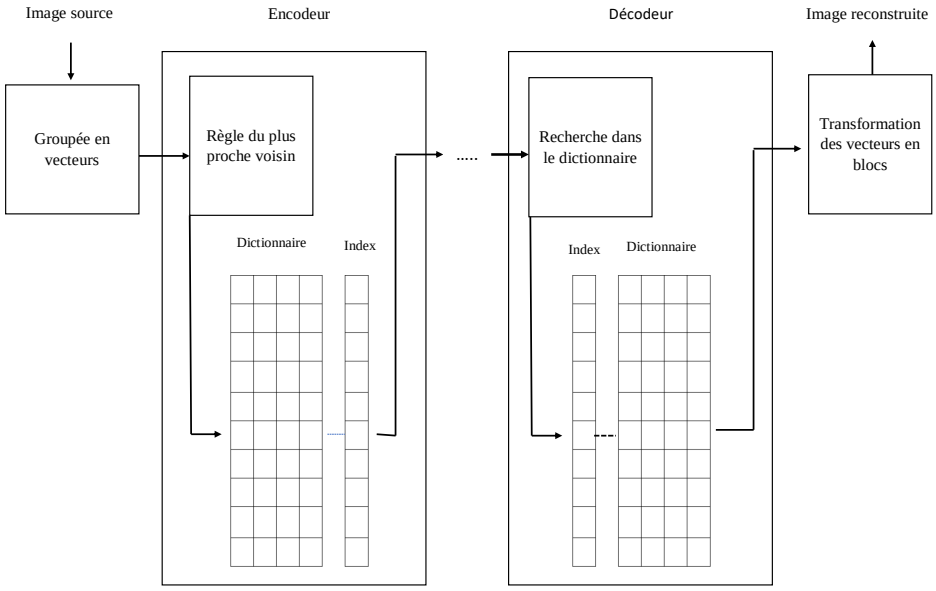


FIGURE 4.1: Schéma général d'un quantificateur vectoriel

Soit une image source de taille IxI pixels divisée en plusieurs blocs de taille nxn, $M = (\frac{I}{n} \times \frac{I}{n})$. Ces M blocs représentent les vecteurs sources *(l'ensemble d'apprentissage)*, notés $X := \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{M-1}\}$.
Nous supposons que les vecteurs sources sont de dimension k.

$$\mathbf{x}_m = (x_{m,1}, x_{m,2}, \ldots, x_{m,k}), \quad m = 0, 1, \ldots, M-1 \tag{4.1}$$

Soit N le nombre de code-vecteurs du dictionnaire C, tel que :

$$C := \{\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_{N-1}\} \tag{4.2}$$

Chaque code-vecteur est aussi de dimension k.

$$\mathbf{c}_n = (c_{n,1}, c_{n,2}, \ldots, c_{n,k}), \quad n = 0, 1, \ldots, N-1 \tag{4.3}$$

Soit r_n la région de codage associée au code-vecteur c_n.

$$R := \{r_0, r_1, \ldots, r_{N-1}\} \tag{4.4}$$

Si le vecteur source x_m se trouve dans la région de codage r_n, alors son approximation (désignée par $Q(\mathbf{x}_m)$) est c_n :

$$Q(\mathbf{x}_m) = \mathbf{c}_n \quad \text{if} \quad \mathbf{x}_m \in r_n \tag{4.5}$$

L'objectif de la QV est de trouver le dictionnaire C et l'ensemble de regions R qui minimise la distrotion D_{ave} donnée par:

$$D_{\text{ave}} := \frac{1}{Mk} \sum_{m=0}^{M-1} \|\mathbf{x}_m - Q(\mathbf{x}_m)\|^2 \tag{4.6}$$

Où ||...|| représente la distance euclidienne. Si C et R sont une solution au problème de minimisation ci-dessus, alors les deux critères suivants doivent être satisfaits :

- ***La condition du plus proche voisin*** :

$$r_n = \left\{ \mathrm{x} : \|\mathrm{x} - \mathrm{c}_n\|^2 \leq \|\mathbf{x} - \mathbf{c}_n\|^2 \text{ for all } n' = 0, 1, \ldots, N-1 \right\} \tag{4.7}$$

  Cette condition indique que la région de codage r_n contient tous les vecteurs qui sont plus proches de c_n .

- ***Condition du centroide :***

$$c_n = \frac{\sum_{x_n \in r_n} x_m}{\sum_{x_n \in r_n}}, \quad n = 0, 1, \ldots, N-1 \tag{4.8}$$

  Le code-vecteur c_n doit être la myenne de tous les vecteurs source appartenant à la région (classe) r_n.

Bien qu'il existe plusieurs façons de générer un dictionnaire, la plupart d'entre elles sont basées sur une approche particulière, connue sous le nom de **Lind-Buzo-Gray** (Linde, Buzo, and Gray, 1980; Nag, 2019; Bardekar and Tijare, 2011).

4.2.2 Conception d'un dictionnaire pour la QV par L'algorithme LBG

En 1980, Linde, Buzo et Gray (Linde, Buzo, and Gray, 1980) ont proposé une amélioration de la technique de Lloyd (Lloyd, 1982). Ils ont étendu les résultats de Lloyd des cas unidimensionnels aux cas multidimensionnels (k-dimensions). Pour cette raison, leur algorithme est connu sous le nom d'algorithme de Lloyd généralisé (GLA) ou bien ***LBG*** d'après les initiales de ses auteurs.

L'algorithme *LBG* est une séquence finie d'étapes dans laquelle, à chaque étape, un nouveau quantificateur (dictionnaire), dont la distorsion totale est inférieure ou égale à la précédente, est produit. Cet algorithme necessite un dictionnaire initail $C^{(0)}$ (Yanxia, Jiawei, and Ye, 2011), qui va être optimisé par la suite, en résolvant de manière alternative les deux critères d'optimalité vu précédemment, équations: (4.7) et (4.8).

Cependant, les *performances* de l'algorithme *LBG* dépendent fortement du choix du dictionnaire initial. Dans ce contexte, plusieurs techniques ont été proposées dans la littérature (Franti et al., 2000; Yanxia, Jiawei, and Ye, 2011; Bardekar and Tijare, 2011), celles qui feront l'objet de notre étude sont la méthode d'initialisation aléatoire et la méthode d'initialisation par dichotomie vectorielle *(spllinting)*.

- ***Initialisation aléatoire :*** La méthode aléatoire *(Random)* est la méthode la plus simple parmi les méthodes de génération de dictionnaire *(codebook)* initial. Tous les codes-vecteurs sont choisis au hasard dans l'ensemble d'apprentissage. Cette méthode garantit que, dans les étapes initiales, il y aura toujours au moins un vecteur de l'ensemble d'apprentissage dans chaque région de quantification. Cependant, nous pouvons toujours obtenir différents dictionnaires si nous utilisons différents sous-ensembles de l'ensemble d'apprentissage comme dictionnaire initial (Pal, Bezdek, and Tsao, 1993; Salomon and Motta, 2010).
- *Initialisation par dichotomie vectorielle (Splitting) :* Dans cette méthode, un code-vecteur initial est calculé par la moyenne de l'ensemble de l'apprentissage et est divisé en deux. L'itération se déroule avec ces deux vecteurs comme dictionnaire initial. Après l'itération, les deux derniers codes-vecteurs sont divisés en quatre et le processus d'itération se répète jusqu'à ce que le nombre souhaité de code-vecteurs dans le dictionnaire soit obtenu (Linde, Buzo, and Gray, 1980). L'algorithme de conception est présenté dans l'algorithme 11.

Inconvénient : l'algorithme *LBG* converge vers le minimum local le plus proche du dictionnaire initial. Comme alternative à ce problème, les *méta-heuristiques* ont été utilisées pour produire un *dictionnaire optimal global*.

4.3 Application des méta-heuristiques à la quantification vectorielle

Plusieurs techniques d'optimisation ont été proposées pour la génération d'un dictionnaire global pour la QV afin d'améliorer la qualité de la compression d'images. Parmi

Algorithm 11 Algorithme LBG (utilisant l'initialisation Splitting)

1: *Etant donné l'ensemble d'apprentissage T. Fixer $\epsilon > 0$ à une valeur minime*
2: *Mettre $N = 1$ (nombre des codes-vecteurs) et*

$$\mathbf{c}_1^* := \frac{1}{M} \sum_{m=0}^{M-1} \mathbf{x}_m \tag{4.9}$$

Calculer

$$D_{ave}^* = \frac{1}{Mk} \sum_{m=0}^{M-1} \|\mathbf{x}_m - \mathbf{c}_1^*\|^2 \tag{4.10}$$

3: *Splitting:* ***Pour*** *$i = 0, 1, \ldots, N-1$,* ***faire****:*

$$\begin{aligned} \mathbf{c}_i^{(0)} &:= (1+\epsilon)\mathbf{c}_i^* \\ \mathbf{c}_{N+i}^{(0)} &:= (1-\epsilon)\mathbf{c}_i^* \end{aligned} \tag{4.11}$$

Ainsi au début nous commençons avec deux codes-vecteurs. Mettre $N = 2N$
4: ***Itération :*** *j=0 et*

$$D_{ave}^{(0)} = D_{ave}^* \tag{4.12}$$

a) ***Pour*** *$m = 0, 1, \ldots, M-1$ trouver le minimum de :*

$$\left\| \mathbf{x}_m - \mathbf{c}_n^{(j)} \right\|^2 \tag{4.13}$$

Soit n^* l'indice qui fournit le minimum pour un m donné. Ainsi chaque m aura un certain n^*, tel que :

$$Q(\mathbf{x}_m) = \mathbf{c}_{n^*}^{(j)} \tag{4.14}$$

b) ***Pour*** *$n = 0, 1, \ldots, N-1$ mettre à jour les codes-vecteurs :*

$$\mathbf{c}_n^{(j+1)} = \frac{\sum_{Q(\mathbf{x}_m)=\mathbf{c}_n^{(j)}} \mathbf{x}_m}{\sum_{Q(\mathbf{x}_m)=c_n^{(j)}} 1} \tag{4.15}$$

c) ***j=j+1***
d) *Calculer :*

$$D_{ave}^{(j)} = \frac{1}{Mk} \sum_{m=0}^{M-1} \|\mathbf{x}_m - Q(\mathbf{x}_m)\|^2 \tag{4.16}$$

e) ***Si :***

$$\frac{D_{ave}^{(j-1)} - D_{ave}^{(j)}}{D_{ave}^{(j-1)}} > \epsilon \tag{4.17}$$

alors *retour à l'étape (a)*
f) *Mettre :*

$$D_{\text{ave}}^* = D_{\text{ave}}^{(j)} \tag{4.18}$$

Pour *$n = 0, 1, \ldots, N-1$ faire*

$$\mathbf{c}_n^* = \mathbf{c}_n^{(j)} \tag{4.19}$$

5: *Répéter les étapes **3** et **4** jusqu'à ce que le nombre de codes-vecteurs souhaité soit obtenu.*

ces travaux, nous citons : (Rajpoot et al., 2004) ont appliqué l'algorithme du système de colonies de fourmis *(ACS)* pour développer l'algorithme de conception du dictionnaire. La génération du dictionnaire à l'aide de l'ACS a été facilitée par la représentation des vecteurs de coefficients dans un graphe bidirectionnel. (Chen, Yang, and Gou, 2005) et (Liao et al., 2007) ont proposé une amélioration basée sur l'optimisation par essaims de particules (PSO). Le résultat de l'algorithme LBG a été utilisé pour initialiser la meilleure particule globale, ce qui permet d'accélérer la convergence de l'algorithme PSO. Le même principe a été introduit par (Horng, 2012) et (Karri and Jena, 2016) en utilisant Firefly algorithme et Bat algorithme, respectivement.

(Sanyal, Chatterjee, and Munshi, 2013) ont appliqué une nouvelle approche dans la sélection des étapes de chimiotaxie de l'algorithme d'optimisation **(BFOA)**, permettant ainsi à l'algorithme de développer un dictionnaire quasi-optimal pour la compression d'image avec une bonne qualité d'image reconstruite et un PSNR élevé. De plus, (Horng and Jiang, 2011) ont appliqué l'algorithme d'optimisation de l'accouplement des abeilles *HBMO* pour la QV. Leur approche permet de produire un meilleur dictionnaire avec une petite distorsion par rapport aux algorithmes PSO-LBG, QPSO-LBG et LBG.

4.4 L'approche proposée

L'algorithme d'optimisation des baleines (WOA) a été proposé par les auteurs Mirjalili et Lewis (Mirjalili and Lewis, 2016) en s'inspirant du comportement des baleines à bosse. Cet algorithme est discuté en détail à la section 2.3.9.

Notre objectif pour le problème de la QV est de générer un dictionnaire optimal, qui représente aux mieux l'image codée, sans trop altérer sa qualité. Donc, le but de l'algorithme WOA est de déterminer la position dans l'espace de recherche (les valeurs de dictionnaire optimal) qui optimise la fonction objective, dans notre cas minimiser la distortion donnée par l'Eq. (4.20).

$$MSE = \frac{\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}(x(i,j)-y(i,j))^2}{M \times N} \tag{4.20}$$

x et y représentent l'image originale et l'image segmentées de taille MxN, respectivement.

Chaque individu de la population des baleines est représenté comme un vecteur de valeurs (entre 0 et 255) correspondant à des dictionnaires. Cependant, dans notre travail, l'initialisation de cette population d'individus est faite de deux manières :

1. Tous les individus (dictionnaires candidats) sont générés de façon aléatoire, à partir de l'ensemble d'apprentissage (Figure 4.2).
2. Le résultat (dictionnaire) obtenu par l'algorithme LBG est pris comme individu dans la population à optimiser (Figure 4.3).

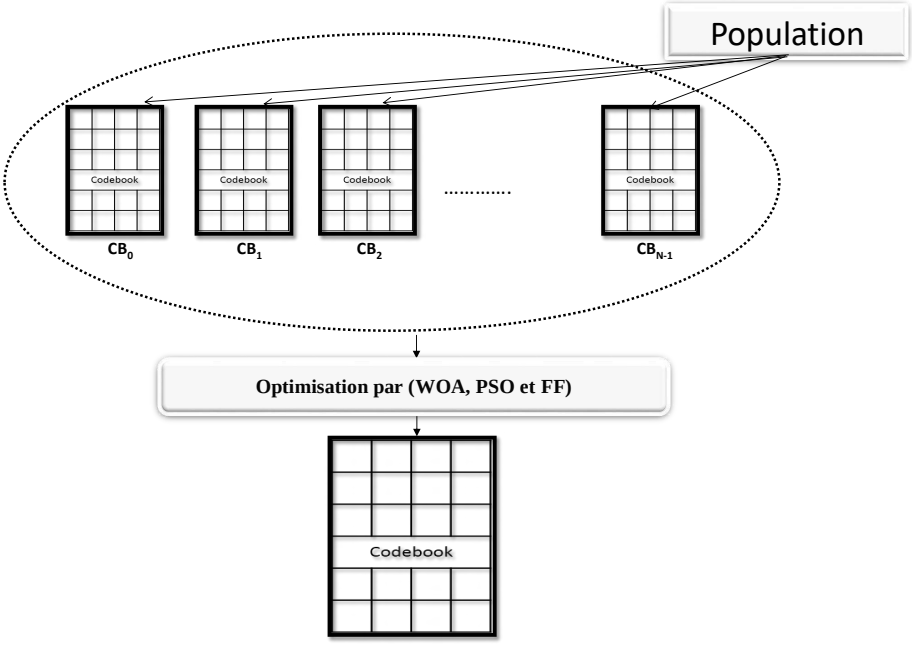


FIGURE 4.2: la résolution du problème de la QV avec la premiére méthode d'initialisation.

4.5 Résultats expérimentaux et discussion

L'étude expérimentale a été mise en œuvre sur cinq images en niveau de gris de tailles (512x512) pixels avec une résolution de 8 bits, nommées : « Lena », « Pepper », « Baboon » , « Barbar » et « GoldHill », représentées sur la Figure 4.4.

Cependant, pour évaluer la qualité du dictionnaire conçu par les différentes méthodes, nous avons mené l'expérimentation de deux manières, dont la différence réside dans le nombre d'image utilisées dans l'apprentissage de chaque algorithme. Cela dit,

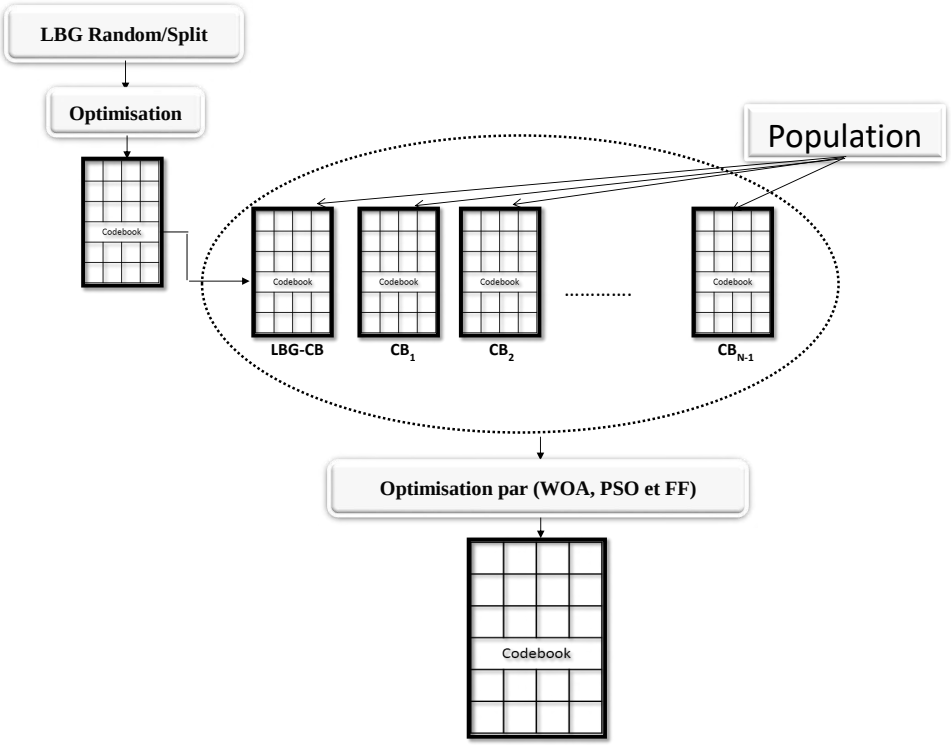
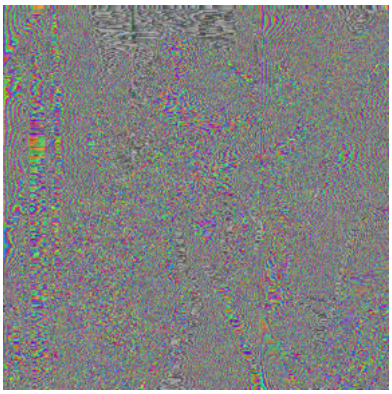


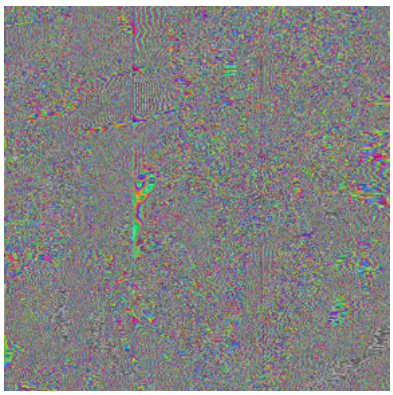
FIGURE 4.3: la résolution du problème de la QV en utilisant le résultat de l'algorithme LBG.

dans la première expérimentation nous avons utilisé quatre images comme ensemble d'apprentissage (Lena,Barbara,Baboon et Pepper). Ces dernières sont divisées en blocs de $4x4$ pixels, ce qui donne 65536 vecteurs d'entrée de dimension 16, présents dans l'ensemble d'apprentissage. L'évaluation du dictionnaire produit a été testé sur les images de la base d'apprentissage et sur une autre image ne faisant pas partie de cette base, notamment « ***Godhill*** ». Tandis que dans la deuxième expérimentation, l'ensemble d'apprentissage est composé des vecteurs d'une seule image (celle à encoder seulement). Ce qui donne 16384 vecteurs d'entrée de dimension 16.

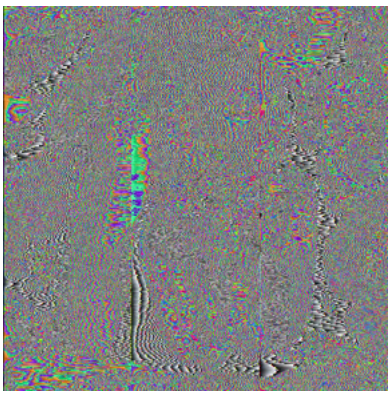
L'entrée de chaque algorithme utilisé est l'ensemble d'apprentissage et la sortie est la solution optimale x^* qui représente le dictionnaire optimal. L'implémentation des algorithmes utilisés est illustrée par les Figures 4.5, 4.6 et 4.7, dont les paramètres sont présentés dans les tableaux 4.1, 4.2 et 4.3, respectivement. Lors de l'expérimentation, les sept différents dictionnaires de taille $8, 16, 32, 64, 128, 256$ et 512 sont mis en œuvre. Nous avons comparé la méthode proposée avec quatre algorithmes différents, à savoir le LBG aléatoire traditionnel, LBG avec dichotomie vectorielle, les algorithmes d'optimisation PSO et Firefly.



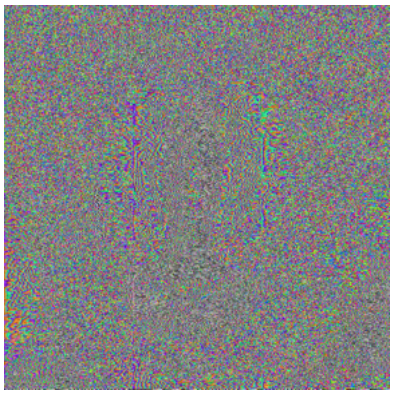
(a)



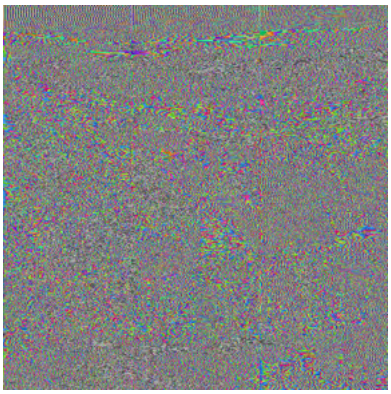
(b)



(c)



(d)



(e)

FIGURE 4.4: Les images tests: (a) Lena; (b) Barbara; (c) Pepper; (d) Baboon;(e) Goldhill

TABLE 4.1: Paramètres de l'algorithme WOA

Paramètres	Valeur
Taille de la Population	30
nombre d'itérations	100
Lower bound	0
Upper bound	255

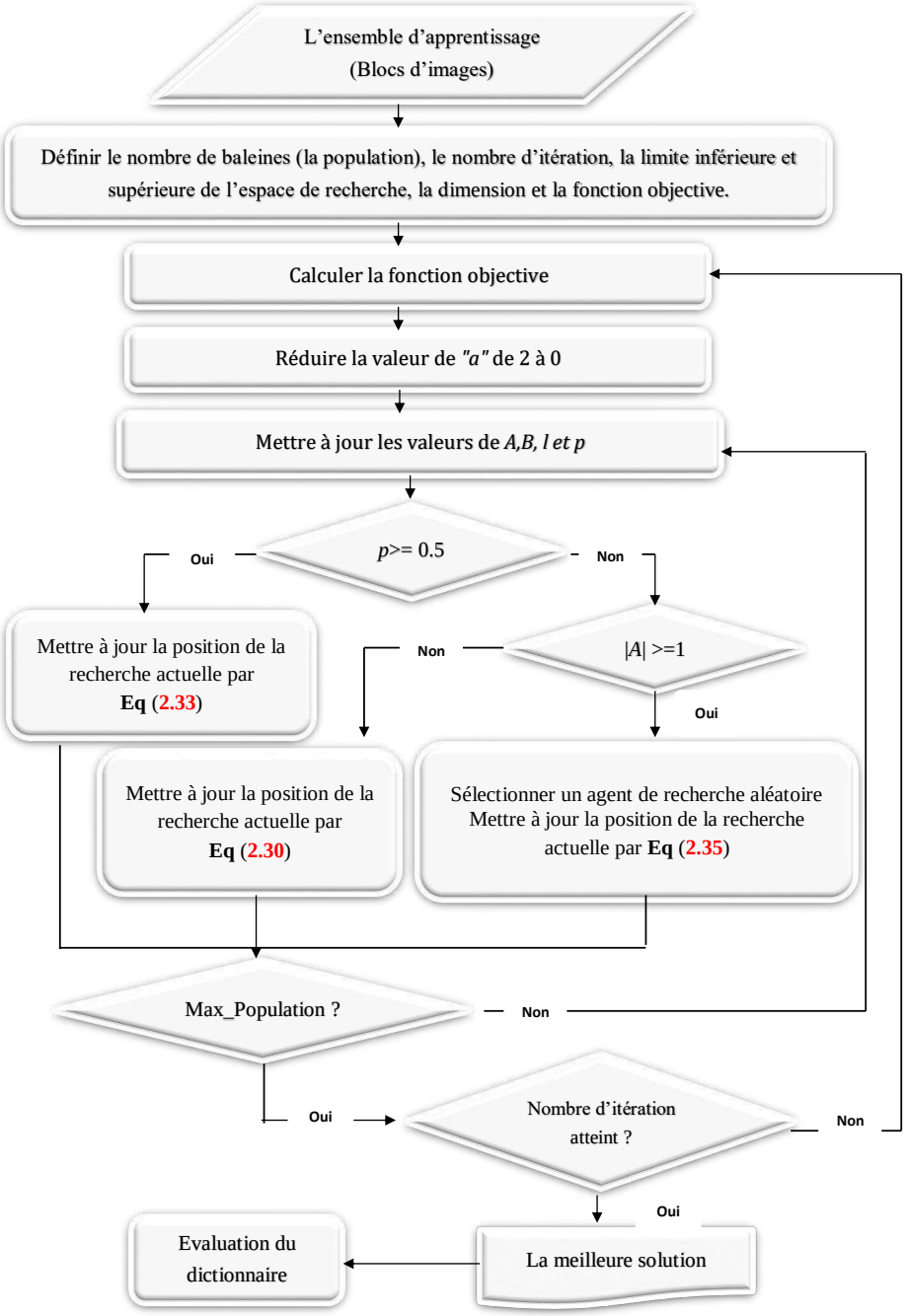


FIGURE 4.5: Diagramme le l'algorithme WOA..

Les programmes des cinq algorithmes sont développés en langage C sour Visual Studio sur un ordinateur personnel avec un CPU de 2.30 GHz. 8 Go RAM sous le système Window 10. La performance des méthodes utilisées a été évaluée par la qualité

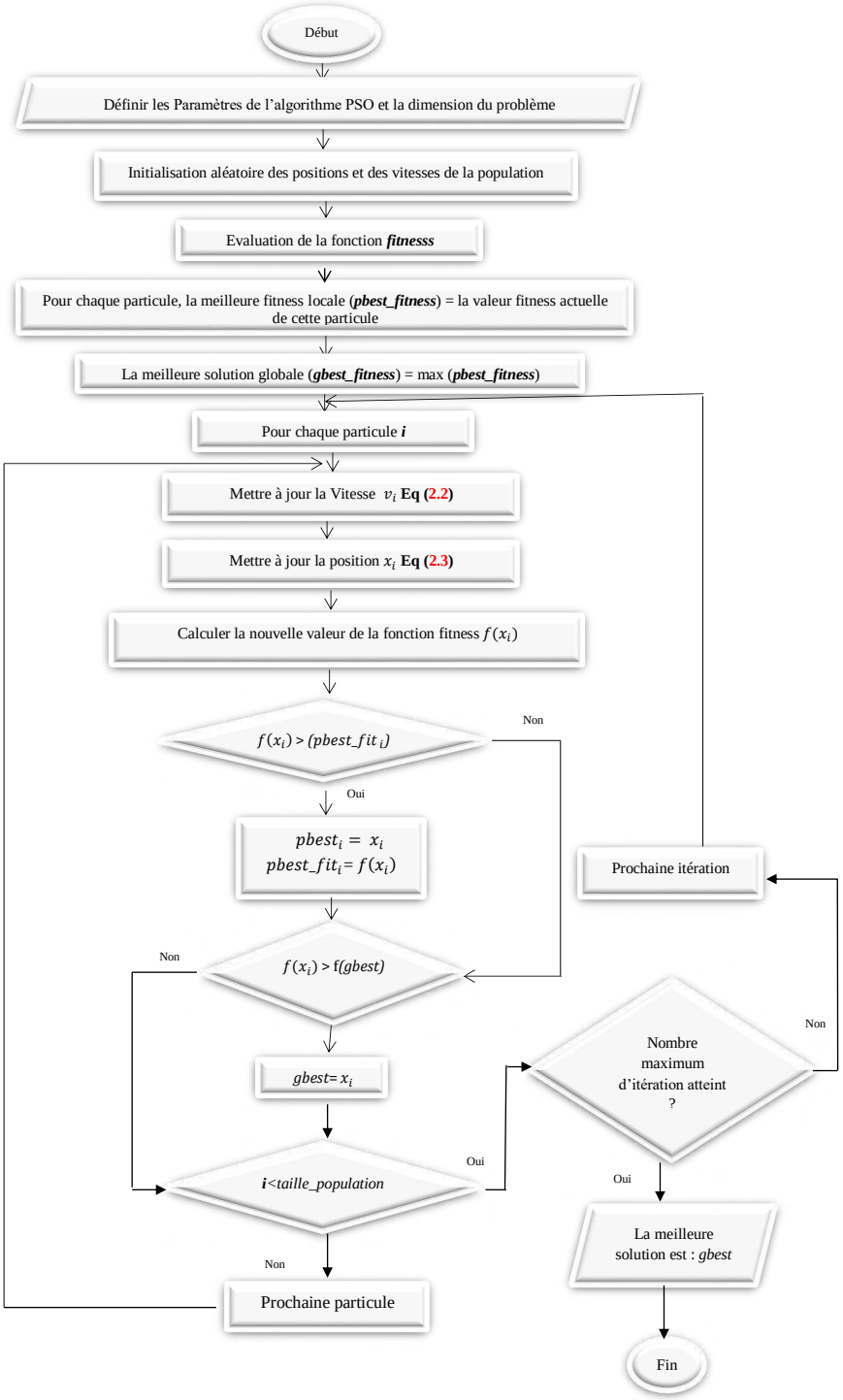


FIGURE 4.6: Diagramme le l'algorithme PSO.

de l'image codée, en utilisant la métrique PSNR et par le temps d'exécution nécessaire

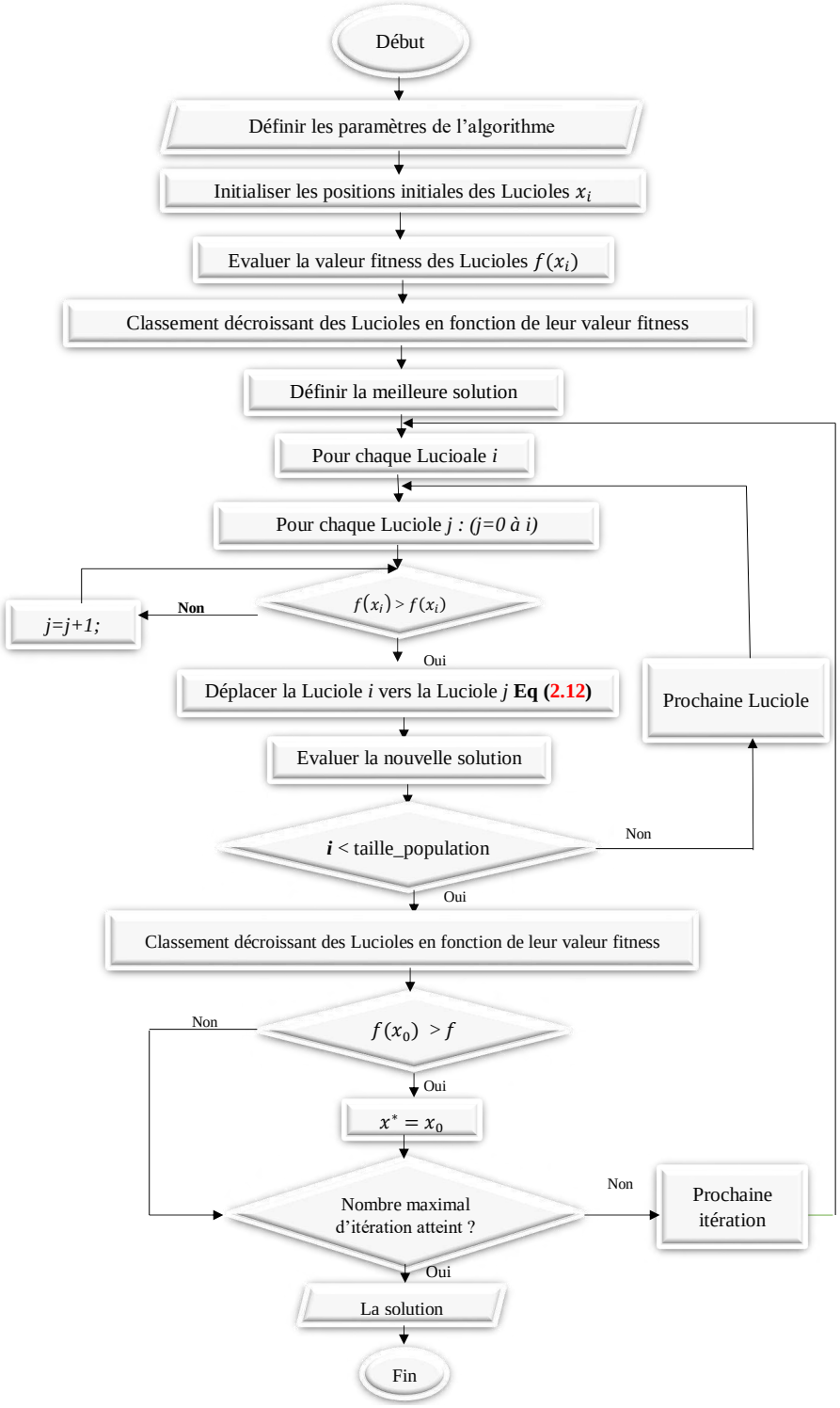


FIGURE 4.7: Diagramme le l'algorithme Firefly.

pour la conception du dictionnaire. Le débit binaire est défini dans l'équation (4.21).

$$\text{Bit\_Rate} = \frac{\log_2 N_c}{K} \tag{4.21}$$

TABLE 4.2: Paramètres de l'algorithme PSO

Paramètres	Valeur
Taille de Population	30
nombre d'itérations	100
Lower bound	0
Upper bound	255
w_min	0.4
w_max	0.9
C_1	2
C_2	2.1
v_min	-1
v_max	1

TABLE 4.3: Paramètres de l'algorithme Firefly (Luciole)

Paramètres	Valeur
Taille de Population	30
nombre d'itérations	100
Lower bound	0
Upper bound	255
α	0.01
γ	2.0
β_0	1.0

N_c représente la taille du dictionnaire conçu et K représente le nombre de pixels de chaque bloc.

Lors de nos expérimentations, chacun des algorithmes est éxécuté 20 fois.

Les Figures 4.8 et 4.9 présentent la valeur moyenne de *PSNR* des images « ***Lena*** » et « ***Baboon*** » codées par le dictionnaire obtenu à partir de l'ensemble d'apprentissage composé de *quatre images*. Ces résultats indiquent que l'algorithme **LBG-Split** avec dichotomie vectorielle produit le *meilleur dictionnaire*, suivis de LBG-Rand et que l 'utilisation des algorithmes métaheuristiques (WOA, PSO et FF) n'apporte aucune amélioration au résultat de l'algorithme LBG. Contrairement à ce qu'il a été rapporté par (Horng, 2012) et (Horng and Jiang, 2011) stipulant que le résultat de l'algorithme LBG se stagne au débit binaire= 0.35 (taille de dictionnaire =64) et ne connait aucune amélioration, nous prenons comme exemple la Figure 4.10 qui montre leurs résultats obtenu pour l'image « Lena ». De plus, les algorithmes WOA, PSO et FA produisent des résultats *quasi-semblables*, nous remarquons aussi que les algorithmes LBG-WOA, LBG-PSO et LBG-FA donnent de meilleurs résultats comparés à WOA, PSO et FA.

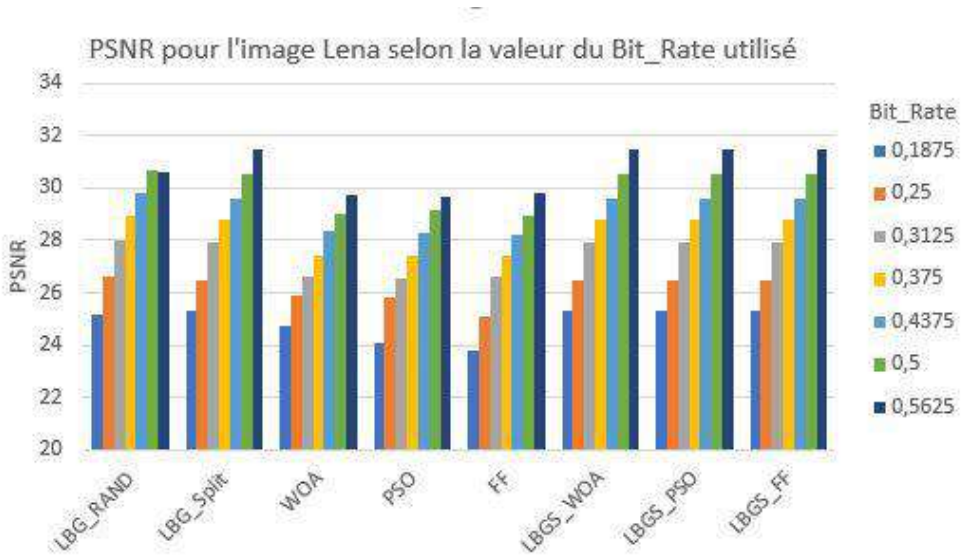


FIGURE 4.8: La valeur moyenne de PSNR de l'image **Lena** codée par le dictionnaire obtenu à partir de quatre différentes images.

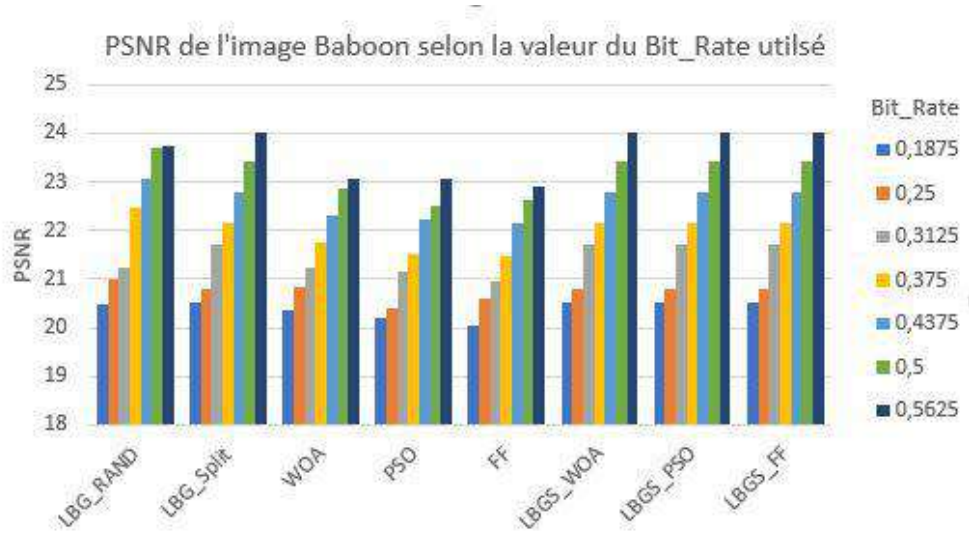


FIGURE 4.9: La valeur moyenne de PSNR de l'image **Baboon** codée par le dictionnaire obtenu à partir de quatre différentes images.

La Figure 4.11 montre le résultat moyen de PSNR obtenu lors du codage de l'image « **GoldHill** » en utilisant le dictionnaire obtenu à partir de la base d'apprentissage des quatre images. D'après ces résultats nous constatons que les algorithmes *LBG-Rand* et *LBG-Split* produisent des dictionnaires *efficaces* même pour le codage des images ne faisant pas partie de la base d'apprentissage utilisée.

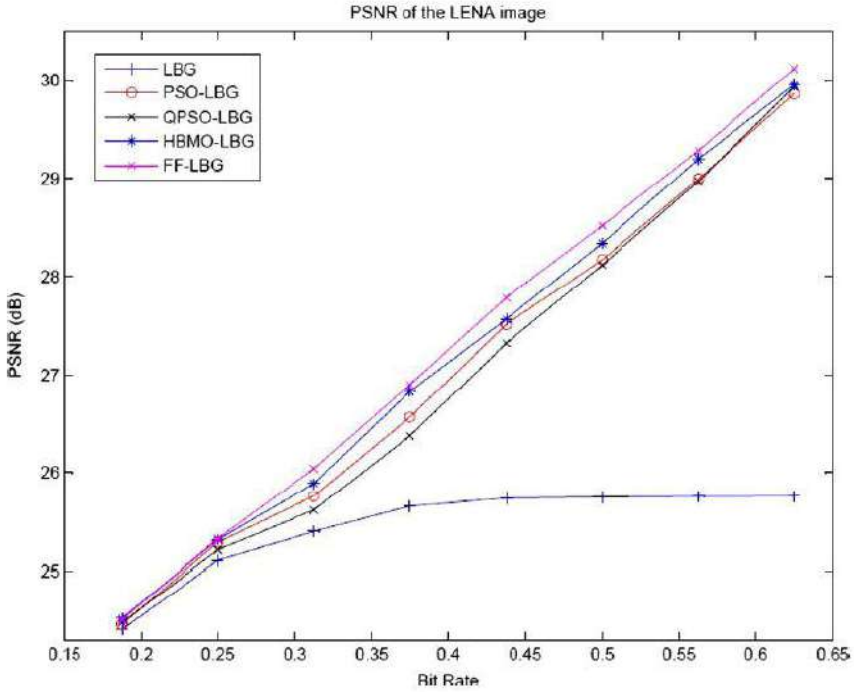


FIGURE 4.10: La moyenne (PSNR) de l'image LENA des cinq différentes méthodes de quantification vectorielle (Horng, 2012).

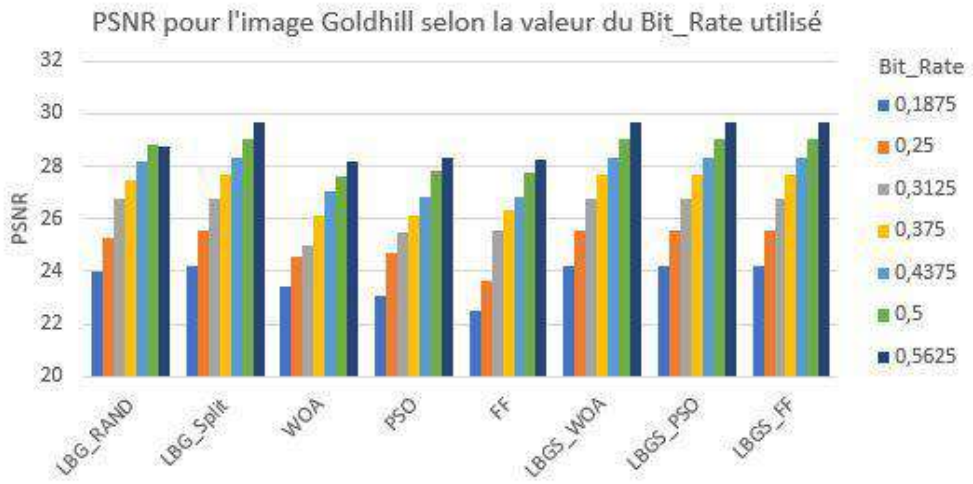


FIGURE 4.11: [La valeur moyenne de PSNR de l'image **Goldhill** codée par le dictionnaire obtenu à partir de quatre différentes images.

Pour une meilleure comparaison des algorithmes LBG traditionnels : aléatoire (Random) et par dichotomie vectorielle (Splitting), nous avons les Figures 4.12 et 4.13 qui présentent les résultats obtenus lorsque le dictionnaire utilisé lors du codage est produit à partir de l'image à coder elle-même. Cependant, nous remarquons que ces deux algorithmes produisent presque les mêmes résultats en termes de PSNR, la différence

est très minime. Par contre, lorsqu'il sagit du *temps de calcul*, l'algorithme ***LBG-Split*** dépasse de loin l'algorithme LBG-Rand. En effet, les tableaux 4.4 - 4.10 montrent que l'algorithme ***LBG-Split*** est le plus rapide de tous les algorithmes utilisés et que les algorithmes WOA, PSO et FF nécessitent un temps de calcul très élevé par rapport aux algorithmes LBG (Rand et Split), ce qui ne favorise surtout pas leur utilisation pour la résolution du problème de la QV

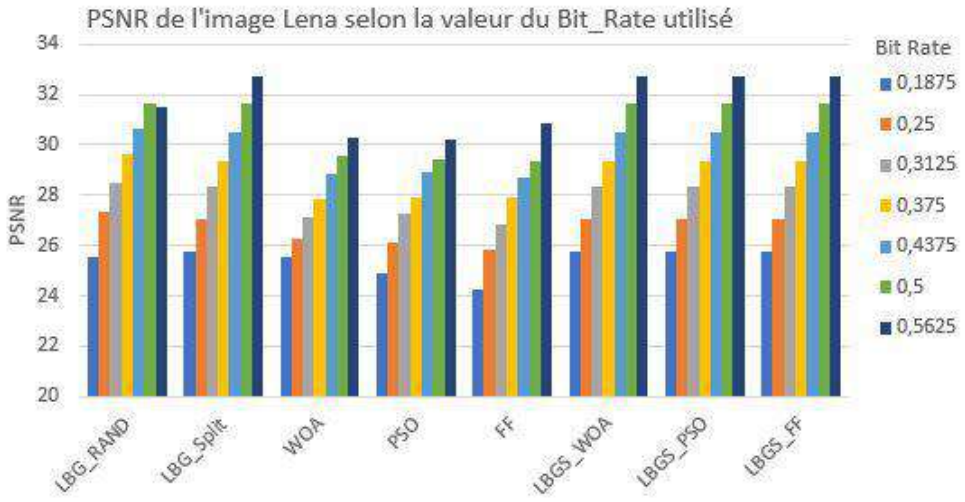


FIGURE 4.12: La valeur moyenne de PSNR de l'image **Lena** codée par le dictionnaire obtenu à partir de l'image elle-même.

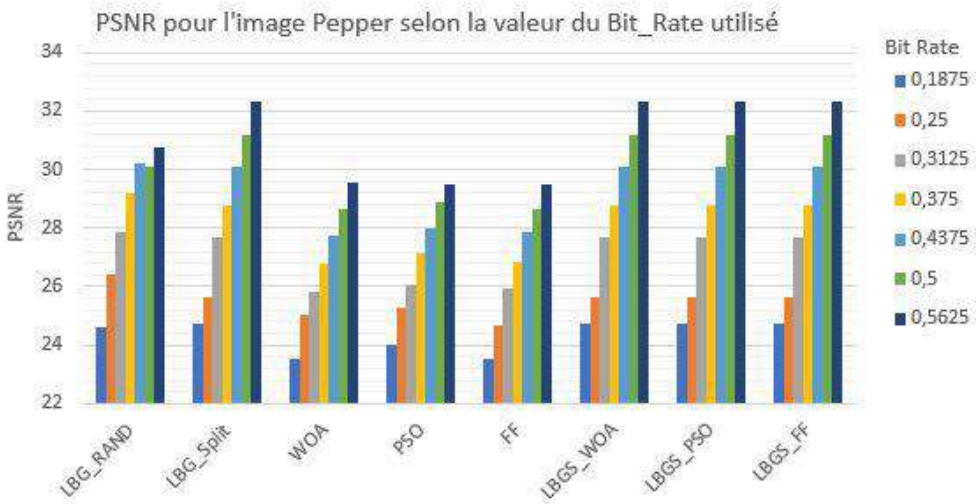


FIGURE 4.13: La valeur moyenne de PSNR de l'image **Pepper** codée par le dictionnaire obtenu à partir de l'image elle-même.

TABLE 4.4: Le temps de calcul moyen des images de test en utilisant les cinq (différents) algorithmes avec un débit binaire = 0.1875

Images de test	taille du dictionnaire	Temps de calcul moyen (sec)				
		LBG (Rand)	LBG (Split)	WOA	PSO	FA
DB (4 images)	8	1.004	**0.185**	22.602	28.028	28.305
Lena	8	0.247	**0.059**	5.923	7.189	7.234
Pepper	8	0.297	**0.068**	6.432	7.204	7.205

TABLE 4.5: Le temps de calcul moyen des images de test en utilisant les cinq (différents) algorithmes avec un débit binaire = 0.25

Images de test	taille du dictionnaire	Temps de calcul moyen (sec)				
		LBG (Rand)	LBG (Split)	WOA	PSO	FA
DB (4 images)	16	1.652	**0.341**	45.127	55.177	54.663
Lena	16	0.445	**0.176**	11.709	13.820	13.893
Pepper	16	0.567	**0.112**	11.240	13.78	13.912

TABLE 4.6: Le temps de calcul moyen des images de test en utilisant les cinq (différents) algorithmes avec un débit binaire = 0.3125

Images de test	taille du dictionnaire	Temps de calcul moyen (sec)				
		LBG (Rand)	LBG (Split)	WOA	PSO	FA
DB (4 images)	32	3.521	**1.667**	91.616	107.901	107.095
Lena	32	0.848	**0.413**	21.951	27.932	26.86
Pepper	32	0.986	**0.524**	20.023	26.720	26926

TABLE 4.7: Le temps de calcul moyen des images de test en utilisant les cinq (différents) algorithmes avec un débit binaire = 0.375

Images de test	taille du dictionnaire	Temps de calcul moyen (sec)				
		LBG (Rand)	LBG (Split)	WOA	PSO	FA
DB (4 images)	64	6.482	**3.101**	175.032	216.178	209.701
Lena	64	1.826	**0.837**	43.385	52.821	52.714
Pepper	64	1.897	**1.056**	47.543	62.482	52.743

TABLE 4.8: Le temps de calcul moyen des images de test en utilisant les cinq (différents) algorithmes avec un débit binaire = 04375

Images de test	taille du dictionnaire	Temps de calcul moyen (sec)				
		LBG (Rand)	LBG (Split)	WOA	PSO	FA
DB (4 images)	128	12.004	**6.963**	342.895	419.509	424.726
Lena	128	3.294	**1.455**	86.769	104.984	104.926
Pepper	128	3.402	**1.898**	99.850	105.475	204.967

TABLE 4.9: Le temps de calcul moyen des images de test en utilisant les cinq (différents) algorithmes avec un débit binaire = 0.5

Images de test	taille du dictionnaire	Temps de calcul moyen (sec)				
		LBG (Rand)	LBG (Split)	WOA	PSO	FA
DB (4 images)	256	24.701	**13.481**	683.274	844.456	770.532
Lena	256	6.787	**3.527**	173.953	209.123	187.966
Pepper	256	6.731	**3.281**	201.361	209.419	200.055

TABLE 4.10: Le temps de calcul moyen des images de test en utilisant les cinq (différents) algorithmes avec un débit binaire = 0.5625

Images de test	taille du dictionnaire	Temps de calcul moyen (sec)				
		LBG (Rand)	LBG (Split)	WOA	PSO	FA
DB (4 images)	512	93.534	**24.303**	1384.916	1667.415	1521.490
Lena	512	28.112	**6.281**	341.321	417.050	372.001
Pepper	512	27.095	**6.420**	398.543	417.754	374.656

4.6 Conclusion

Dans cette étude, nous avons proposé l'utilisation d'un nouvel algorithme méta-heuristique, notamment, **WOA** pour la résolution du problème de la QV, particulièrement la production d'un dictionnaire optimal. La performance de cet algorithme a été comparée à celle obtenue par les algorithmes LBG-Rand, LBG-Split, PSO et Firfly. Tous nos résultats expérimentaux ont montré que l'utilisation de l'algorithme classique ***LBG-Split*** reste le choix le **plus judicieux** pour la production du dictionnaire et cela en termes de qualité de l'image codée et en termes de temps de calcul. D'autant plus qu'il a été montré que les algorithmes méta-heuristiques n'apportent aucune amélioration dans ce domaine-là.

Chapitre 5

Contribution à la parallélisation des algorithmes méta-heuristiques appliqués au multi-seuillage basé sur SSIM et PSNR

5.1 Introduction

La segmentation d'image est considérée comme l'une des méthodes les plus fondamentales dans le traitement d'image (Pal and Pal, 1993; Zhang, Fritts, and Goldman, 2008). Elle consiste à diviser l'image en entrée en régions homogènes tout en respectant des critères bien déterminés : la couleur, la structure de la texture ou bien l'intensité du niveau de gris (Fu and Mui, 1981a; Gonzalez and Woods, 2006). Le but de cette division est d'extraire les parties les plus significatives pour faciliter l'analyse et l'interprétation des images.

Le multi-seuillage joue un rôle important dans la segmentation d'image (Haralick and Shapiro, 1985). Il est pratique pour l'extraction des objets de l'arrière-plan et pour la distinction des objets ayant des intensités différentes.

Il est à noter que le multi-seuillage est un axe de recherche très actif car la multi-modalité des histogrammes de certaines images rend la sélection des seuils difficile et cela affecte l'efficacité de la fonction objective.

Récemment, les mesures les plus connues pour l'évaluation de la qualité des images SSIM (*Structural SIMilarity*) et PSNR (*Peak Signal to Noise Ration*) (Wang et al., 2004; Wang and Bovik, 2009) ont été utilisées comme fonctions objectives optimisées par différents algorithmes méta-heuristiques afin d'améliorer la sélection des seuils.

A nos connaissances, l'utilisation du SSIM comme fonction objective pour résoudre le problème du multi-seuillage a été proposée pour la première fois par (Balabanian, Silva, and Pedrini, 2017). Dans leur travail, les auteurs ont prouvé l'efficacité du SSIM par rapport à l'approche classique Otsu. Cependant, l'étude proposée ne prend en considération que le seuillage local à deux niveau (*binarisation*).

Par ailleurs, l'étude faite dans (Kotte, Kumar, and Injeti, 2018) traite le multi-seuillage global en utilisant PSNR comme fonction objective optimisée par un algorithme de recherche différentiel amélioré. Leurs résultats montrent que l'approche PSNR a de meilleures performances comparées aux méthodes classiques Otsu et Kapur. En revanche, les auteurs ne semblent pas bénéficier de la recherche exhaustive pour ajuster leurs algorithmes afin d'améliorer leurs PSNR obtenus. De plus, aucune comparaison avec l'utilisation de la mesure SSIM n'est notée.

Dans notre travail (Boubechal, Seghir, and Benzid, 2019), nous avons amélioré l'utilisation des mesures ***SSIM*** et ***PSNR*** afin d'obtenir de meilleures performances que celles des méthodes proposées dans la littérature. Pour ce faire, nous avons d'abord généralisé l'utilisation de la mesure SSIM pour le multi-seuillage en comparaison avec (Balabanian, Silva, and Pedrini, 2017). Ensuite, nous avons implémenté l'algorithme de recherche exhaustive en utilisant SSIM, PSNR et Otsu comme fonctions objectives dans le but d'obtenir les solutions exactes, lorsque cela est possible (compléxité de temps de calcul). Par la suite, nous avons sélectionné trois algorithmes bien connus et largement utilisés de l'intelligence en essaim, notamment : **PSO**, **FA** et **BA** et nous avons ajusté empiriquement leurs paramètres afin de produire des solutions quasi-exactes. Dans ce contexte, la mise en œuvre de la recherche exhaustive s'est avérée *intéressante* car elle a permis de vérifier la précision des solutions des méta-heuristiques (les solutions les plus proches de l'optimum global). Mais, bien entendu, le temps d'exécution, notamment lors de l'utilisation de la recherche exhaustive, augmente exponentiellement en fonction du nombre de seuils. Par conséquent, nous avons également implémenté tous les algorithmes en parallèle en utilisant la bibliothèque standard de la programmation parallèle à mémoire partagée **OpenMP**, ce qui a conduit à une amélioration significative des performances des algorithmes.

L'étude expérimentale que nous avons menée sur 110 images montre que notre approche, produisant des solutions quasi-exactes, est plus performante que les méthodes

classiques. De plus, la complexité de calcul a été considérablement réduite par notre implémentation parallèle.

5.2 Application des méta-heuristiques dans la résolution du problème du multi-seuillage

La sélection des seuils dans le multi-seuillage est considérée comme un problème NP-Difficile; car il n'existe pas d'algorithme qui produit des solutions optimales en un temps polynomial. Ce problème a retenu l'attention des chercheurs durant ces dernières décennies et la majorité d'entre eux ont appliqué les algorithmes évolutionnaires et des méta-heuristiques pour y faire face. Parmi ces travaux nous citons :

(Hammouche, Diaf, and Siarry, 2010) présentent une comparaison de six différents algorithmes (**GA**,**PSO**, **DE**, **TS**, **ACO** et **SA**) appliqués à l'optimisation de la fonction Otsu. Ces auteurs concluent que l'algorithme **PSO** est meilleur que les autres algorithmes utilisés en termes de précision, d'efficacité et de temps d'exécution.

(Horng, 2011) a proposé une nouvelle approche pour le multi-seuillage nommée *Maximum Entropy Based Artificial Bee Colony Thresholding*. Dans son travail, il compare MEABCT avec d'autres méthodes proposées dans la littérature où il constate que cette approche est meilleure en termes de temps de calcul et de valeurs de fonction objective par rapport aux méthodes : **PSO**, **HCOCLPSO** et **HBMO**.

(Agrawal et al., 2013) ont appliqué l'algorithme *Cucko Search* dans la maximisation de l'entropie de Tsallis et ont comparé leurs résultats avec ceux obtenus en appliquant **BFO**, **PSO**, **ABC** et **GA**. D'après leur analyse, l'algorithme **CS** réalise de meilleures performances que les autres techniques utilisées.

(Bhandari et al., 2014)) ont employé deux algorithmes de l'intelligence en essaim : *Cucko Search* et *Wind Driven Optimization* pour le seuillage multi-niveaux des images satellitaires en utilisant l'entropie de Kapur comme fonction objective. En se basant sur les valeurs de la fonction objective obtenues, ces auteurs trouvent que l'utilisation des algorithmes **CS** et **WDO** peut être avantageuse pour la résolution du problème de segmentation.

Quant à (El Aziz, Ewees, and Hassanien, 2017), ils ont montré l'éficacité des algorithmes **WOA** et **MFO** dans l'optimisation de la méthode Otsu dans la segmentation basée sur le multi-seuillage.

5.3 Déscription formelle du problème de multi-seuillage

L'objectif du multi-seuillage est de trouver les seuils optimaux en se basant sur l 'histogramme de l'image. La sélection des seuils se fait selon des critères bien définis, ils peuvent être déterminés soit par une minimisation ou une maximisation d'une fonction objective. Cette dernière utilise les seuils sélectionnés comme paramètres d'entrée. Le processus peut être décrit comme suit (Bhandari et al., 2016; Oliva and Cuevas, 2016) :

$$\begin{aligned} C_1 &\longleftarrow p \quad if \quad 0 \leq p < th_1 \\ C_2 &\longleftarrow p \quad if \quad th_1 \leq p < th_2 \\ C_i &\longleftarrow p \quad if \quad th_{i-1} \leq p < th_i \\ C_M &\longleftarrow p \quad if \quad th_m \leq p < L-1 \end{aligned}$$

Tel que : p représente chaque pixel de l'image, qui peut avoir 256 nuances de niveaux de gris; notés $(L = 0, 1, \ldots, 255)$), classés en $(M = m + 1)$ classes $(C_1, C_2, ..., C_M)$ utilisant m seuils $(th_1, th_2, ..., th_m)$.

L'image segmentée est générée par quantification, où la valeur du niveau de gris attribuée à tous les pixels appartenant à la même classe est calculée par leur moyenne.

Parmi les techniques classiques les plus efficaces pour effectuer le seuillage des images, nous avons choisi la méthode de *variance entre classes* Otsu (Otsu, 1979). Cette méthode est discutée en détail dans le chapitre 3 (section 3.3.3).

La méthode d'Otsu a montré son efficacité dans la segmentation d'images. Néanmoins, le temps d'exécution augmente exponentiellement avec le nombre de seuils. Pour remédier à ce problème, le multi-seuillage est traité comme un problème d'optimisation résolu par des méta-heuristiques, comme montré sur la Figure 5.1. Nous avons choisi trois algorithmes faisant partie de l'intelligence en essaim.

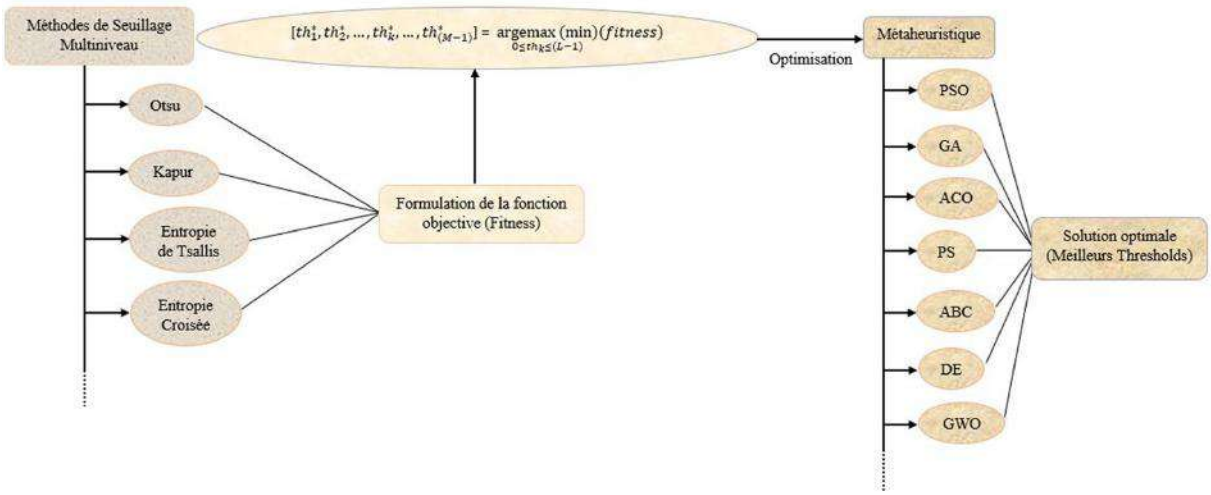


FIGURE 5.1: Résolution du problème de segmentation par les méta-heuristiques

5.4 Les techniques de l'intelligence en essaim utilisées dans notre approche

Plusieurs techniques de l'intelligence en essaim (*Swarm Intelligence* en anglais) ont été proposées dans la littérature pour résoudre les problèmes d'optimisation globale. Cependant, dans notre travail, nous avons utilisé trois algorithmes connus par leur efficacité à résoudre différents problèmes, notamment : **PSO**, **FA** et **BA**. Ces trois algorithmes ont été discutés en détail dans le chapitre 2 (section 2.3.3, 2.3.5 et 2.3.6).

L'objectif principal des algorithmes susmentionnés est de déterminer le maximum ou le minimum d'une fonction mathématique donnée, appelée la fonction objective (*fitness* ou *fonction coût*). Ces trois algorithmes (PSO, FA et BA) ont montré leur efficacité dans la résolution de problèmes d'optimisation globale. Dans notre étude, nous les avons appliqués pour évaluer l'efficacité des fonctions objectives de *PSNR* et de *SSIM* dans la résolution du problème de multi-seuillage. Des versions parallèles de ces algorithmes sont également proposées et mises en œuvre afin d'améliorer leur temps d'exécution.

5.5 L'approche proposée

Dans notre travail, résumé dans la Figure 5.2, nous proposons d'explorer et d' améliorer l'utilisation des mesures de qualité les plus utilisées **SSIM** (Eq.5.1) et **PSNR** (Eq.5.2)

en tant que fonctions objectives dans le but d'obtenir de meilleurs résultats que ceux rapportés par (Balabanian, Silva, and Pedrini, 2017) et (Kotte, Kumar, and Injeti, 2018) dans la résolution du problème de multi-seuillage. Le premier article utilise SSIM comme fonction objective pour traiter le problème de *binarisation* uniquement. Quant au second, il se base sur la mesure PSNR pour résoudre le problème de multi-seuillage. Mais, l'étude semble *moins performante* par rapport à la notre, comme montré à la section 5.6.

Similarité structurelle (*Structural SIMilarity*): est une métrique de qualité proposée par (Wang et al., 2004) afin de déterminer la similarité structurelle entre deux images. Sa valeur est comprise entre 0 et 1 indiquant la corrélation entre l'image originale et l'image traitée. Plus cette valeur est proche de 1, plus les deux images sont similaires. Le SSIM est défini mathématiquement comme suit :

$$\text{SSIM}(x,y) = \frac{(2\mu_x\mu_y + C_1)\,(2\sigma_{xy} + C_2)}{\left(\mu_x^2 + \mu_y^2 + C_1\right)\left(\sigma_x^2 + \sigma_y^2 + C_2\right)} \tag{5.1}$$

Où :
μ_x représente la moyenne de l'image originale x
μ_y est la moyenne de l'image seuillée y
σ_x^2 est la variance de x et σ_y^2 celle de y
σ_{xy} est la covariance de x et y
$C_1 = (k_1 \times L)^2$ et $C_2 = (k_2 \times L)^2$ $(L = 255, k_1 = 0.01$ et $k_2 = 0.03)$.

Rapport signal-bruit (*Peak Signal-to-Noise Ratio*): est également une mesure de qualité largement utilisée pour évaluer la qualité des images. Le PSNR permet de déterminer la distorsion d'une image traitée par rapport à sa source (Wang and Bovik, 2009). Plus le PSNR est élevé, plus l'image reconstruite est similaire à l'image originale. En raison de l'utilisation du logarithme, le PSNR est exprimé en décibels (dB). La valeur PSNR se calcule comme suit :

$$PSNR = 20\log_{10}\left(\frac{255}{\sqrt{MSE}}\right) \tag{5.2}$$

où *l'erreur quadratique moyenne* "MSE" (mean square error) est exprimée comme suit :

$$MSE = \frac{\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}(x(i,j) - y(i,j))^2}{M \times N} \tag{5.3}$$

x et y représentent l'image originale et l'image segmentée de taille MxN, respectivement.

Les *seuils optimaux* basés sur les approches SSIM et PSNR sont obtenus en utilisant les équations (5.4) et (5.5), respectivement.

$$th^* = \{th_0^*, th_1^*, \cdots, th_k^*, \cdots, th_{M-1}^*\} = \underset{0 \leq th_k \leq (L-1)}{\arg\max}\ SSIM(x, y) \tag{5.4}$$

$$th^* = \{th_0^*, th_1^*, \cdots, th_k^*, \cdots, th_{M-1}^*\} = \underset{0 \leq th_k \leq (L-1)}{\arg\max}\ PSNR(x, y) \tag{5.5}$$

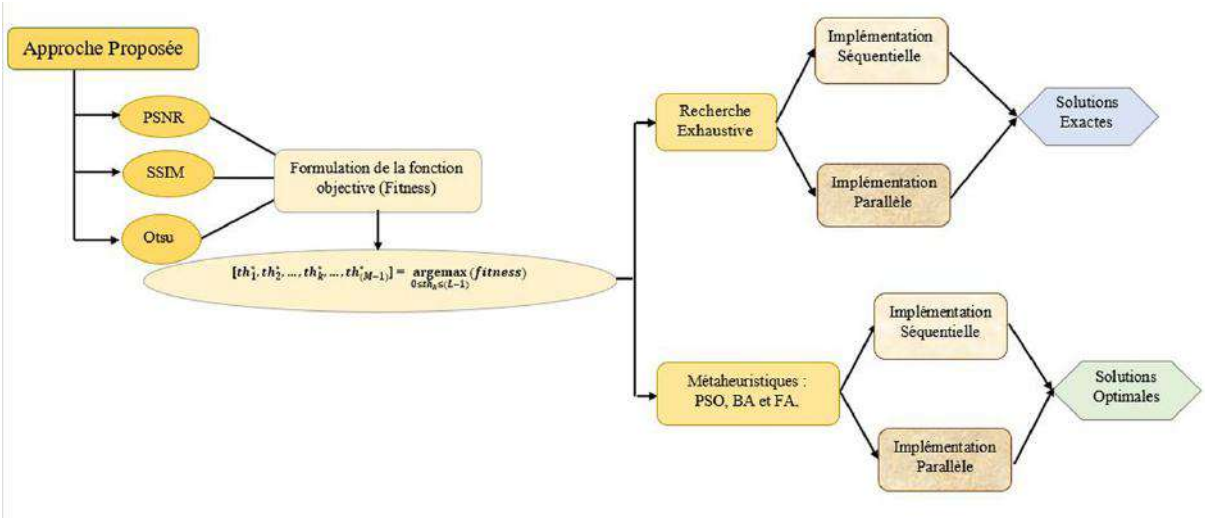


FIGURE 5.2: Principe de notre approche

Nous avons tout d'abord implémenté un algorithme de recherche exhaustive parallèle permettant de déterminer les meilleurs seuils pour une fonction objective donnée. Nous entendons par les meilleurs seuils, ceux qui conduisent à la valeur maximale exacte de la fonction objective. Cela est effectué en calculant la fonction objective pour toutes les combinaisons possibles de seuils et en sélectionnant ceux ayant la valeur optimale. Bien sûr, la complexité de ce calcul augmente exponentiellement en fonction du nombre de seuils, mais notre algorithme parallèle est capable de produire des solutions exactes, dans un temps raisonnable, en utilisant Otsu, et jusqu'à **3** et **4** seuils en utilisant SSIM et PSNR, respectivement. En se basant sur les solutions produites par l'algorithme de seuillage de recherche exhaustive parallèle, nous avons ajusté les paramètres des trois algorithmes d'intelligence en essaim : PSO, FA et BA afin qu'ils puissent produire des solutions quasi-exactes plus rapidement. En effet, nous avons implémenté ces algorithmes, en séquentiel et en parallèle, pour produire les meilleurs

seuils en utilisant SSIM, PSNR et Otsu comme fonctions objectives. Les valeurs de PSNR et de SSIM sont calculées entre l'image donnée et l'image seuillée; une valeur plus élevée indique une meilleure qualité de seuillage. Une étude comparative détaillée est fournie dans la section 5.6.

Les méthodes de méta-heuristiques sont des algorithmes itératifs qui répètent certaines étapes afin d'améliorer les solutions courantes jusqu'à l'obtention de valeurs satisfaisantes. Plus l'algorithme itère, plus la solution obtenue est meilleure (proche de l'optimum global). Malheureusement, l'augmentation du nombre d'itérations pourrait entraîner un temps de calcul énorme. Pour faire face à cet inconvénient, il est généralement recommandé d'effectuer les calculs en parallèle. Dans notre travail, nous utilisons le standard de programmation parallèle à mémoire partagée OpenMP (Dagum and Menon, 1998) afin de réduire le temps d'exécution des différentes méta-heuristiques qui nous intéressent. Nous avons également mis en œuvre un algorithme de recherche exhaustive parallèle pour obtenir les solutions exactes quand cela est possible, dans le but de les comparer aux solutions obtenues par les algorithmes méta-heuristiques. Cette comparaison nous a aidé à détecter lequel des algorithmes méta-heuristiques est plus précis.

5.6 Résultats et discussions

Dans notre travail nous avons utilisé le langage C et la bibliothèque OpenMP[1] pour implémenter les méthodes proposées. Les tests ont été effectués sur une machine ayant : un processeur Intel® Core™ i56-4590S, 3,00G Hz, 8 G de RAM et un système d'exploitation Windows 7 (64) Pro.

5.6.1 L'ensemble des images utilisées

L'étude expérimentale a été réalisée sur un ensemble de 110 images en niveau de gris, dont cent-quatre (104) sont des images pour la segmentation proposée par l'université Berkeley (BSD 500)[2]. Les six restantes représentent les images de test standard (Lena, Barbara, Baboon, Cameraman, Pepper et GoldHill) de taille 512 x 512 pixels. Les résultats sont donnés en détail pour les images (Lena, Cameraman, Baboon, Elephants

[1] Bibliothèque de Programmation parallèle à Mémoire Partagée
[2] https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/

et Man). Les Figures 5.3 et 5.4 présentent ces images accompagnées par leur histogrammes, et en termes de moyenne générale des valeurs obtenues pour tout l'ensemble des cent-dix images.

Pour les trois algorithmes de méta-heuristiques utilisés (PSO, FA et BA), nous avons fixé le nombre d'itération à 100 et la taille de population à 40.

Les paramètres globaux par défauts des algorithmes PSO, FA et BA n'ont pas donné des solutions satisfaisantes. Les Tableaux 5.1, 5.2 et 5.3 présentent les paramètres que nous avons trouvé efficaces et bien meilleurs pour le multi-seuillage. Pour cela, nous avons dû les ajuster manuellement en nous basant sur les solutions exactes produites par l'algorithme de la recherche exhaustive parallèle.

TABLE 5.1: Paramétres de l'algorithme PSO

Paramétres	Valeur
Taille de Population	40
nombre d'itération	100
Lower bound	0
Upper bound	255
w_min	0.4
w_max	1.2
C_1	2
C_2	2.1
v_min	-50
v_max	50

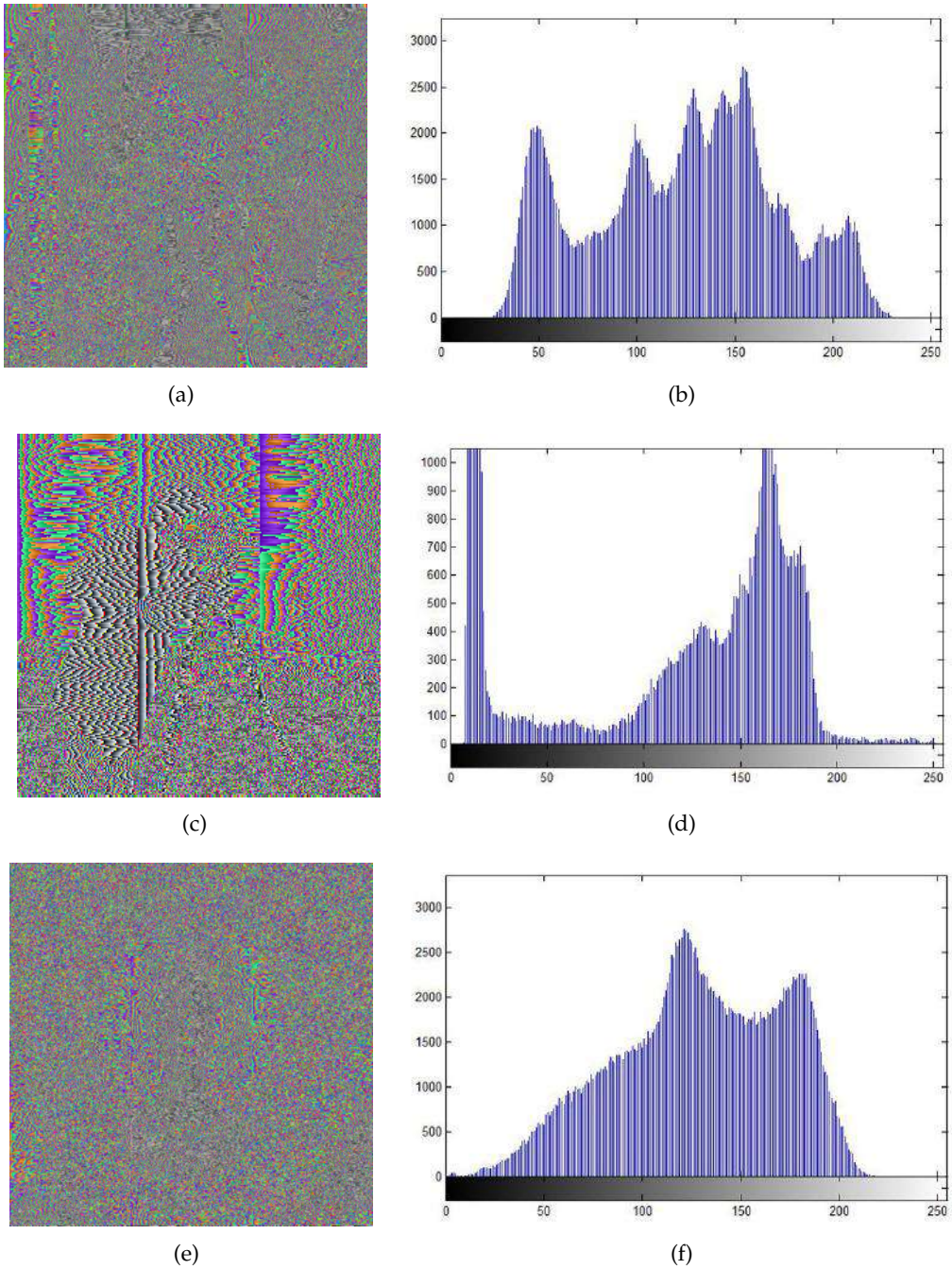
TABLE 5.2: Paramétres de l'algorithme Firefly (Luciole)

Paramétres	Valeur
Taille de Population	40
nombre d'itération	100
Lower bound	0
Upper bound	255
α	0.9
γ	0.00001
β_0	1

5.6.2 Analyse de la qualité des solutions obtenues

Évaluation de la performance des algorithmes PSO, FA et BA

Les valeurs des fonctions objectives des images des Figures 5.3 et 5.4 ont été obtenues par les algorithmes PSO, FA et BA, elles sont présentées dans les tableaux 5.4, 5.5 et



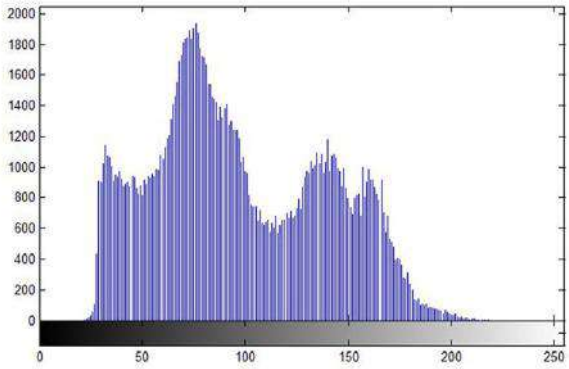
(a) (b)

(c) (d)

(e) (f)

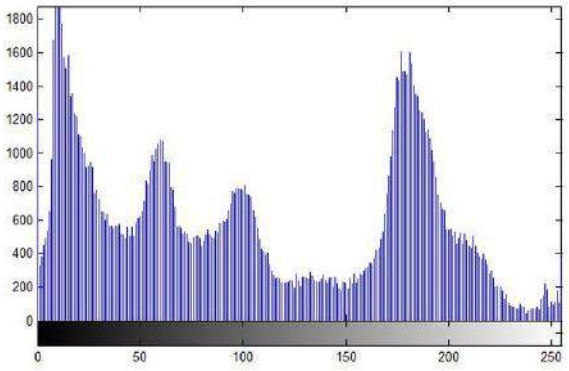
FIGURE 5.3: Images benchmark: (a) Lena; (c) Cameraman; (e) Baboon et leurs histogrammes (b) (d) (f) respectivement.

(a)



(b)

(c)



(d)

FIGURE 5.4: Deux images de la base de données (BSD 500): (a) Elephants; (c) Man et leurs histogrammes (b) (d) respectivement.

TABLE 5.3: Paramétres de l'algorithme Bat (Chauvesouris)

Paramétres	Valeur
Taille de Population	40
nombre d'itération	100
Lower bound	0
Upper bound	255
f_min	0.001
f_max	0.009
α	0.9
γ	0.005
A_max	50

5.6 et accompagnées de leurs seuils optimaux correspondants.

Comparés aux algorithmes FA et BA, les résultats obtenus montrent que l'algorithme **PSO** produit les solutions les plus proches de celles obtenues par la recherche exhaustive.

Les Figures 5.5,5.6, 5.7, 5.8, 5.9, 5.10, 5.11, 5.12 et 5.13 présentent les images segmentées par les seuils optimaux obtenus par les differents algorithmes en utilisant les fonctions objectives Otsu, PSNR et SSIM.

(Kotte, Kumar, and Injeti, 2018) ont utilisé le PSNR comme fonction objective optimisée par leur algorithme de recherche différentielle amélioré (IDSA). Le tableau 5.7 montre leurs résultats obtenus pour les images Lena, Cameraman et Baboon. Nous pouvons clairement voir que les valeurs PSNR obtenues par leur méthode sont nettement inférieures aux nôtres rapportées dans le tableau 5. Par exemple, les valeurs maximales de PSNR qu'ils ont obtenues lorsque le nombre de seuils est égal à 5 **(Th=5)** sont : **22.7919**, **23.6858** et **22,0016** pour les images Lena, Cameraman et Baboon, respectivement. Alors que nous avons obtenu **29,52**, **29,41** et **29,55** pour les mêmes images.

En se basant sur nos résultats, nous pouvons affirmer que les seuils que nous avons obtenus sont plus proches des seuils exacts par rapport à ceux produits par (Kotte, Kumar, and Injeti, 2018).

Étant donné que les algorithmes méta-heuristiques sont de nature stochastique, les solutions trouvées, après chaque exécution, peuvent ne pas être identiques. Par conséquent, nous avons effectué un autre test pour analyser la précision et la stabilité des trois algorithmes utilisés en calculant la *moyenne* (Mean) et *l'écart-type* (StD) des

TABLE 5.4: Comparaison des meilleures valeurs obtenues de la fonctions objective **Otsu** par (PSO, FA et BA) et leurs seuils correspondants.

		Otsu value				Thresholds			
Images de test	TH	Exhaustive	PSO	FA	BA	Exhaustive	PSO	FA	BA
Lena	2	**1961.58**	**1961.58**	1957.57	1961.24	**93,151**	**93, 151**	88, 152	91, 150
	3	**2128.26**	**2128.26**	2031.29	2127.31	**81,127,171**	**81, 127, 171**	96, 137, 153	79,126,168
	4	2191.60	2191.51	2168.45	2175.15	75,114,145,180	76,114,145,180	85,129,149,183	78,113,152,195
	5	2217.50	2217.28	2184.38	2170.61	73,109,136,160,188	73,110,137,160,188	76,87,103,140,172	21,76,122,142,168
Cameraman	2	**3650.33**	**3650.33**	3628.63	3649.42	**70,144**	**70, 144**	85, 139	74, 144
	3	**3725.71**	**3725.71**	3718.49	3715.75	**59,119,156**	**59, 119, 156**	52, 125, 159	44, 116, 156
	4	3780.68	3780.66	3766.69	3765.95	42,95,140,170	43,95,140,170	23,77,140,174	37,86,133,154
	5	**3812.00**	**3812.00**	3776.11	3796.80	**36,82,122,149,173**	**36,82,122,149,173**	58,83,130,145,162	50,101,116,147,174
Baboon	2	**1548.14**	**1548.14**	1545.33	1547.56	**97, 149**	**97, 149**	95, 152	97, 151
	3	**1638.31**	**1638.31**	1630.39	1623.25	**85, 124, 160**	**85, 124, 160**	81, 120, 165	94, 135, 158
	4	**1692.14**	**1692.14**	1689.53	1678.87	**72, 106, 137, 168**	**72, 106, 137, 168**	76, 111, 141, 167	86, 116, 147, 170
	5	**1717.88**	**1717.88**	1682.16	1687.82	**67, 99, 125, 149, 174**	**67, 99, 125, 149, 174**	76, 88, 101, 145, 173	76, 116, 124, 157, 172
Man	2	**4893.38**	**4893.38**	4884.38	4876.15	**57, 139**	**57, 139**	63, 135	45, 139
	3	**5041.99**	**5041.99**	5017.02	5016.97	**40, 86, 150**	**40, 86, 150**	55, 124, 189	25, 79, 151
	4	**5149.22**	**5149.22**	5068.44	5117.96	**39, 84, 142, 197**	**39, 84, 142, 197**	62, 119, 149, 204	42, 91, 160, 190
	5	**5196.16**	**5196.16**	5177.79	5180.42	**37, 77, 118, 161, 201**	**37, 77, 118, 161, 201**	44, 70, 100, 149, 197	48, 79, 123, 153, 196
Elephants	2	**1533.64**	**1533.64**	1527.20	1533.14	**70, 119**	**70, 119**	77, 126	72, 121
	3	**1626.29**	**1626.29**	1613.01	1600.37	**65, 105, 145**	**65, 105, 145**	57, 110, 147	50, 84, 123
	4	**1675.86**	**1675.86**	1614.17	1641.81	**57, 85, 117, 151**	**57, 85, 117, 151**	77, 106, 116, 141	56, 88, 133, 172
	5	**1695.04**	**1695.04**	1679.87	1675.74	**57, 83, 111, 137, 161**	**57, 83, 111, 137, 161**	65, 80, 100, 117, 147	58, 87, 109, 119, 141

TABLE 5.5: Comparaison des meilleures valeurs Obtenues de la fonctions objective **PSNR** par (PSO, FA et BA) et leurs seuils correspondants.

		PSNR value				Thresholds			
Images de test	TH	Exhaustive	PSO	FA	BA	Exhaustive	PSO	FA	BA
Lena	2	**22.96**	**22.96**	22.81	22.95	**92,152**	**92,152**	84,153	94,149
	3	**26.04**	**26.04**	25.23	25.65	**79,126,171**	**79,126,171**	75,139,169	89,138,181
	4	**28.18**	**28.18**	26.03	26.16	**74,112,144,180**	**74,112,144,180**	89,116,123,172	92,115,127,171
	5	_	**29.52**	26.28	28.09	_	**73,110,137,160,187**	66,119,171,178,184	83,106,145,174,204
Cameraman	2	**24.39**	**24.39**	24.31	24.29	**69,144**	**69,144**	78,142	80,146
	3	**26.06**	**26.06**	26.04	26.01	**58,118,156**	**58,118,156**	59,118,153	63,117,154
	4	**27.87**	**27.87**	26.66	26.85	**42,95,139,170**	**42,95,139,170**	29,83,146,158	49,107,127,151
	5	_	**29.41**	27.14	28.39	_	**37,83,122,149,173**	59,113,117,145,159	53,96,127,156,213
Baboon	2	**24.31**	**24.31**	24.15	24.30	**97, 149**	**97, 149**	104, 151	97, 149
	3	**26.34**	**26.34**	25.45	25.82	**85, 124, 160**	**85, 124, 160**	89, 106, 149	68, 113, 146
	4	**28.24**	**28.24**	26.93	26.30	**72, 106, 137, 167**	**72, 106, 137, 167**	58, 98, 147, 184	85, 121, 159, 254
	5	_	**29.55**	27.17	27.97	_	**64, 95, 121, 147, 172**	83, 131, 151, 172, 199	81, 114, 151, 163, 175
Man	2	**21.96**	**21.96**	21.95	21.88	**57, 139**	**57, 139**	58, 135	65, 147
	3	**23.89**	**23.89**	23.42	23.82	**40, 86, 150**	**40, 86, 150**	47, 115, 167	46, 87, 151
	4	**26.14**	**26.14**	25.69	24.58	**40, 85, 144, 199**	**40, 85, 144, 199**	34, 88, 136, 207	52, 114, 140, 204
	5	_	**27.65**	26.29	26.26	_	**39, 77, 120, 163, 200**	41, 76, 116, 181, 201	32, 73, 142, 201, 227
Elephants	2	**24.63**	**24.63**	24.23	24.62	**70, 119**	**70, 119**	70, 108	70, 117
	3	**26.96**	**26.96**	26.61	26.18	**65, 105, 146**	**65, 105, 146**	65, 94, 140	67, 115, 135
	4	**29.03**	**29.03**	27.44	26.88	**57, 85, 116, 151**	**57, 85, 116, 151**	50, 92, 118, 136	66, 108, 141, 245
	5	_	**30.29**	28.17	28.19	_	**54, 80, 104, 129, 155**	54, 78, 106, 157, 228	43, 81, 109, 116, 162

(_) La solution ne peut être calculée en un temps raisonable.

TABLE 5.6: Comparaison des meilleures valeurs Obtenues de la fonctions objective **SSIM** par (PSO, FA et BA) et leurs seuils correspondants.

		SSIM value				Thresholds			
Images de test	TH	Exhaustive	PSO	FA	BA	Exhaustive	PSO	FA	BA
Lena	2	**0.944850**	**0.944850**	0.941312	0.938636	**89,155**	**89,155**	92,150	104,165
	3	0.985007	0.984894	0.963022	0.961980	85,124,175	75,129,178	86,122,169	56,111,177
	4	_	0.993289	0.981416	0.985963	_	80,109,150,184	80,121,153,176	72,113,151,169
	5	_	0.999165	0.988423	0.993838	_	80,110,142,172,196	73,120,141,186,209	68,112,144,161,177
Cameraman	2	**0.972196**	**0.972196**	0.969106	0.970382	**70 ,143**	**70,143**	85,128	72,146
	3	0.982389	0.982222	0.976299	0.980056	51,110,153	51,107,153	76,139,153	58,111,152
	4	_	**0.989393**	0.982687	0.988221	_	**44,96,139,173**	58,67,133,156	31,96,142,168
	5	_	**0.993984**	0.986314	0.989208	_	**35,81,122,151,176**	56,75,135,156,175	31,62,103,139,153
Baboon	2	**0.960584**	**0.960584**	0.947771	0.955111	**99, 151**	**99, 151**	97, 143	98, 147
	3	**0.982272**	**0.982272**	0.958482	0.963363	**68, 122, 164**	**68, 122, 164**	98, 152, 159	51, 96, 143
	4	_	**0.998290**	0.982217	0.985941	_	**77, 108, 143, 164**	82, 117, 143, 155	83, 101, 139, 172
	5	_	**0.998925**	0.986683	0.987442	_	**80, 97, 130, 139, 168**	57, 93, 133, 173, 189	57, 104, 109, 141, 164
Man	2	**0.965404**	**0.965404**	0.963060	0.955111	**58, 134**	**58, 134**	56, 134	54, 133
	3	**0.978772**	**0.978772**	0.969947	0.974499	**37, 93, 161**	**37, 93, 161**	55, 108, 143	50, 115, 178
	4	_	**0.988956**	0.983349	0.985393	_	**44, 93, 144, 196**	33, 75, 102, 161	24, 67, 130, 197
	5	_	**0.993265**	0.992121	0.990749	_	**47, 82, 122, 158, 201**	36, 85, 128, 166, 196	37, 72, 94, 150, 202
Elephants	2	**0.943727**	**0.943727**	0.916594	0.911706	**70, 121**	**70, 121**	87, 110	60, 96
	3	**0.970600**	**0.970600**	0.960142	0.961880	**63, 105, 148**	**63, 105, 148**	58, 107, 126	60, 110, 156
	4	_	**0.982779**	0.968369	0.973117	_	**52, 86, 117, 148**	75, 120, 147, 178	56, 99, 122, 165
	5	_	**0.991580**	0.971074	0.977126	_	**54, 87, 113, 138, 163**	63, 112, 121, 158, 164	66, 101, 132, 166, 226

(_) La solution ne peut être calculée en un temps raisonable.

valeurs des fonctions objectives. En particulier, une valeur inférieure de l'écart-type (StD) indique une stabilité plus élevée, et une valeur élevée de la moyenne (Mean) de la fonction objective indique une meilleure précision.

TABLE 5.7: [Les résultats obtenus par l'algorithme de rechereche différentielle amélioré (IDSA) (Kotte, Kumar, and Injeti, 2018)

Test image	TH	PSNR	Thresholds	SSIM
Lena	2	16.3330	69, 121	0.5457
	3	18.4791	59, 105, 152	0.6094
	4	20.7278	32, 77, 116, 159	0.7152
	5	22.7919	31, 69, 101, 130, 172	0.7499
Cameraman	2	18.5141	96, 147	0.6255
	3	20.6560	84, 124, 158	0.6611
	4	22.3218	53, 100, 132, 160	0.6910
	5	23.6858	43, 92, 121, 148, 168	0.7165
Baboon	2	16.0609	79, 138	0.6436
	3	18.8248	54, 111, 162	0.7257
	4	20.4395	42, 87, 126, 172	0.7780
	5	22.0016	28, 68, 110, 144, 175	0.8286

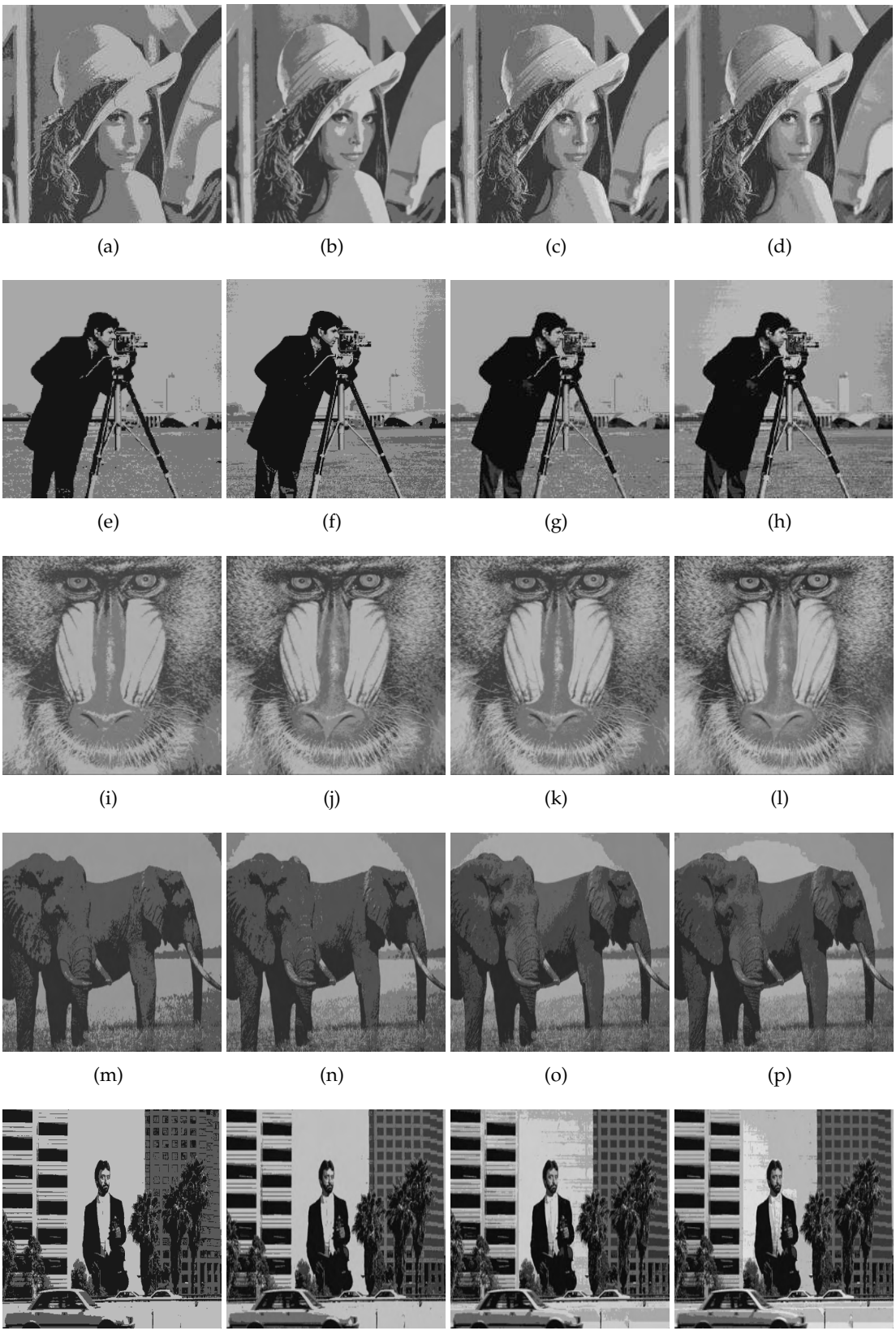
La moyenne des valeurs des fonctions objectives PSNR et SSIM sont données dans le Tableau 5.8, où nous pouvons observer que la méthode basée sur ***PSO*** a une moyenne plus élevée des fonctions objectives par rapport aux méthodes basées sur FA et BA. L'écart-type des algorithmes est donné dans le Tableau 5.9, qui montre que la méthode basée sur ***PSO*** a un écart-type plus faible que les méthodes basées sur FA et BA. Ce qui signifie que ***PSO*** est l'algorithme le plus stable et qui produit des seuils plus précis comparé aux autres algorithmes.

5.6.3 Performances des approches basées sur SSIM, PSNR et Otsu

En se basant sur la conclusion précédente, nous utiliserons dans ce qui suit l'algorithme ***PSO*** pour comparer les fonctions objectives : SSIM, PSNR et Otsu.

La qualité des images segmentées par les seuils obtenus est évaluée en utilisant les métriques PSNR et SSIM.

Les Tableaux 5.10 et 5.11 montrent les valeurs de PSNR et SSIM obtenues par l'algorithme PSO, pour les images segmentées (Lena, Cameraman, Baboon, Elephant et Man)



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

(e) (f) (g) (h)

(i) (j) (k) (l)

(m) (n) (o) (p)

(q) (r) (s) (t)

FIGURE 5.5: Images segmentées en utilisant les seuils obtenus par la méthode Otsu basée sur PSO, (a)-(q) Th= 2, (b)-(r) Th= 3, (c)-(s) Th= 4, (d)-(t) Th= 5.

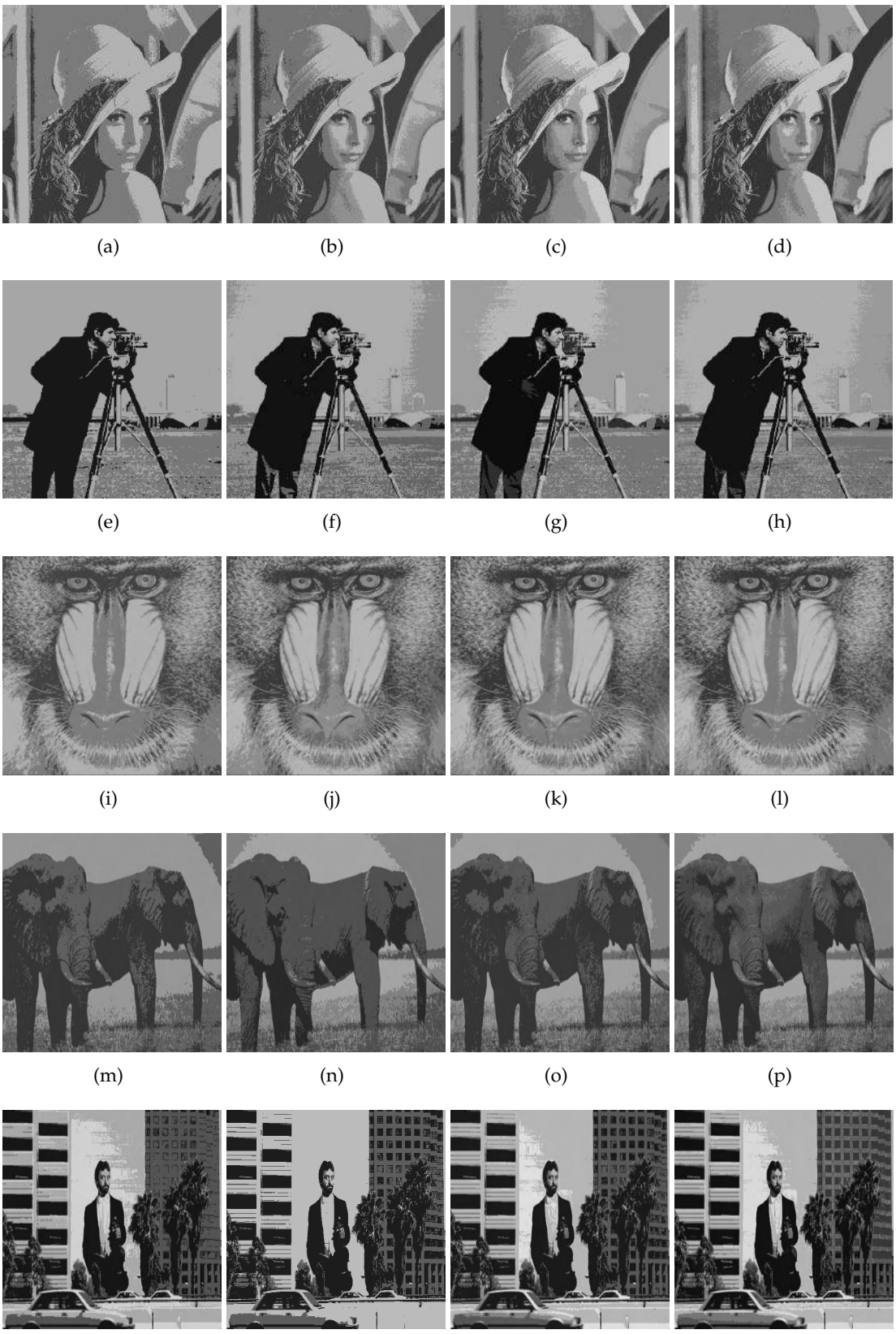
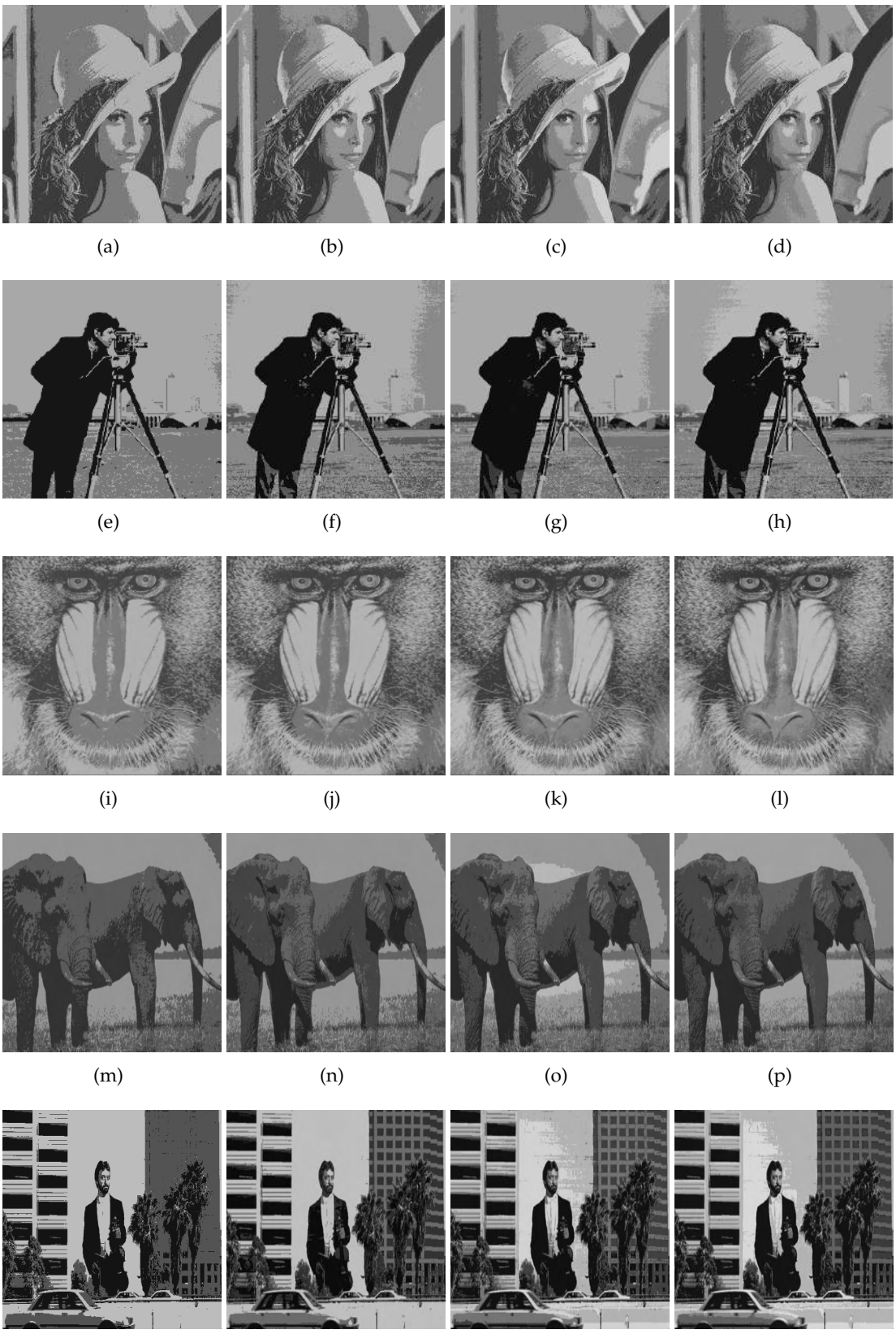


FIGURE 5.6: Images segmentées en utilisant les seuils obtenus par la méthode Otsu basée sur FA, (a)-(q) Th= 2, (b)-(r) Th= 3, (c)-(s) Th= 4, (d)-(t) Th= 5.



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

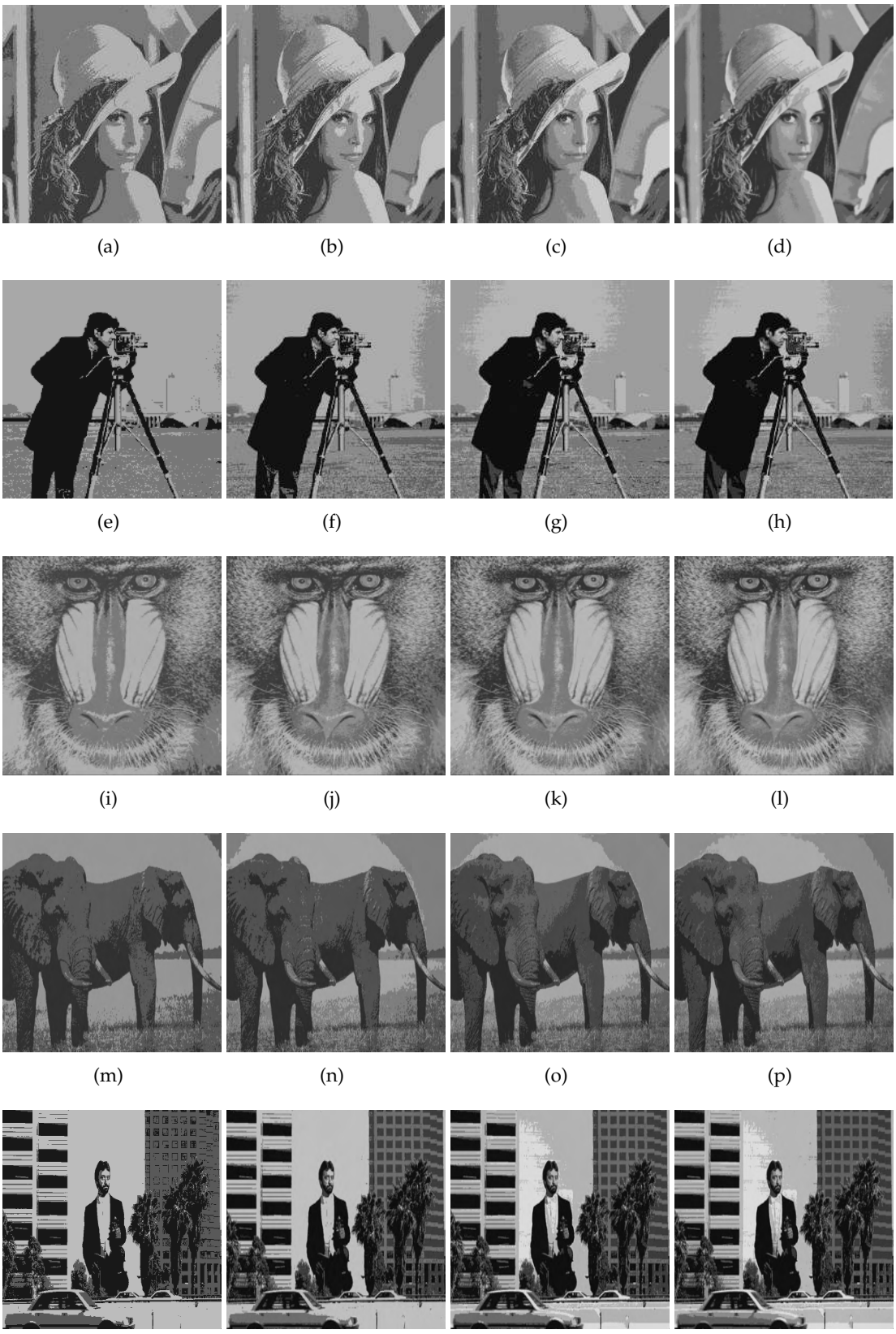
(e) (f) (g) (h)

(i) (j) (k) (l)

(m) (n) (o) (p)

(q) (r) (s) (t)

FIGURE 5.7: Images segmentées en utilisant les seuils obtenus par la méthode Otsu basée sur BA, (a)-(q) Th= 2, (b)-(r) Th= 3, (c)-(s) Th= 4, (d)-(t) Th= 5.



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

(e) (f) (g) (h)

(i) (j) (k) (l)

(m) (n) (o) (p)

(q) (r) (s) (t)

FIGURE 5.8: Images segmentées en utilisant les seuils obtenus par la méthode PSNR basée sur PSO, (a)-(q) Th= 2, (b)-(r) Th= 3, (c)-(s) Th= 4, (d)-(t) Th= 5.

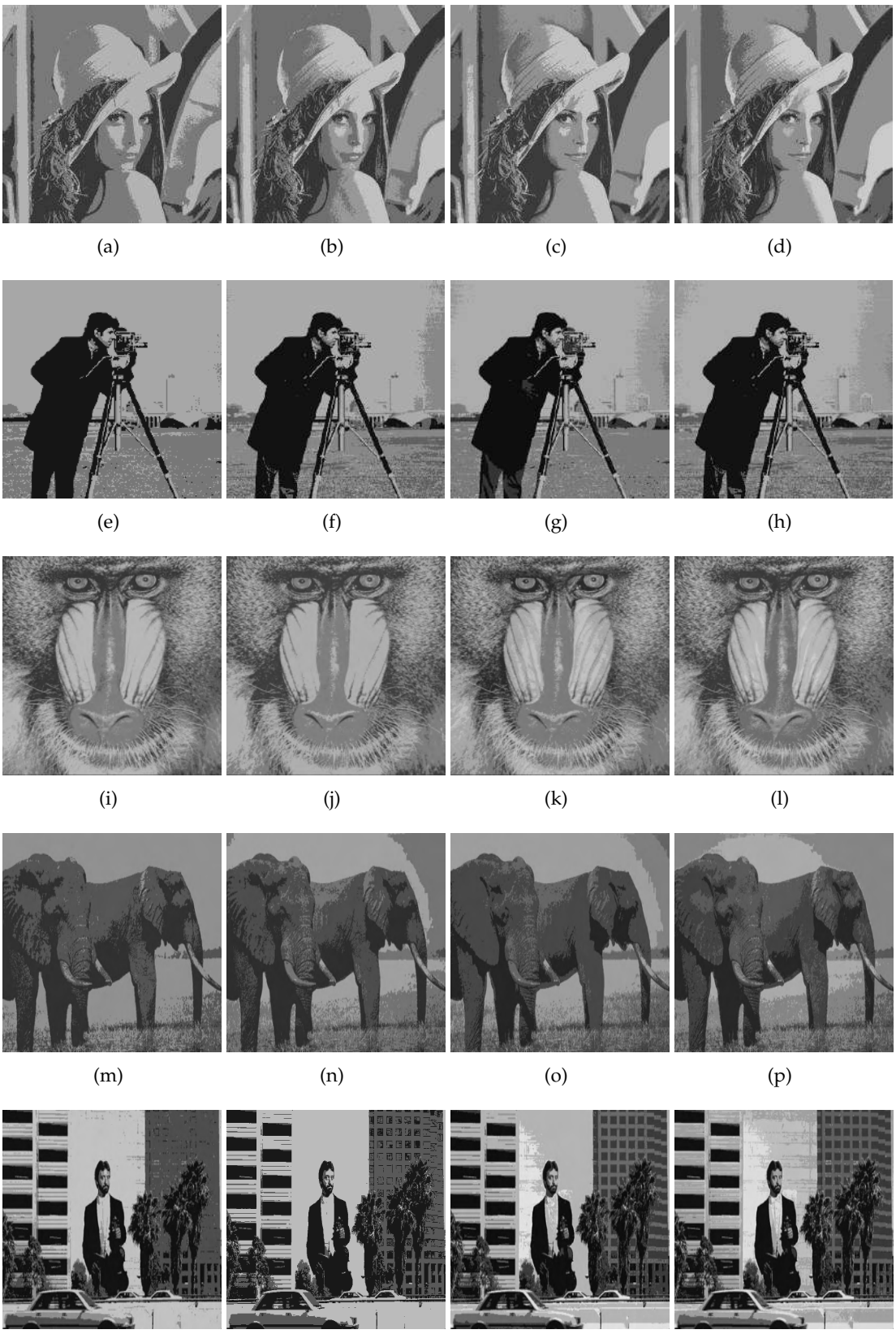


FIGURE 5.9: Images segmentées en utilisant les seuils obtenus par la méthode PSNR basée sur FA, (a)-(q) Th= 2, (b)-(r) Th= 3, (c)-(s) Th= 4, (d)-(t) Th= 5.

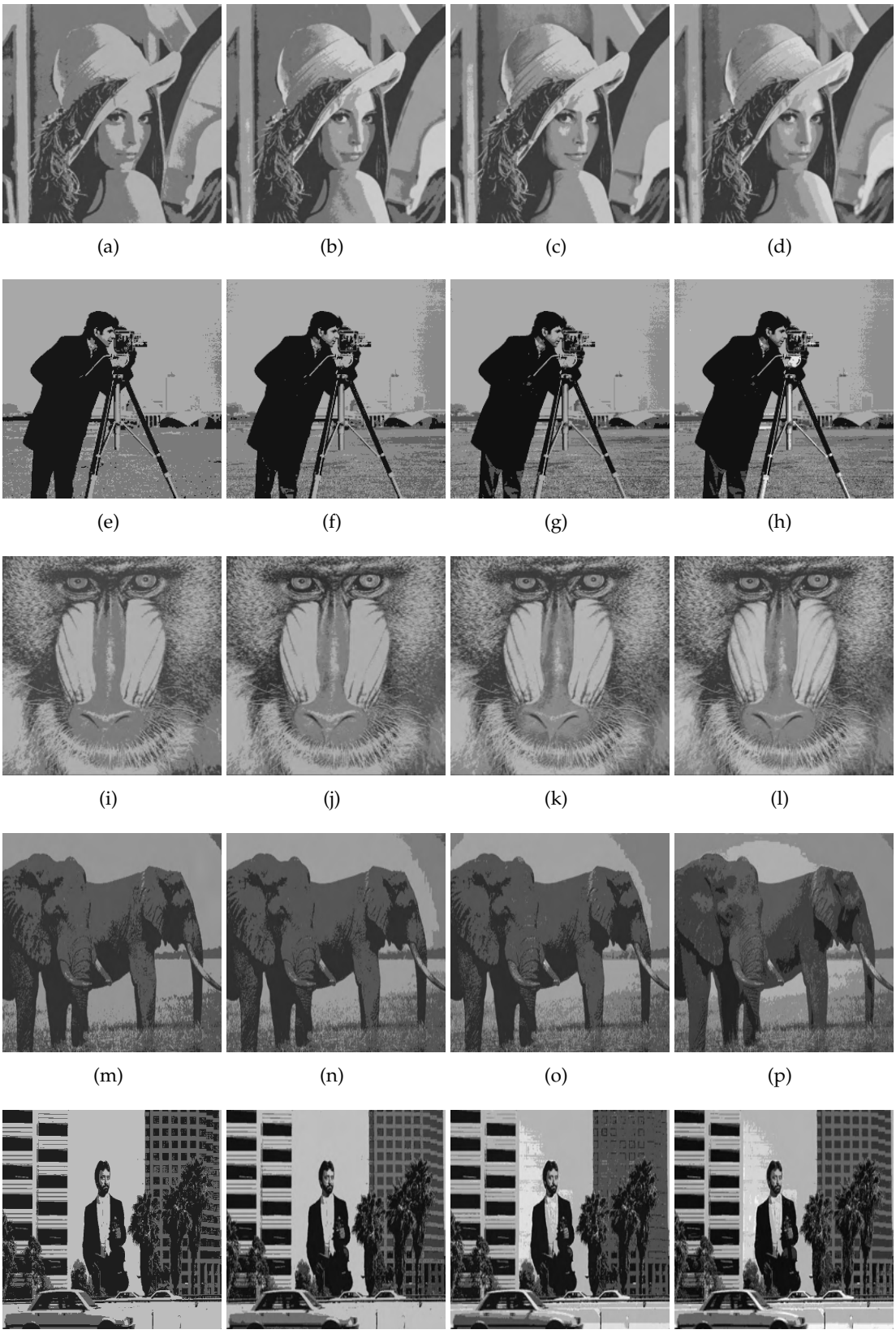


FIGURE 5.10: Images segmentées en utilisant les seuils obtenus par la méthode PSNR basée sur BA, (a)-(q) Th= 2, (b)-(r) Th= 3, (c)-(s) Th= 4, (d)-(t) Th= 5.

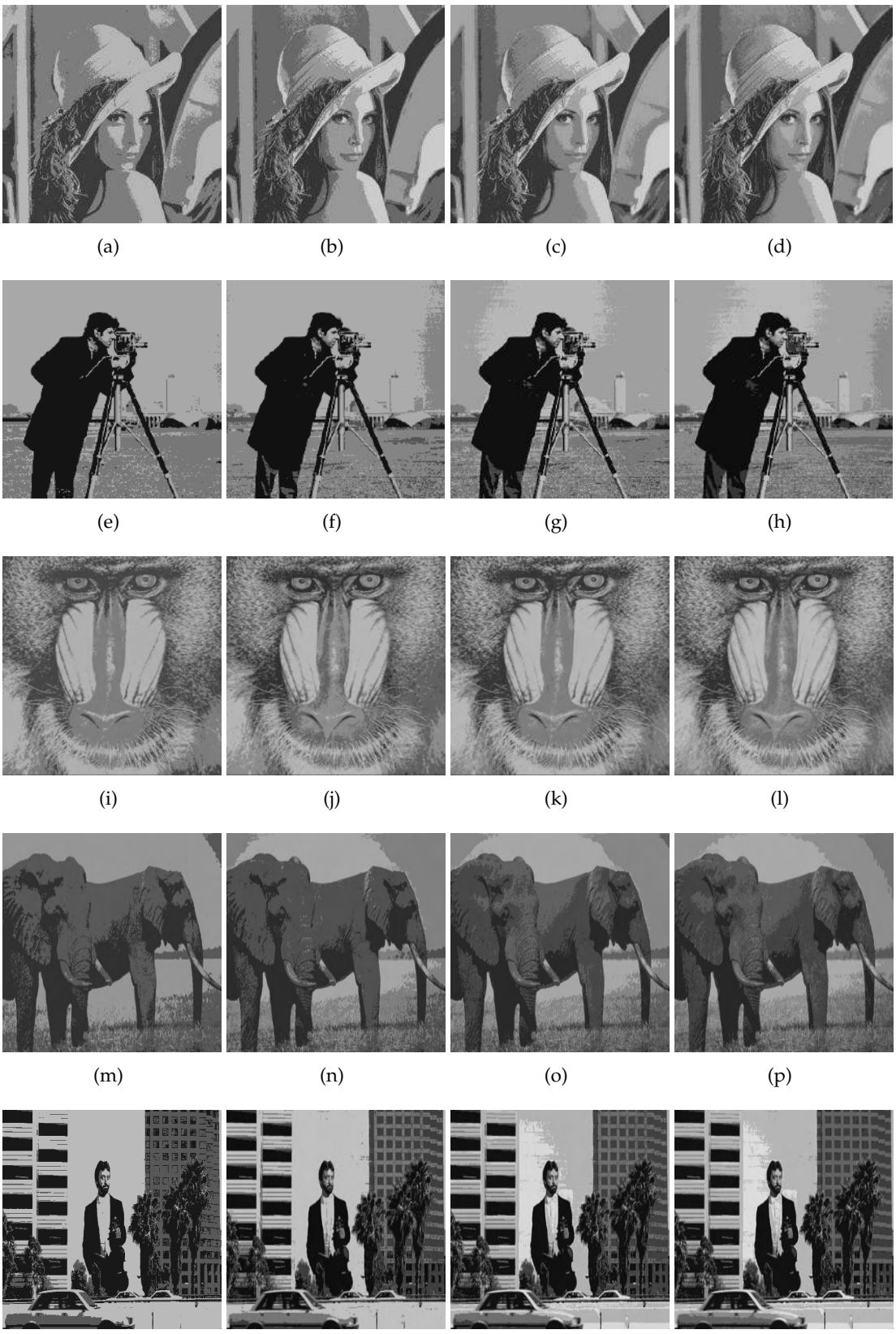
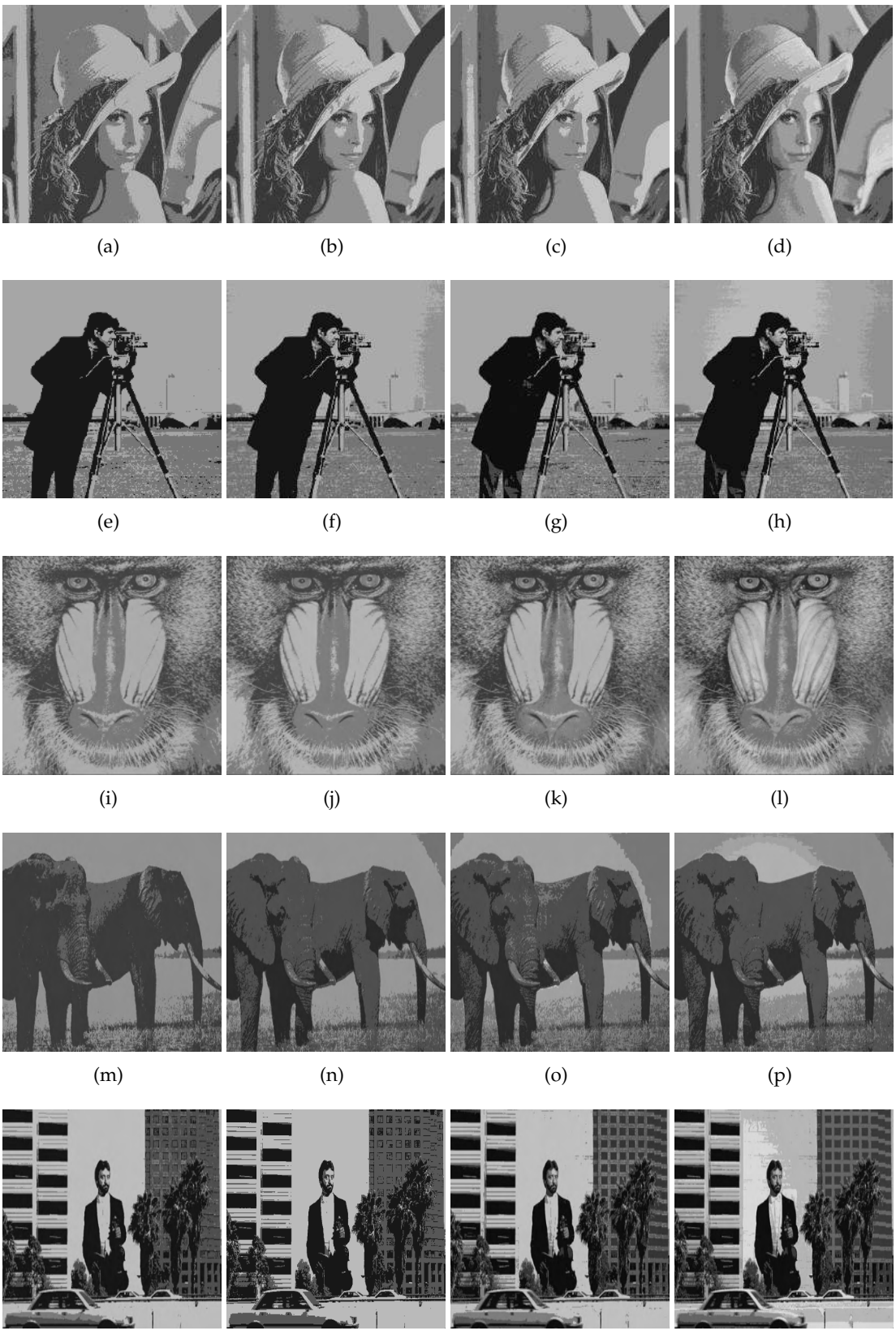


FIGURE 5.11: Images segmentées en utilisant les seuils obtenus par la méthode SSIM basée sur PSO, (a)-(q) Th= 2, (b)-(r) Th= 3, (c)-(s) Th= 4, (d)-(t) Th= 5.



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

(e) (f) (g) (h)

(i) (j) (k) (l)

(m) (n) (o) (p)

(q) (r) (s) (t)

FIGURE 5.12: Images segmentées en utilisant les seuils obtenus par la méthode SSIM basée sur FA, (a)-(q) Th= 2, (b)-(r) Th= 3, (c)-(s) Th= 4, (d)-(t) Th= 5.

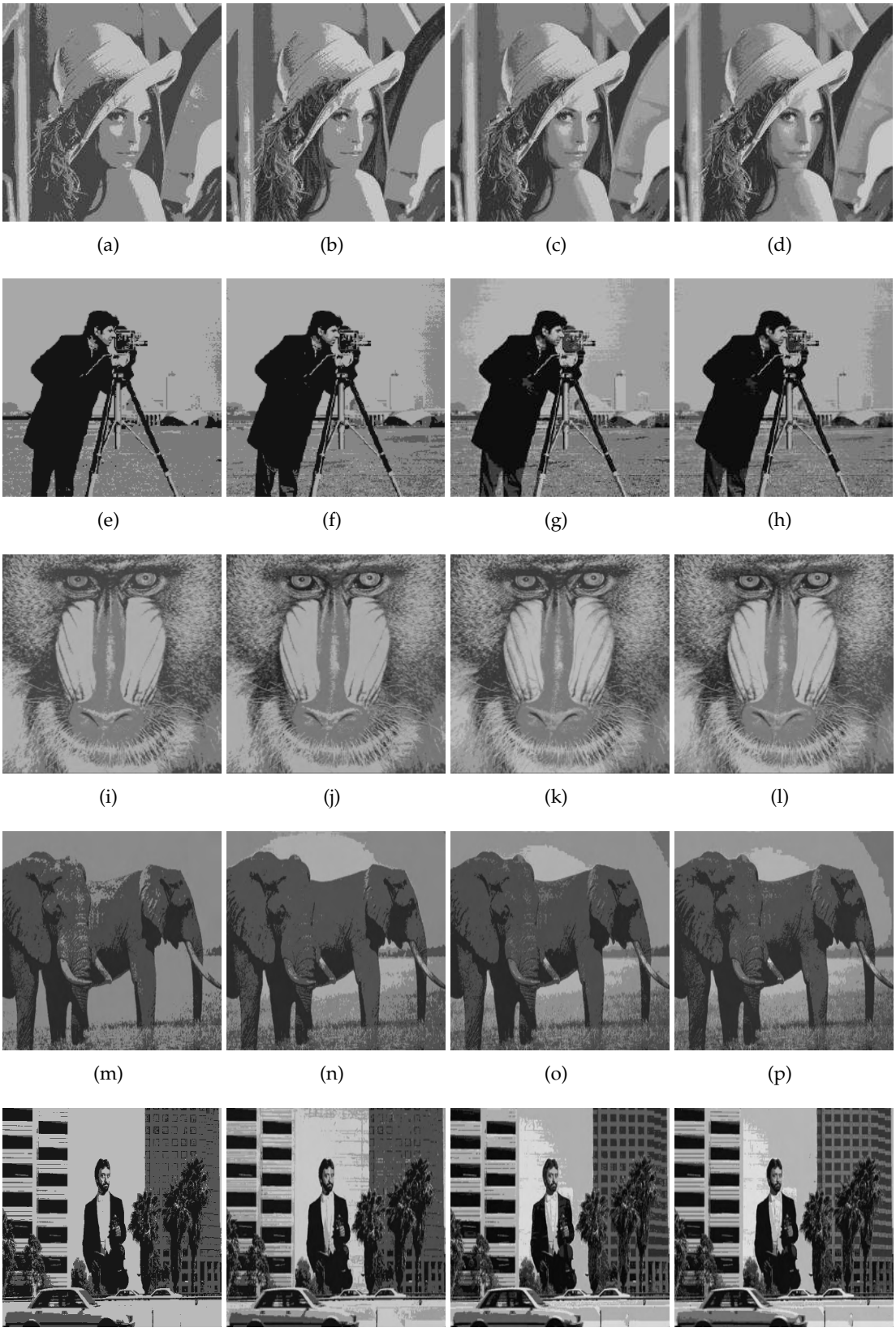


FIGURE 5.13: Images segmentées en utilisant les seuils obtenus par la méthode SSIM basée sur BA, (a)-(q) Th= 2, (b)-(r) Th= 3, (c)-(s) Th= 4, (d)-(t) Th= 5.

TABLE 5.8: Les valeurs de Mean des fonctions objectives PSNR et SSIM obtenues par les algorithmes PSO, FA et BA

		Fonction objective PSNR			Fonction objective SSIM		
Images de test	TH	PSO	FA	BA	PSO	FA	BA
Lena	2	**22,97**	22,82	22,89	**0,944753**	0,932601	0,938599
	3	**26,04**	25,29	25,59	**0,981988**	0,968078	0,970431
	4	**28,19**	26,65	27,06	**0,994832**	0,980758	0,980972
	5	**29,49**	28,06	28,05	**0,998110**	0,985730	0,987207
Cameraman	2	**24,40**	24,23	24,24	**0,972141**	0,968338	0,969033
	3	**26,07**	25,71	25,88	**0,982038**	0,978315	0,979081
	4	**27,88**	26,86	27,16	**0,989312**	0,984866	0,985090
	5	**29,41**	28,08	28,30	**0,993077**	0,988070	0,988952
Baboon	2	**24.31**	24.00	24.03	**0.959018**	0.941895	0.940292
	3	**26.34**	25.70	25.58	**0.981571**	0.964691	0.964446
	4	**28.25**	26.89	26.95	**0.994553**	0.976528	0.975448
	5	**29.56**	28.07	27.93	**0.999653**	0.983253	0.983338
Elephant	2	**24.64**	24.46	24.35	**0.943320**	0.935075	0.932925
	3	**26.96**	26.24	26.18	**0.971076**	0.958244	0.954836
	4	**29.03**	27.60	27.36	**0.984751**	0.970859	0.968674
	5	**30.27**	28.66	28.49	**0.989723**	0.978487	0.982146
Man	2	**21.96**	21.82	21.82	**0.964938**	0.961418	0.959769
	3	**23.89**	23.44	23.41	**0.978996**	0.974966	0.974175
	4	**26.14**	25.06	25.07	**0.989105**	0.983285	0.982662
	5	**27.67**	26.26	26.31	**0.993211**	0.987779	0.987309

en utilisant SSIM, PSNR et Otsu comme fonctions objectives. Nous pouvons remarquer que notre méthode généralisée basée sur **SSIM** donne une *meilleure* qualité d'image par rapport à PSNR et Otsu. Par exemple (Lena avec th = 3, SSIM = 0,98154), en utilisant la fonction objectif SSIM par rapport à Otsu (SSIM = 0,94664) et PSNR (SSIM = 0,97639). Cela est confirmé en utilisant l'ensemble des images de test (110 images), comme le montrent les Figures 5.14 et 5.15, où nous pouvons clairement remarquer que lorsque nous considérons *PSNR* comme une métrique de qualité, les résultats obtenus, en utilisant SSIM, PSNR et Otsu comme fonctions objectives, sont quasi-identiques. Mais, lorsque nous considérons *SSIM* comme une métrique de qualité, les résultats obtenus en l'utilisant comme fonction objective sont nettement meilleurs par rapport aux fonctions objectives PSNR et Otsu. Ceci confirme l'avantage d'utiliser **SSIM** comme mesure de qualité, comme indiqué pour la première fois dans (Wang et al., 2004).

TABLE 5.9: Les valeurs de StD des fonctions objectives PSNR et SSIM obtenues par les algorithmes PSO, FA et BA

		Fonction objective PSNR			Fonction objective SSIM		
Images de test	TH	PSO	FA	BA	PSO	FA	BA
Lena	2	**0**	0,180179	0,120916	**0,000936**	0,009503	0,004812
	3	**0,003042**	0,528541	0,391998	**0,001827**	0,009892	0,007396
	4	**0,004682**	0,623804	0,558027	**0,002581**	0,006826	0,005618
	5	**0,052787**	0,729292	0,61101	**0,001930**	0,006903	0,004498
Cameraman	2	**0**	0,193174	0,209822	**0,000126**	0,002861	0,004518
	3	**0,001672**	0,279028	0,570294	**0,000296**	0,002729	0,001853
	4	**0,004352**	0,413611	0,356242	**0,000398**	0,002227	0,001402
	5	**0,011042**	0,542404	0,384759	**0,000613**	0,002111	0,001916
Baboon	2	**0.000702**	0.372227	0.335012	**0.002009**	0.012953	0.012926
	3	**0.001760**	0.472491	0.566440	**0.002557**	0.008270	0.009158
	4	**0.007352**	0.655293	0.643191	**0.003488**	0.008182	0.007133
	5	**0.022726**	0.637013	0.632973	**0.003615**	0.006902	0.006555
Elephant	2	**0.000560**	0.233810	0.411830	**0.000735**	0.006132	0.009073
	3	**0.002376**	0.562967	0.553241	**0.001119**	0.007884	0.008428
	4	**0.006756**	0.727532	0.695293	**0.001401**	0.005999	0.006757
	5	**0.047774**	0.655797	0.689141	**0.001704**	0.005068	0.003236
Man	2	**0.000152**	0.205116	0.200234	**0.000588**	0.002543	0.004304
	3	**0.000339**	0.295730	0.339326	**0.000438**	0.002104	0.002142
	4	**0.001352**	0.524894	0.550813	**0.000426**	0.002545	0.002673
	5	**0.003337**	0.602808	0.534207	**0.000532**	0.002334	0.002152

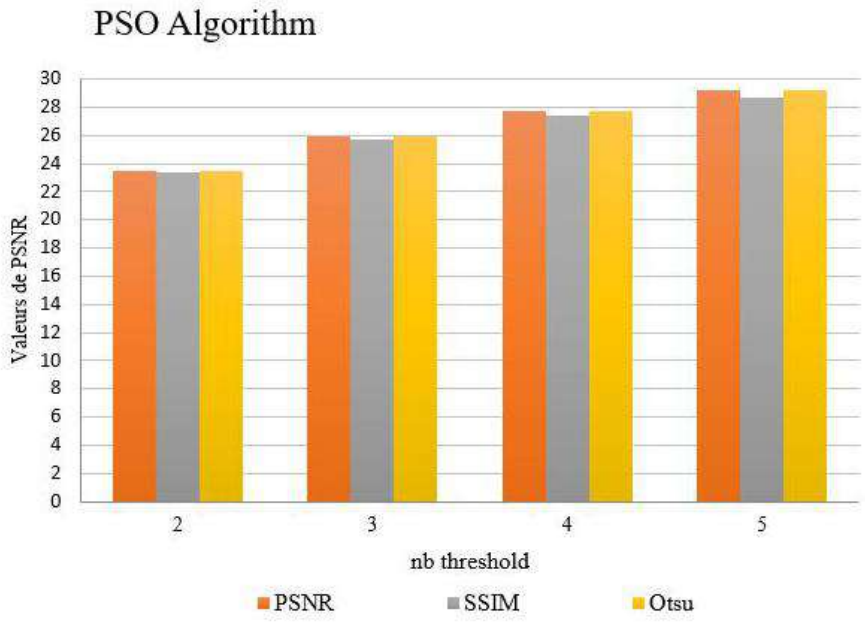


FIGURE 5.14: La moyenne de PSNR des 110 images.

TABLE 5.10: Comparaison des valeurs de SSIM en utilisant l'algorithme PSO

Images de test	Th	Otsu	SSIM	PSNR
Lena	2	0.92563	**0.94485**	0.92627
	3	0.94664	**0.98489**	0.97639
	4	0.98055	**0.99329**	0.98797
	5	0.98463	**0.99916**	0.98786
Cameraman	2	0.97055	**0.97219**	0.97055
	3	0.98079	**0.98222**	0.97972
	4	0.98671	**0.98939**	0.98710
	5	0.99253	**0.99398**	0.99114
Baboon	2	0.91617	**0.96058**	0.91642
	3	0.96106	**0.98227**	0.96144
	4	0.96752	**0.99830**	0.96412
	5	0.99316	**0.99893**	0.97935
Elephant	2	0.93366	**0.94373**	0.93393
	3	0.96249	**0.97060**	0.96436
	4	0.97173	**0.98278**	0.97798
	5	0.97840	**0.99158**	0.98612
Man	2	0.95938	**0.96540**	0.95907
	3	0.97365	**0.97877**	0.97346
	4	0.98742	**0.98895**	0.98789
	5	0.99162	**0.99326**	0.98783

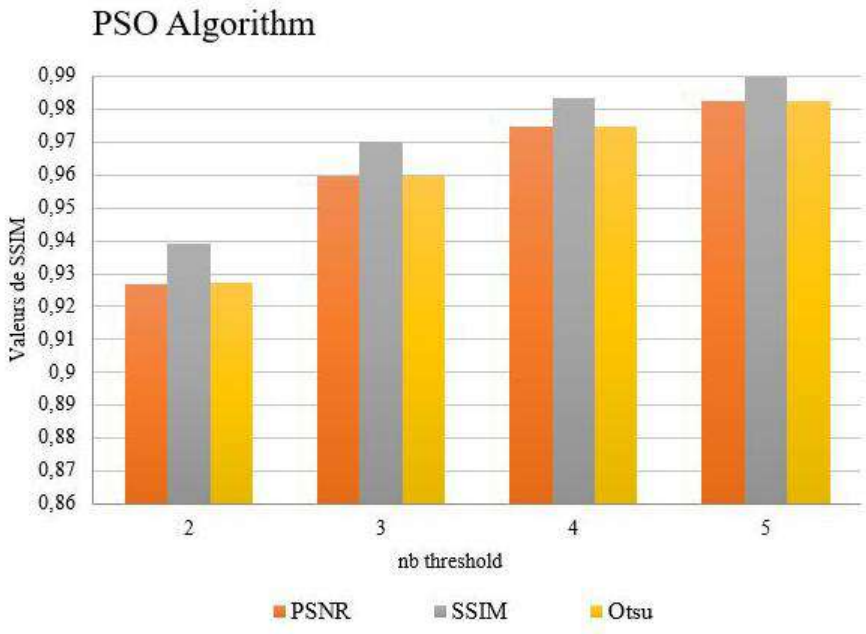


FIGURE 5.15: La moyenne de SSIM des 110 images.

TABLE 5.11: Comparaison des valeurs de PSNR en utilisant l'algorithme PSO

Test images	Th	Otsu	SSIM	PSNR
Lena	2	**22.96**	22.90	**22.96**
	3	26.03	25.87	**26.04**
	4	28.17	27.75	**28.19**
	5	29.49	28.88	**29.52**
Cameraman	2	24.39	24.39	**24.40**
	3	**26.06**	25.98	**26.06**
	4	**27.88**	27.82	**27.88**
	5	29.37	29.31	**29.41**
Baboon	2	21.30	24.29	**24.31**
	3	26.32	25.76	**26.34**
	4	28.23	27.86	**28.25**
	5	**29.56**	28.24	29.55
Elephant	2	24.63	24.63	**24.64**
	3	**26.96**	26.90	**26.96**
	4	29.01	28.81	**29.03**
	5	30.13	29.88	**30.29**
Man	2	**21.96**	21.92	**21.96**
	3	23.88	23.70	**23.90**
	4	**26.14**	25.96	**26.14**
	5	**27.65**	27.33	**27.65**

5.6.4 Comparaisons de temps de calcul

Le Tableau 5.12 montre une comparaison en termes de temps d'exécution, pour les images (Lena , Cameraman, Baboon, Elephant et Man) entre les algorithmes PSO, FA et BA utilisant Otsu, SSIM et PSNR comme fonctions objectives. D'une part, nous pouvons voir que la méthode Otsu est nettement plus rapide que les fonctions SSIM et PSNR, et que SSIM est environ deux fois plus lente que PSNR.

D'un autre côté, on peut remarquer que notre implémentation de PSO est légèrement plus lente que celle des algorithmes FA et BA. La comparaison du temps de calcul pour l'ensemble des données d'image (110 images) est présentée dans le Tableau 5.13. Encore une fois, nous pouvons remarquer que PSO est légèrement plus lent que les algorithmes FA et Bat, et que SSIM est environ deux fois plus lent que PSNR. Mais comme nous l'avons conclu dans les expériences précédentes, la combinaison de

l'algorithme PSO avec SSIM comme fonction objective conduit aux meilleurs résultats. Par conséquent, il est judicieux d'envisager notre méthode généralisée basée sur le SSIM en utilisant l'algorithme PSO avec nos paramètres réglés pour le problème du multi-seuillage.

TABLE 5.12: Comparaison du temps d'éxecution des differentes méthodes en utilisant PSO, FF et Bat (en seconde)

		Otsu			SSIM			PSNR		
Images de test	TH	PSO	FA	BA	PSO	FA	BA	PSO	FA	BA
Lena	2	0.039	0.028	0.033	8.518	8.517	8.283	3.713	3.526	3.323
	3	0.040	0.031	0.036	9.375	9.063	8.955	4.508	4.368	3.775
	4	0.041	0.040	0.040	10.015	9.422	9.422	5.304	4.446	4.711
	5	0.062	0.058	0.056	10.702	10.015	9.921	5.897	4.665	5.413
Cameraman	2	0.012	0.009	0.005	2.122	2.121	2.075	0.905	0.889	0.858
	3	0.020	0.012	0.009	2.340	2.153	2.200	1.138	1.061	0.967
	4	0.016	0.014	0.012	2.621	2.309	2.371	1.341	1.108	1.108
	5	0.020	0.018	0.015	2.761	2.465	2.542	1.560	1.248	1.217
Baboon	2	0.015	0.012	0.010	8.126	7.846	7.658	3.305	3.040	2.526
	3	0.039	0.019	0.016	9.015	8.657	8.110	4.444	3.695	3.539
	4	0.042	0.021	0.019	10.201	9.589	8.891	5.302	4.132	4.023
	5	0.042	0.019	0.015	11.184	9.717	9.984	6.379	4.912	4.944
Elephant	2	0.016	0.012	0.010	3.868	3.867	3.820	1.184	1.137	1.028
	3	0.021	0.016	0.013	4.413	4.133	4.148	1.574	1.209	2.137
	4	0.021	0.016	0.015	4.803	5.213	4.663	2.011	1.543	1.402
	5	0.023	0.019	0.017	5.256	5.866	4.725	2.479	2.195	1.761
Man	2	0.013	0.010	0.011	4.008	3.930	3.883	1.153	1.150	1.044
	3	0.022	0.016	0.013	4.428	4.226	4.304	1.527	1.318	1.450
	4	0.021	0.017	0.015	4.975	4?874	4.616	2.058	1.559	1.621
	5	0.024	0.020	0.019	5.240	5.600	4.913	2.394	2.027	2.010

TABLE 5.13: Comparaison du temps d'éxecution des differents algorithme pour les 110 image en utilisant l'implémentation séquentielle (en secondes).

	Fonction objective PSNR			Fonction objective SSIM		
TH	PSO	FA	BA	PSO	FA	BA
2	250,778	245,307	230,375	579,455	572,888	560,002
3	303,864	282,912	269,264	631,776	610,622	599,703
4	360,789	311,506	307,997	685,317	642,96	636,799
5	405,339	345,363	342,647	731,507	671,726	681,76

Le temps de calcul supplémentaire induit par cette méthode peut facilement être compensé en adoptant la version parallèle de notre algorithme. En effet, nous avons

implémenté tous les algorithmes discutés ci-dessus en parallèle en utilisant la bibliothèque OpenMP. Les temps d'exécution obtenus pour les différents algorithmes de l'ensemble des images de test sont résumés dans le Tableau 5.14 où nous pouvons voir qu'avec seulement quatre cœurs de processeur, nous avons gagné une accélération entre 2,42 et 3,10 pour tous les algorithmes mis en œuvre, comme indiqué dans le Tableau 5.15. Par exemple, l'accélération de notre méthode basée sur SSIM généralisée en utilisant l'algorithme PSO avec des paramètres ajustés est de **2,98** (avec th = 5). Il est à noter que l'accélération peut être améliorée en utilisant une machine plus puissante ayant un nombre de processeurs plus élevé.

TABLE 5.14: Comparaison du temps d'éxecution des differents algorithme pour les 110 image en utilisant l'implémentation parallèle (en secondes).

	Fonction objective PSNR			Fonction objective SSIM		
TH	PSO	FA	BA	PSO	FA	BA
2	90,547	87,631	86,264	194,713	191,673	187,955
3	106,221	95,796	91,385	214,324	200,116	193,249
4	128,225	104,721	100,221	240,727	217,45	221,727
5	142,949	113,028	118,544	245,388	235,097	233,637

TABLE 5.15: L'accelération (SpeedUp obtenue par l'implémentation parallèlé.

	Fonction objective PSNR			Fonction objective SSIM		
TH	PSO	FA	BA	PSO	FA	BA
2	2,7695	2,7993	2,6705	2,9759	2,9888	2,9794
3	2,8606	2,6634	2,9464	2,9477	3,0513	3,1032
4	2,8137	2,4293	3,0731	2,8468	2,9568	2,8719
5	2,8355	2,3969	2,8904	2,9810	2,8572	2,9180

5.7 Conclusion

Le multi-seuillage est une technique utilisée à large échelle dans la segmentation d'image. Le succès ou l'échec de l'analyse et de l'interprétation des images dépendent de la précision de la technique de segmentation utilisée.

Dans notre travail, nous avons généralisé l'utilisation de la mesure de qualité SSIM pour résoudre le problème de multi-seuillage afin d'obtenir les meilleures performances par rapport aux méthodes utilisant PSNR et Otsu comme fonctions objectives. Nous avons mis en œuvre l'algorithme de la recherche exhaustive en utilisant

les fonctions objectives SSIM, PSNR et Otsu pour obtenir les meilleures solutions exactes, lorsque cela est possible. Par la suite, nous avons sélectionné trois algorithmes PSO, FA et BA bien connus et fréquemment utilisés, et nous avons ajusté empiriquement leurs paramètres afin de produire des solutions quasi-identiques aux solutions exactes. Tous ces algorithmes ont également été mis en œuvre en parallèle à l'aide d'OpenMP.

L'étude expérimentale réalisée sur 110 images a montré que l'approche SSIM donne de meilleurs seuils par rapport aux autres techniques de seuillage (PSNR et Otsu). De plus, l'algorithme PSO s'avère plus précis que FA et BA. Les seuils produits sont quasiment les mêmes ou très proches que ceux obtenus par la recherche exhaustive. Par ailleurs, nous avons montré que l'utilisation du parallélisme améliore considérablement le temps de calcul des différents algorithmes, ce qui permet d'obtenir des résultats plus précis dans des délais tout à fait raisonnables.

Chapitre 6

Algorithme BFO amélioré UBFO

6.1 Introduction

Dans ce chapitre, nous apportons des modifications à l'algorithme BFO (Bacterial Foraging Optimizaiton), qui a été développé pour la première fois par (Passino, 2002) dans le but d'améliorer sa façon d'explorer et d'exploiter l'espace de recherche pour qu'il puisse converger rapidement tout en évitant d'être piégé dans les optimums locaux.

Afin d'évaluer les performances de notre version d'algorithme, nommé ***UBFO*** « Upgraded BFO », nous l'avons testé en premier lieu sur 21 fonctions benchmark, comparé les résultats obtenus avec ceux obtenus par les algorithmes métaheuristiques les plus connus à savoir : PSO (Kennedy and Eberhart, 1995; Shi and Eberhart, 1999), WOA (Mirjalili and Lewis, 2016), GWO (Mirjalili, Mirjalili, and Lewis, 2014) , MFO (Mirjalili, 2015), BFO (Passino, 2002) et MBFO (Sathya and Kayalvizhi, 2011b). L'évaluation de la performance de notre algorithme a été basé sur la comparaison des valeurs des fonctions benchmarks obtenues, des tests statistiques pour étudier sa précision, sa stabilité et son temps d'exécution. En second lieu, pour approuver son éfficacité dans la résolution des problèmes du monde réel, nous l'avons appliqué au traitement d'image, précisément la segmentation des images en nivaux de gris.

Nos résultats expérimentaux montrent que l'algorithme *UBFO* est plus performant que les algorithmes BFO, MBFO, MFO. En revanche, il produit des résultats compétitifs comparé aux algorihtmes PSO et WOA.

6.2 L'approche proposée

L'algorithme d'optimisation BFO, introduit en 2002 par Passino, modélise le comportement individuel et collectif des bactéries du type Escerichia Coli, qui vivent couramment dans les intestins de l'être humain (Passino, 2002). Comme il a été décrit en section 2.3.4, l'algorithme BFO est construit de quatre phases essentielles : la chimiotaxie, l'essaimage, la reproduction et l'élimination-dispersion. Dans notre travail, nous avons apporté des modifications dans les phases chimiotaxie et élimination-dispersion.

1. ***La chimiotaxie :*** C'est la phase qui permet d'améliorer les valeurs des positions initiales dans le but d'une meilleure convergence, son calcul est donné par l'équation (2.6). Cependant, cette phase est étroitement liée à la valeur du paramètre consacré au pas ajouté à chaque position (run lengh) C_i $(i = 0, 1, \ldots, S-1)$, qui influence le déplacement des bactéries dans l'espace de recherche. Sachant qu'une petite valeur de C_i favorise l'exploitation et néglige l'exploration, l'inverse se produit en lui attribuant une grande valeur.

   Afin de remédier à ce problème nous avons proposé d'utiliser une valeur de C_i adaptatif aux nombres d'itération (Eq 6.1). L'idée est de commencer avec une valeur élevée favorisant la diversification et qui diminue au cours des itérations de la phase chimiotaxie (N_c) pour favoriser l'intensification.

$$C_i = C_{max} - \left((C_{max} - C_{min}) \times \frac{j}{N_c} \right), \quad j = 0, 1, \ldots, N_c \tag{6.1}$$

   Tel que : j représente l'étape chimiotaxie courante et N_c est le nombre maximal d'étapes chimiotaxie.

2. ***L'élimination-dispersion :*** elle a un fort impact sur la convergence de l'algorithme, et permet de renouveler la population pour explorer le maximum de zones dans l'espace de recherche. Néanmoins, cette phase dépend de la valeur attribuée au paramètre P_{ed}. Rappelons que si la valeur de P_{ed} est supérieure à une valeur générée aléatoirement comprise entre 0 et 1, alors une nouvelle position (bactérie) est attribuée à la population, tout en éliminant une déjà existante peu importe la qualité de la solution qu'elle produit. Autrement dit, une très grande valeur de P_{ed} peut conduire l'algorithme à une recherche exhaustive-aléatoire. Pour cela, nous avons proposé qu'au lieu d'utiliser une valeur fixe pour toute la

population, de calculer P_{ed} de la manière suivante :

$$P_{ed} = \frac{i}{S}, \quad i = 0, 1, \ldots, S-1 \tag{6.2}$$

L'idée est de diminuer la probabilité de perdre les meilleures positions et de favoriser l'élimination de celles les moins performantes. Notons que, la probabilité d'éliminer la meilleure position (solution optimale) est nulle, ce qui lui permet de faire toujours partie de la population et de s'améliorer durant les prochaines itérations.

L'algorithme 12 résume le fonctionnement de notre algorihtme *UBFO*.

6.3 Résultats et Discussion

Pour valider un algorithme d'optimisation, plusieurs fonctions mathématiques ont été proposées dans la littérature d'optimisation (Digalakis and Margaritis, 2001; Yang, 2010a), l'optimum global de ces fonctions est connu d'avance.

Dans notre cas, nous avons utilisé 21 fonctions parmi celles les plus performantes dans l'évaluation des algorithmes d'optimisation (Hussain et al., 2017; Li et al., 2013; Jamil and Yang, 2013). Ces fonctions sont divisées en deux groupes : Unimodale et multimodale. Le premier type permet d'évaluer la capacité de l'algorithme à exploiter l'espace de recherche, car ces fonctions se caractérisent par un seul optimum global et aucun optimum local contrairement aux fonctions multimodales. Effectivement, ces dernières sont efficaces pour étudier l'exploration de l'algorithme ainsi que sa capacité d'échapper aux optimums locaux. Notons que parmi ces 21 fonctions il y en a celles qui sont de dimensions fixes (f_8-f_{12},f_{16},f_{18},f_{20} et f_{21}).

Les formules mathématiques des fonctions tests utilisées sont données dans la table 6.1.

Les résultats de notre algorithme *UBFO*, ont été comparés avec ceux obtenus par les algorithmes : PSO (Kennedy and Eberhart, 1995), WOA (Mirjalili and Lewis, 2016), MFO (Mirjalili, 2015), GWO (Mirjalili, Mirjalili, and Lewis, 2014), BFO (Passino, 2002), MBFO (Sathya and Kayalvizhi, 2011b). Chacun de ces algorithmes a été exécuté 100

Algorithm 12 Algorithme UBFO *Upgraded-BFO*

Initialiser les paramètres : $D, S, N_c, N_s, N_{re}, N_{ed}, p_{ed}, C_{min} = 0.2, C_{max} = 0.8, \theta^i (i = 0, 1, \ldots, S-1)$
pour $(l = 0, 1, \ldots, N_{ed} - 1)$ (Etapes d'élimination-disperssion) **faire**
  pour $(k = 0, 1, \ldots, N_{re} - 1)$ (Etapes de reproduction) **faire**
    pour $(j = 0, 1, \ldots, N_c)$ (Etapes chimiotactiques) **faire**
      pour toute bactérie $(i = 0, 1, \ldots, S-1)$ **faire**
        Evaluation de la fonction objective $J(i, j, k, l)$ *par l'Eq (2.7)*
        $J_{last} = J(i, j, k, l)$
        Pivotement (Tumble):Génératon aléatoire du vecteur $\Delta(i) \in \Re^D$
        Calculer: $C_i = C_{max} - \left((C_{max} - C_{min}) \times \frac{j}{N_c} \right)$
        Déplacement: Calculer la position $\theta^i(j+1, k, l)$

$$\theta^i(j+1,k,l) = \theta^i(j,k,l) + C(i)\phi(j),$$
$$\phi(j) = \frac{\Delta(i)}{\sqrt{\Delta^T \cdot \Delta(i)}}$$

        Evaluation de la fonction objective $J(i, j+1, k, l)$ *par l'Eq (2.7)*
        Nage: $m = 0$
        tant que $(m < N_s)$ **faire**
          $m = m + 1$
          si($J(i, j+1, k, l) < J_{\text{last}}$ **) alors**
            $J_{\text{last}} = J(i, j+1, k, l)$
            Déplacement: Calcule de la position $\theta^i(j+1, k, l)$

$$\theta^i(j+1,k,l) = \theta^i(j,k,l) + C(i)\phi(j),$$
$$\phi(j) = \frac{\Delta(i)}{\sqrt{\Delta^T \cdot \Delta(i)}}$$

            Evaluation de la fonction objective $J(i, j+1, k, l)$ *par l'Eq (2.7)*
          sinon
            $m = N_s$
          fin si
        fin tant que
      fin pour
    fin pour
    pour $(i = 0, 1, \ldots, S-1)$ **faire**
      Calcul de : $J^i_{\text{health}} = \sum_{j=1}^{N_c+1} J(i, j, k, l)$
    fin pour
    Trier les bactéries par ordre croissant de J_{health}
  fin pour
  pour $(i = 0, 1, \ldots, S-1)$ **faire**
    Calculer: $P_{ed} = \frac{i}{S}$
    si $P_{ed} > rand(0, 1)$ **alors**
      Eliminer la i^{me} *bactérie, et générer une nouvelle position aléatoirement.*
  fin pour
fin pour

TABLE 6.1: Détails sur les fonctions tests utilisées.

Num	Formule	Nom	ER	Type	$f(\mathbf{x}^*)$
f_1	$f(\mathbf{x}) = \sum_{i=1}^{D} x_i^2$	Sphere	[-100,100]	M	0
f_2	$f(\mathbf{x}) = \sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	Schwefel 2.22	[-100,100]	U	0
f_3	$f(x) = \sum_{i=1}^{D} (x_i + 0.5)^2$	Step	[-100,100]	U	0
f_4	$f(\mathbf{x}) = \sum_{i=1}^{D} i x_i^4 + \text{random}[0,1)$	Quartic	[-1.28,1.28]	M	0
f_5	$f(\mathbf{x}) = \frac{1}{2} \sum_{i=1}^{D} \left(x_i^4 - 16x_i^2 + 5x_i\right)$	Styblinski-Tang	[-5,5]	M	-39.16599D
f_6	$f(\mathbf{x}) = -20 \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + e$	Ackley	[-32,32]	M	0
f_7	$f(x) = \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	Griewank	[-600,600]	M	0
f_8	$f(\mathbf{x}) = (1.5 - x_1 + x_1 x_2)^2 + \left(2.25 - x_1 + x_1 x_2^2\right)^2 + \left(2.625 - x_1 + x_1 x_2^3\right)^2$	Beale	[-4.5,4.5]	M	0
f_9	$f(\mathbf{x}) = 100\left(x_1^2 - x_2\right)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90\left(x_3^2 - x_4\right)^2 + 10 \cdot 1\left((x_2 - 1)^2 + (x_4 - 1)^2\right) + 19.8(x_2 - 1)(x_4 - 1)$	Colville	[-10,10]	M	0
f_{10}	$f(x,y) = 0.5 + \frac{\sin^2\left(x^2+y^2\right)^2 - 0.5}{(1+0.001(x^2+y^2))^2}$	Schaffer	[-100,100]	U	0
f_{11}	$f(X) = -e^{\lvert \cos(x_1)\cos(x_2) e^{\lvert 1 - \frac{\sqrt{x_1^2+x_2^2}}{\pi} \rvert} \rvert^{-1}}$	Pen holder	[-11,11]	M	0.9635348
f_{12}	$f(x,y) = \left[1 + (x+y+1)^2 (19 - 14x + 3x^2 - 14y + 6xy + 3y^2)\right]\left[30 + (2x-3y)^2 (18 - 32x + 12x^2 + 4y - 36xy + 27y^2)\right]$	Goldstein-Price	[-2,2]	M	3
f_{13}	$f(\mathbf{x}) = \sum_{i=1}^{D-1}\left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right]$	Rosenbrock	[-5,10]	U	0
f_{14}	$f(x) = \sum_{i=1}^{D} i x_i^2$	SumSquare	[-10,10]	U	0
f_{15}	$f(x) = \frac{\pi}{D}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{D-1}(y_i - 1)^2\left[1 + 10\sin^2(\pi y_{i+1})\right] + (y_D - 1)^2\right\} + \sum_{i=1}^{D} u_i$ $u_i = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & a \le x_i \le a \\ k(-x_i - a)^m & x_i < -a \end{cases}$ $y = 1 + (x_i + 1)/4$	Penalized1	[-50,50]	M	0
f_{16}	$f(x,y) = \left(x^2 + y - 11\right)^2 + \left(x + y^2 - 7\right)^2$	Himmelblau	[-6,6]	M	0
f_{17}	$f(x,y) = (x + 2y - 7)^2 + (2x + y - 5)^2$	Booth	[-10,10]	U	0
f_{18}	$f(x,y,z) = \frac{4}{3}\left(x^2 + y^2 - xy\right)^{0.75} + z$	Wolfe	[0,2]	M	0
f_{19}	$f(x) = \sum_{i=1}^{n} x_i^{10}$	Schwefel 2.23	[-10,10]	U	0
f_{20}	$f(\mathbf{x}) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right)x_1^2 + x_1 x_2 + \left(-4 + 4x_2^2\right)x_2^2$	SIX-HUMP CAMEL	[-5,5]	U	-1.0316
f_{21}	$f(\mathbf{x}) = -\cos(x_1)\cos(x_2)\exp\left(-(x_1 - \pi)^2 - (x_2 - \pi)^2\right)$	Easom	[-100,100]	U	-1

U:Unimodale, M: Multimodale, ER: Espace de Recherche, $f(\mathbf{x}^*)$ la valeur de l'optimum global.

fois pour chaque fonction, de manière indépendante en utilisant à chaque exécution une nouvelle génération de population. Ceci permet d'effectuer les tests statistiques (la moyenne de la fonction objective et l'écart-type correspondant) qui nous permettent d'étudier la stabilité et la précision de ces algorithmes. Cependant, ces tests dit paramétriques n'effectuent qu'une comparaison globale des performances des algorithmes, c'est pour cela que nous avons utilisé un second test statistique non-paramétrique appelé Wilcoxon. La valeur de p calculée est donnée dans les tables 6.4 6.11 6.12 et 6.13, pour les fonctions de dimension fixe, les fonctions avec une dimension égale à 10, 30 et 50 respectivement.

6.3.1 Evaluation de la qualité des solutions

Fonctions benchmark de dimension fixe : Les tableaux 6.2 et 6.3 montrent que l'algorithme *UBFO* produit de meilleurs résultats par rapport aux algorithmes BFO de base, MBFO et MFO, avec une différence remarquable. Par contre, *UBFO* est trés competetif avec PSO, WOA et GWO.

TABLE 6.2: Résultats obtenues pour les fonctions ayant une dimension fixe.

f	Algorithme	Best	Worst	Mean	StD	CPU (sec)	Rang
f_8	***UBFO***	**0.000e-00**	**2.000e-05**	**5.000e-05**	**4.417e-006**	0.7271	**1**
	PSO	**0.000e-00**	7.620e-01	7.621e-03	7.582e-02	**0.0145**	2
	WOA	**0.000e-00**	7.621e-01	3.391e-02	1.299e-01	0.0152	3
	GWO	**0.000e-00**	5.856e-00	2.341e-01	7.451e-01	0.0157	7
	MFO	3.400e-05	2.611e-01	3.135e-02	4.068e-02	0.0121	5
	BFO	7.190e-04	1.152e-01	2.943e-02	2.328e-02	0.0551	4
	MBFO	1.465e-03	1.015e-01	3.172e-02	2.489e-02	0.0485	6
f_9	***UBFO***	1.098e-02	7.086e-01	1.526e-01	1.036e-001	1.060	3
	PSO	**0.000e-00**	**1.452e-01**	**2.765e-03**	**1.869e-002**	**0.0125**	**1**
	WOA	3.180e-04	3.910e-00	4.365e-01	6.361e-01	0.0128	4
	GWO	9.230e-04	1.088e-01	2.563e-01	2.097e-02	0.0261	2
	MFO	2.091e-01	3.675e+02	2.298e+01	5.084e+01	0.0173	7
	BFO	3.984e-01	1.145e+01	5.011e-00	2.278e+00	0.0995	6
	MBFO	9.347e-01	9.131e+01	4.124594	1.706e+00	0.0929	5
f_{10}	***UBFO***	**0.000e-00**	9.724e-03	2.110e-03	3.599e-003	0.8515	2
	PSO	**0.000e-00**	**9.716e-03**	**1.555e-03**	**3.561e-003**	**0.0059**	**1**
	WOA	1.000e-06	**9.716e-03**	6.723e-03	4.356e-03	0.0181	3
	GWO	3.722e-02	1.269e-01	7.858e-02	1.424e-02	0.0159	5
	MFO	2.960e-04	3.782e-02	1.041e-02	5.116e-03	0.0168	4
	BFO	9.730E603	4.980e-01	3.432e-02	1.657e-01	0.0773	7
	MBFO	4.215e-03	4.663e-01	1.898e-01	1.253e-01	0.0725	6
f_{11}	***UBFO***	**-0.963534**	**-0.963534**	**-0.963534**	**5.487e-07**	0.7347	**1**
	PSO	**-0.963534**	**-0.963534**	**-0.963534**	5.961e-07	**0.0136**	2
	WOA	**-0.963534**	-0.963519	-0.963533	2.075e-06	0.0141	2
	GWO	**-0.963534**	-0.963519	-0.963531	2.880e-06	0.0199	4
	MFO	-0.963469	-0.681450	-0.949907	3.225e-02	0.0675	6
	BFO	-0.963531	-0.856868	-0.929287	3.005e-02	0.0653	7
	MBFO	**-0.963534**	-0.958836	-0.963071	7.806e-04	0.0159	5
f_{12}	***UBFO***	**3.000e+00**	**3.000e+00**	**3.000e+00**	**0.000e+00**	0.8607	**1**
	PSO	**3.000e+00**	**3.000e+00**	**3.000e+00**	**0.000e+00**	**0.0068**	**1**
	WOA	3.661e+00	8.811e+01	1.314e+01	1.754e+01	0.0097	2
	GWO	3.138e+00	8.403e+01	3.871e+01	3.461e+01	0.0156	3
	MFO	3.033e+00	1.052e+02	1.592e+01	1.381e+01	0.0119	4
	BFO	3.359e+00	2.353e+02	1.903e+01	2.597e+01	0.0501	6
	MBFO	3.402e+00	1.099e+02	1.635e=01	1.741e+01	0.0491	5

Fonctions benchmarks à multidimension : Les tableaux 6.5 - 6.10 indiquent que l'algorithme *UBFO* triomphe par rapport aux algorithmes BFO, MBFO et MFO, par exemple : quand dimension = 10 f_1_**UBFO= 4.448e-01** alors que f_1_BFO=1.721e+03 et f_1_MBOF = 1.753e+03 et nous remarquons que plus la dimension augmente plus la

TABLE 6.3: Suites des résultats de la Table. 6.2

f	Algorithme	Best	Worste	Mean	StD	CPU (sec)	Rang
f_{16}	*UBFO*	**0.000e+00**	6.880e-04	1.710e-04	1.513e-04	0.1922	2
	PSO	**0.000e+00**	**0.000e+00**	**0.000e+00**	1.185e-013	0.0148	**1**
	WOA	**0.000e+00**	4.100e-04	6.500e-05	9.1302e-05	0.0149	2
	GWO	4.000e-06	2.163e+00	3.619e-02	2.569e-01	0.0156	4
	MFO	1.480e-04	1.831e+00	2.075e-01	2.9162e-01	0.0123	5
	BFO	2.776e-02	3.885e+01	2.963e+00	6.129e+00	0.0595	6
	MBFO	1.907e-02	1.088e+02	6.542e+00	1.607e+001	0.0737	7
f_{17}	***UBFO***	**0.000e+00**	8.800e-05	2.300e-05	2.001e-005	0.2087	2
	PSO	**0.000e+00**	**0.000e+00**	**0.000e+00**	**0.000e+00**	**0.0144**	**1**
	WOA	1.100e-05	6.591e-02	7.831e-03	1.201e-02	0.0087	3
	GWO	**0.000e+00**	1.801e+00	7.202e-02	3.527e-01	0.0147	4
	MFO	4.350e604	1.059e+00	1.031e-01	1.618e-01	0.0132	5
	BFO	4.180e-04	1.193e+01	4.896e-01	2.387e+00	0.0675	6
	MBFO	2.337e-02	2.071e+01	3.009e+00	5.134e+00	0.0737	7
f_{18}	***UBFO***	4.007e-05	5.340e-03	8.770e-04	9.365e-04	0.3029	3
	PSO	**0.000e+00**	**0.000e+00**	**0.000e+00**	**0.000e+00** 0	**0.0361**	**1**
	WOA	**0.000e+00**	6.743e-01	3.386e-02	1.306e-01	**0.0099**	4
	GWO	**0.000e+00**	**0.000e+00**	**0.000e+00**	8.659e-17	0.0209	2
	MFO	**0.000e+00**	1.246e+00	4.363e-01	2.634e-01	0.0164	5
	BFO	6.060e-03	1.499e+00	6.312e-01	2.862e-01	0.0521	6
	MBFO	1.152e-01	1.592e+00	8.219e-01	3.093e-01	0.0681	7
f_{20}	***UBFO***	**-1.031628**	-1.031528	-1.031607	2.084e-05	0.2061	2
	PSO	**-1.031628**	**-1.031628**	**-1.031629**	**7.152e-07**	0.0088	**1**
	WOA	-1.031626	-1.020868	-1.030440	1.895e-003	**0.0079**	3
	GWO	0.000000	0.000000	0.000000	8.6592e-17	0.0209	7
	MFO	0.115296	1.592036	0.821938	3.0932e-01	0.0681	6
	BFO	-2.142169	-0.042459	-0.995860	3.1447e-01	0.0544	5
	MBFO	-1.031508	-0.669007	-0.984801	7.4167e-02	0.0124	4
f_{21}	***UBFO***	**-1.000e+00**	-1.400e-05	-9.799e-01	1.399e-001	0.5553	
	PSO	**-1.000e+00**	0.000e+00	-9.900e-01	9.948e-02	**0.0039**	2
	WOA	**-1.000e+00**	**-9.994e-01**	**-9.999e-01**	**8.407e-05**	0.0081	**1**
	GWO	-1.031e+00	2.104e+00	-8.804e-01	3.992e-01	0.0151	4
	MFO	-9.995e-01	0.000e+00	-8.065e-01	3.264e-01	0.0109	5
	BFO	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.0639	6
	MBFO	-3.300e-05	0.000e+00	0.000e+00	3.280e-06	0.0856	7

différence entre les résultats est remarquable. Par exemple : lorsque la dimension = 50 *UBFO* ateint l'optimum global, nous avons f_{19}_**UBFO=0.0**, alors que f_1_BFO=2.477e+09, f_{19}_MBFO = 6.434e+10 et f_{19}MFO = 1.826e+09.

Par ailleurs, ces résultats indiquent que *UBFO* est compétitif avec WOA et GWO. Ainsi, quand la dimension est égale à 10 et 30, les résultats sont suivis par PSO. Tandis

TABLE 6.4: Résultats statistiques basés sur le test de Wilcxon pour les fonctions de dimension fixe

f	UBFO vs					
	PSO	WOA	GWO	MFO	BFO	MBFO
f_8	* 1.720e-32	* 1.685e-31	* 2.808e-03	* 2.226e-34	* 2.226e-34	*2.226e-34
f_9	* 2.687e-36	4.151e-01	* 1.413e-28	* 5.420e-34	* 2.721e-34	*2.562e-34
f_{10}	* 1.562e-18	*1.972e-13	* 9.604e-37	* 2.215e-30	* 2.506e-34	*7.820e-34
f_{11}	_	*2.229e-03	* 6.809e-29	* 2.154e-38	* 5.639e-39	* 5.639e-39
f_{12}	* 5.638e-39	* 1.783e-33	* 3.642e-11	* 2.561e-34	* 2.562e-34	* 2.561e-34
f_{16}	* 5.636e-39	* 1.467e-10	* 3.321e-02	* 1.091e-33	* 2.561e-34	* 2.560e-34
f_{17}	* 2.997e-37	* 6.568e-33	* 2.100e-02	* 2.534e-34	* 2.534e-34	* 2.534e-34
f_{18}	* 5.638e-39	* 1.177e-24	* 1.575e-10	* 5.167e-33	*2.561e-34	* 2.561e-34
f_{21}	* 5.091e-35	* 2.417-30	* 5.638e-39	* 2.514e-34	* 3.376e-03	* 6.263e-04
f_{22}	* 1.132e-35	4.351e-01	* 5.897e-32	* 6.979e-32	* 5.467e-39	* 8.130e-39

Une valeur p où UBFO est plus performant que les autres algorithmes est marquée avec '*' en exposant et avec '_' où la performance de UBFO est similaire et '#' où la performance de UBFO est pire que les autres algorithmes.

que lorsque la dimension est égale à 50 UBFO produit de meilleurs résultats dans les fonctions (f_1,f_2,f_3,f_7 et f_{19}). La figure 6.1 montre que l'efficacité des algorithmes PSO, WOA, GWO, MFO, BFO et MBFO se dégrade avec l'augmentation de la dimension, contrairement à notre algorithme *UBFO*.

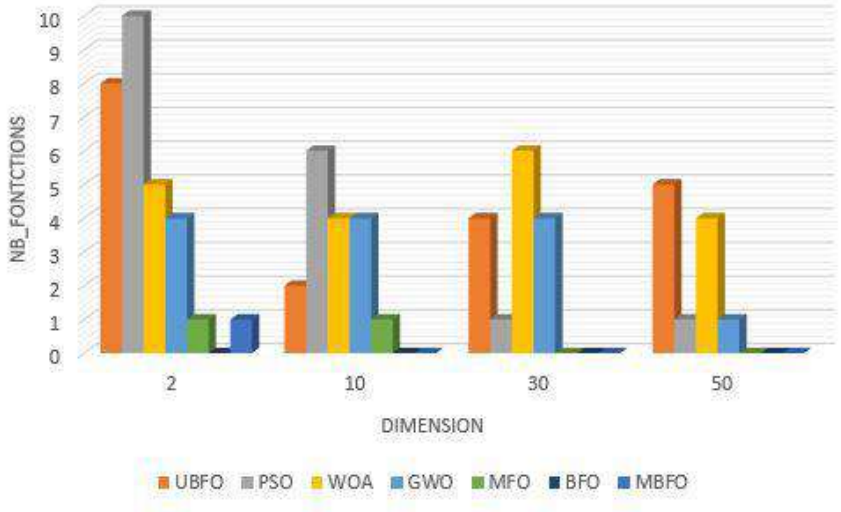


FIGURE 6.1: L'efficacité des algorithmes à résoudre les fonctions de problème d'optimisation selon la dimension utilisée.

TABLE 6.5: Résultats obtenus pour dim=10.

f	Algorithme	Best	Worste	Mean	StD	CPU (sec)	Rang
f_1	***UBFO***	4.448e-01	1.120e-01	7.782e-02	1.576e-02	1.5696	3
	PSO	1.300e-05	**4.989e-02**	**3.112e-03**	**6.724e-03**	0.0083	**1**
	WOA	1.060e-02	3.464e-01	9.390e-02	6.302e-02	0.0145	4
	GWO	**0.000e+00**	6.765e-02	7.245e-03	1.241e-02	0.0635	2
	MFO	8.397e-01	6.644e+02	5.119e+01	1.011e+02	0.0336	5
	BFO	1.721e+03	1.406e+04	9.204e+03	2.334e+03	0.1874	7
	MBFO	1.753e+03	1.522e+04	9.180e+04	2.812e+03	0.1864	6
f_2	***UBFO***	**0.000e+00**	**0.000e+00**	**0.000e+00**	**0.000e+00**	1.4488	**1**
	PSO	**0.000e+00**	**0.000e+00**	**0.000e+00**	**0.000e+00**	**0.0065**	**1**
	WOA	**0.000e+00**	**0.000e+00**	**0.000e+00**	**0.000e+00**	0.0171	**1**
	GWO	1.000e+00	4.300e+01	7.250e+00	5.564e+00	0.0661	3
	MFO	**0.000e+00**	1.300 e+00	1.170e+00	2.135e+00	0.0350	2
	BFO	7.000e+00	3.200e+00	2.000e+01	5.108e+00	0.1073	5
	MBFO	7.000e+00	2.600e+01	1.758e+01	3.605e+00	0.0986	4
f_3	***UBFO***	2.367e-02	1.102e-01	7.526e-02	1.790e-02	1.5838	2
	PSO	**2.000e-06**	**1.611e-02**	**1.766e-03**	**2.978e-03**	**0.0122**	**1**
	WOA	7.750e-03	3.231e-01	8.87e-02	5.682e-02	0.0153	3
	GWO	1.281e+00	2.500e+00	2.121e+00	2.799e-01	0.0636	4
	MFO	1.056e+00	4.927e+02	5.906e+01	1.019e+02	0.0338	5
	BFO	9.369e+02	1.533e+04	8.750e+03	2.684e+03	0.1873	7
	MBFO	1.967e+03	1.423e+04	8.833e+03	2.599e+03	0.1841	6
f_4	***UBFO***	5.345e-03	5.370e-02	2.891e-02	1.1339e-02	1.5960	2
	PSO	6.640e-04	1.003e+00	3.570e-02	1.701e-01	**0.0046**	3
	WOA	1.651e-01	6.776e+00	1.650e+00	1.131e+00	0.0156	4
	GWO	**6.460e-04**	**1.698e-02**	**2.348e-03**	**2.032e-03**	0.0645	**1**
	MFO	5.485e-02	2.487e+00	7.542e-01	4.685e-01	0.0475	5
	BFO	7.691e-02	1.545e+00	8.231e-01	2.500e-01	0.0861	7
	MBFO	3.407e-01	1.193e+00	7.985e-01	1.824e-01	0.0731	6
f_5	***UBFO***	-39.057x10	-31.931x10	-35.846x10	1.564e+01	1.5222	3
	PSO	-39.166x10	-30.683x10	-34.824x10	2.051e+01	**0.0038**	2
	WOA	**-39.165x10**	**-35.509x10**	**-39.126x10**	**3.635e+00**	0.0156	**1**
	GWO	-34.876x10	-16.266x10	-24.627x10	3.095e+01	0.0639	7
	MFO	-37.836 x10	-25.512x10	-33.122x10	2.183e+01	0.0325	6
	BFO	-38.983x10	-33.191x10	-36.250x10	1.219e+01	0.1337	5
	MBFO	-38.977x10	-33.029x10	-36.262x10	1.216e+01	0.1296	4
f_6	***UBFO***	2.155e-01	**8.994e-01**	6.605e-01	**1.305e-01**	1.0615	3
	PSO	**2.808e-03**	2.816e+00	5.871e-01	7.102e-01	**0.0106**	1
	WOA	1.221e-01	1.777e+00	**5.752e-01**	2.679e-01	0.0217	3
	GWO	2.316e+00	1.996e+01	1.872e+01	3.686e+00	0.0705	7
	MFO	2.309e+00	1.974e+01	6.671e+00	3.262e+00	0.0398	4
	BFO	1.506e+01	1.973e+01	1.868e+01	8.327e-01	0.1703	5
	MBFO	1.544e+01	2.013e+01	1.897e+01	8.827e-01	0.1387	6

TABLE 6.6: Suites aux résultats Table. 6.5

f	Algorithme	Best	Worste	Mean	StD	CPU (sec)	Rang
f_7	***UBFO***	8.790e-04	2.838e-01	8.055e-02	7.794e-02	1.2268	2
	PSO	**7.780e-04**	7.793e-01	2.021e-01	1.575e-01	0.0095	3
	WOA	1.423e-03	2.534e-01	**2.178e-02**	**4.587e-02**	0.0208	**1**
	GWO	1.285e-01	1.119e+00	4.337e-01	2.243e-01	0.0697	5
	MFO	1.147e-02	1.431e+00	2.867e-01	3.155e-01	0.0520	4
	BFO	4.898e+01	1.895e+02	1.348e+02	2.803e+01	0.1354	6
	MBFO	5.954e+01	1.944e+02	1.318e+02	3.056e+01	0.0836	7
f_{13}	***UBFO***	1.111e+01	2.907e+02	3.308e+01	5.054e+01	1.1234	3
	PSO	**0.000e+00**	2.502e+03	1.024e+02	3.575e+02	**0.0084**	4
	WOA	3.946e-01	**2.601e+01**	**1.097e+01**	**4.328e+00**	0.0151	**1**
	GWO	3.608e+00	7.721e+01	1.634e+01	1.902e+01	0.0638	2
	MFO	6.127e+01	1.470e+04	2.212e+03	2.685e+03	0.0332	5
	BFO	6.471e+03	2.249e+05	7.835e+04	4.470e+04	0.0764	7
	MBFO	5.149e+03	2.051e+05	7.309e+04	4.321e+04	0.0754	6
f_{14}	*UBFO*	1.27e-01	5.395e-01	3.153e-01	8.494e-02	1.1067	3
	PSO	0.3.200e-05	1.009e+02	1.014e+00	9.948e+00	**0.0082**	4
	WOA	2.601e-02	7.174e-01	2.374e-01	1.465e-01	0.0151	2
	GWO	**0.000e+00**	**2.589e-03**	**3.100e-04**	**5.173e-04**	0.0636	**1**
	MFO	2.438e+00	1.912e+02	3.583e+01	3.711e+01	0.0335	5
	BFO	2.134e+02	9.821e+02	5.604e+02	1.585e+02	0.0904	7
	MBFO	9.646e+01	1.029e+03	5.582e+02	1.613e+02	0.0762	6
f_{15}	***UBFO***	6.318e+00	5.105e+01	2.077e+01	1.103e+01	1.5515	3
	PSO	8.000e-06	1.208e+00	9.932e-02	2.291e-01	**0.0171**	2
	WOA	**7.000e-06**	**8.914e-03**	**1.110e-03**	**1.338e-03**	0.0265	**1**
	GWO	3.648e+00	2.530e+01	1.411e+01	3.528e+00	0.0820	4
	MFO	1.410e-01	4.452e+06	7.706e+04	4.722e+05	0.0421	5
	BFO	1.731e+04	1.962e+08	5.663e+07	3.626e+07	0.0915	7
	MBFO	7.345e+05	1.471e+08	5.054e+07	3.167e+007	0.0931	6
f_{19}	***UBFO***	**0.000e+00**	**0.000e+00**	**0.000e+00**	**1.229e-15**	1.6268	**1**
	PSO	**0.000e+00**	**0.000e+00**	**0.000e+00**	4.642e-10	**0.0254**	3
	WOA	**0.000e+00**	0.000008	**0.000e+00**	9.752e-07	0.028	4
	GWO	**0.000e+00**	**0.000e+00**	**0.000e+00**	9.163e-15	0.0797	2
	MFO	4.745e-01	1.003e+08	2.511e+06	1.084e+07	0.0467	5
	BFO	5.136e+05	5.552e+08	1.243e+08	1.3323e+08	0.0890	6
	MBFO	7.951e+05	6.528e+08	1.243e+08	1.315e+08	0.0849	7

6.3.2 Analyse statistique

Etant donné que les algorithmes métaheuristiques sont de nature stochastiques, la solution trouvée à chaque exécution peut ne pas être identique. Cependant, nous avons calculé la moyenne (Mean) et l'écart-type (StD) des fonctions d'optimisation pour analyser la précision et la stabilité des algorithmes donnés. Une valeur minime de l'écart-type indique une plus grande stabilité de l'algorithme. En revanche, une

TABLE 6.7: Résultats Obtnus avec D=30.

f	Algorithme	Best	Worste	Mean	StD	CPU (sec)	Rang
f_1	***UBFO***	**2.041e-01**	**5.716e-01**	**3.838e-01**	**4.618e-02**	2.6056	**1**
	PSO	1.361e+00	6.221e+01	1.052e+01	9.008e+00	**0.0176**	3
	WOA	2.334e-01	1.837e+00	7.388e-01	3.261e-01	0.0365	2
	GWO	3.016e+00	2.952e+02	9.605e+01	6.512e+01	0.1867	4
	MFO	2.033e+03	3.541e+04	1.392e+04	7.234e+03	0.0929	5
	BFO	4.205e+04	7.074e+04	5.927e+04	6.130e+03	0.3295	6
	MBFO	3.577e+04	7.311e+04	5.886e+04	6.891e+03	0.3291	7
f_2	***UBFO***	**0.000e+00**	**0.000e+00**	**0.000e+00**	**0.000e+00**	2.6865	**1**
	PSO	**0.000e+00**	3.000e+01	8.120e+00	6.005e+000	0.0194	2
	WOA	**0.000e+00**	**0.000e+00**	**0.000e+00**	**0.000e+00**	0.0440	**1**
	GWO	**0.000e+00**	2.980e+02	1.704e+01	4.372e+01	0.2013	3
	MFO	1.300e+01	7.700e+01	4.427e+01	1.438e+01	0.0934	4
	BFO	7.800e+01	1.360e+02	1.038e+02	1.067e+01	0.2223	6
	MBFO	7.200e+01	1.190e+02	9.890e+1	1.002e+01	0.2109	5
f_3	***UBFO***	**2.059e-01**	**5.771e-01**	**3.801e-01**	**5.199e-02**	2.7057	**1**
	PSO	1.842e+00	1.005e+02	1.139e+01	1.262e+01	**0.0208**	3
	WOA	2.117e-01	1.364e+00	6.456e-01	2.557e-01	0.0363	2
	GWO	7.274e+00	2.835e+02	9.587e+01	5.955e+01	0.1856	4
	MFO	9.694e+02	3.596e+04	1.225e+04	6.777e+03	0.0936	5
	BFO	3.078e+04	7.323e+4	5.828e+04	6.581e+03	0.3315	6
	MBFO	3.737e+04	7.204e+04	5.939e+04	6.756e+03	0.3293	7
f_4	***UBFO***	6.732e-02	5.262e-01	2.838e-01	8.208e-02	2.5080	2
	PSO	7.107e-02	2.973e+01	1.519e+00	4.095e+00	**0.0118**	3
	WOA	6.636e+00	6.566e+01	3.142e+01	1.287e+01	0.0370	4
	GWO	**8.470e-04**	**2.008e-01**	**5.985e-02**	**4.169e-02**	0.187	**1**
	MFO	7.725e+00	6.775e+01	2.604e+01	1.083e+01	0.1262	5
	BFO	1.449e+00	5.980e+00	3.519e+00	8.527e-01	0.2089	6
	MBFO	1.707e+00	5.303e+00	3.216e+00	7.322e-01	0.1910	7
f_5	***UBFO***	-37.117x30	-30.943x30	-33.811x30	3.591e+01	2.3735	2
	PSO	-36.010x30	-29.206x30	-32.740x30	4.007e+01	**0.0116**	3
	WOA	**-39.165x30**	**-36.640x30**	**-38.976x30**	**1.429e+01**	0.0383	**1**
	GWO	-29.399x30	-16.422x30	-23.454x30	7.321e+01	0.1862	6
	MFO	-31.368x30	-16.571x30	-23.453x30	8.062e+01	0.0875	7
	BFO	-35.577x30	-31,736x30	-33.649x30	2.454e+01	0.2435	4
	MBFO	-35.687x30	-31.892x30	-33.853x30	2.189e+01	0.2367	5
f_6	***UBFO***	1.808e+01	1.953e+01	1.908e+01	2.448e-01	3.2282	3
	PSO	2.491e+00	7.279e+00	4.120e+00	8.997e-01	**0.0312**	2
	WOA	**8.729e-02**	**1.886e+00**	**1.026e+00**	**3.687e-01**	0.0540	**1**
	GWO	1.996e+01	1.996e+01	1.996e+01	8.298e-04	0.2035	4
	MFO	1.409e+01	2.017e+01	1.914e+01	1.179e+00	0.1075	5
	BFO	1.870e+01	2.034e+01	2.001e+01	2.283e-01	0.2225	6
	MBFO	1.947e+01	2.076e+01	2.026e+01	2.236e-01	0.1693	7

TABLE 6.8: Suites des résultats Table. 6.7

f	Algorithme	Best	Worste	Mean	StD	CPU (sec)	Rang
f_7	***UBFO***	**5.524e-03**	**1.139e-02**	**8.183e-03**	**1.122e-003**	3.6483	**1**
	PSO	5.702e-01	2.945e+00	1.052e+00	3.324e-01	**0.0281**	3
	WOA	2.249e-02	1.886e-01	9.134e-02	3.918e-02	0.0537	2
	GWO	5.459e-01	3.639e+00	1.768e+00	6.329e-01	0.2034	4
	MFO	6.915e+00	2.535e+02	6.475e+01	5.618e+01	0.1253	5
	BFO	3.945e+02	7.036e+02	5.907e+02	7.106e+01	0.2851	6
	MBFO	4.651e+02	7.856e+02	6.012e+02	6.616e+01	0.1718	7
f_{13}	***UBFO***	5.385e+01	3.761e+02	1.228e+02	9.452e+01	1.7841	2
	PSO	2.316e+02	2.288e+05	3.261e+04	4.728e+04	**0.0251**	4
	WOA	3.497e+01	**1.324e+02**	**6.182e+01**	**1.530e+01**	0.0370	**1**
	GWO	**2.904e+01**	4.346e+02	1.751e+02	1.051e+02	0.1855	3
	MFO	5.781e+04	8.365e+05	3.009e+05	1.453e+05	0.0890	5
	BFO	3.169e+05	2.032e+06	1.344e+06	3.441e+05	0.1626	6
	MBFO	5.505e+05	1.919e+06	1.275e+06	3.103e+05	0.1564	7
f_{14}	***UBFO***	4.229e+00	1.402e+01	6.791e+00	1.736e+00	1.7561	2
	PSO	1.219e+01	2.637e+03	2.142e+02	3.297e+02	**0.0197**	4
	WOA	1.754e+00	**1.258e+01**	**5.316e+00**	**2.427e+00**	0.0368	**1**
	GWO	**7.599e-02**	2.998e+01	8.463e+00	6.335e+00	0.1857	3
	MFO	9.771e+02	5.740e+03	2.845e+03	1.086e+03	0.0892	5
	BFO	6.592e+03	1.106e+04	9.027e+03	1.076e+03	0.1790	6
	MBFO	6.269e+03	1.157e+04	9.150e+03	1.112e+03	0.1579	7
f_{15}	***UBFO***	1.831e+01	5.981e+01	3.547e+01	9.479e+00	2.0960	4
	PSO	1.062e+00	2.267e+01	6.753e+00	3.673e+00	**0.0524**	2
	WOA	**1.100e-04**	**6.075e-02**	**2.878e-03**	**6.395e-03**	0.0697	**1**
	GWO	1.673e+00	3.725e+01	2.061e+01	6.818e+00	0.2380	3
	MFO	9.832e+06	3.658e+08	1.354e+08	9.872e+07	0.1415	5
	BFO	1.419e+08	7.768e+08	5.254e+08	1.282e+08	0.1918	6
	MBFO	1.993e+08	9.224e+08	5.406e+08	1.196e+08	0.1953	7
f_{19}	***UBFO***	**0.000e+00**	**0.000e+00**	**0.000e+00**	**0.000e+00**	2.2501	**1**
	PSO	2.154e-03	2.369e+04	7.217e+02	2.956e+03	0.0771	3
	WOA	**0.000e+00**	**0.000e+00**	**0.000e+00**	**0.000e+00**	**0.0745**	**1**
	GWO	**0.000e+00**	3.665e+01	8.211e-01	3.949e+00	0.2238	2
	MFO	1.143e+08	1.156e+10	3.441e+09	2.229e+09	0.1293	4
	BFO	1.016e+09	1.192e+10	5.336e+09	2.329e+09	0.1999	5
	MBFO	1.262e+09	1.104e+10	5.365e+09	2.067e+09	0.1951	6

moyenne de valeur proche de l'optimum global indique une plus grande précision. Les valeurs de la moyenne et de l'écart-type, présentées dans les tableaux 6.2-6.10 indiquent que ***UBFO*** produit des solutions plus précises comparé aux autres algorithmes. En se basant sur les valeurs de StD, nous pouvons constater que ***UBFO*** est l'algorithme le plus stable.

TABLE 6.9: Résultats Obtnus avec D=50.

f	Algorithme	Best	Worste	Mean	StD	CPU (sec)	Rang
f_1	***UBFO***	**2.101e-01**	**1.073e+00**	**8.059e-01**	**8.921e-02**	4.8717	**1**
	PSO	2.734e+01	3.552e+02	1.171e+02	7.296e+01	**0.0250**	3
	WOA	2.978e-01	3.866e+00	1.738e+00	7.155e-01	0.0581	2
	GWO	6.757e+01	2.355e+03	9.934e+02	5.171e+02	0.3083	4
	MFO	2.764e+04	8.913e+04	5.481e+04	1.305e+04	0.1533	5
	BFO	9.241e+04	1.325e+05	1.151e+05	8.063e+03	0.4679	7
	MBFO	8.861e+04	1.296e+05	1.151e+05	8.128e+03	0.4671	6
f_2	***UBFO***	**0.000e+00**	**0.000e+00**	**0.000e+00**	**0.000e+00**	4.0963	**1**
	PSO	9.000e+00	7.600e+01	2.923e+01	1.005e+01	**0.0322**	2
	WOA	**0.000e+00**	**0.000e+00**	**0.000e+00**	**0.000e+00**	0.0712	**1**
	GWO	1.000e+00	4850e+020	1.875e+01	5.787e+01	0.3205	3
	MFO	7.800e+01	1.750e+02	1.224e+02	1.829e+01	0.1518	4
	BFO	1.390e+02	2.250e+02	1.866e+02	1.435e+01	0.3839	5
	MBFO	1.450e+02	2.080e+02	1.804e+02	1.208e+01	0.3496	6
f_3	***UBFO***	**4.156e-01**	**9.918e-01**	**8.016e-01**	**8.172e-02**	3.1450	**1**
	PSO	2.580e+01	5.734e+02	1.275e+02	9.762e+01	**0.0239**	3
	WOA	5.250e-01	3.296e+00	1.636e+00	6.019e-01	0.0587	2
	GWO	2.559e+02	2.266e+03	1.117e+03	4.666e+02	0.3086	4
	MFO	2.286e+04	9.243e+04	5.294e+04	1.312e+04	0.1524	5
	BFO	8.530e+04	1.385e+05	1.146e+05	8.487e+03	0.4709	6
	MBFO	9.253e+04	1.312e+05	1.155e+05	8.159e+03	0.4677	7
f_4	***UBFO***	2.164e-01	1.384e+00	8.261e-01	2.042e-01	3.4081	2
	PSO	3.102e-01	8.973e+01	1.026e+01	1.447e+01	**0.0198**	3
	WOA	3.385e+01	1.884e+02	9.931e+01	3.341e+01	0.0598	4
	GWO	**8.654e-02**	**9.524e-01**	**4.298e-01**	1.908e-01	0.3101	**1**
	MFO	5.013e+01	2.443e+02	1.188e+02	4.221e+01	0.2046	7
	BFO	4.877e+00	1.333e+01	8.720e+00	1.615e+00	0.3353	5
	MBFO	4.956e+00	1.216e+01	8.048e+00	1.393e+00	0.3191	6
f_5	***UBFO***	-35.511x50	-31.457x50	-33.135x50	4.452e+01	3.2557	2
	PSO	-34.798x50	-27.730x50	-31.256x50	7.041e+01	**0.0212**	3
	WOA	**-39.165x50**	**-36.013x50**	**-38.787x50**	**3.284e+01**	0.0601	**1**
	GWO	-26.003x50	-15.880x50	-21.503x50	9.405e+001	0.3094	6
	MFO	-24.901x50	-13.999x50	-19.410x50	1.054e+002	0.1442	7
	BFO	-33.415x50	-29.407x50	-30.943x50	3.457e+001	0.4167	5
	MBFO	-33.226x50	-29.675x50	-31.009x50	3.239e+001	0.4060	4
f_6	***UBFO***	1.891e+01	1.997e+01	1.963e+01	2.266e-01	3.3934	3
	PSO	4.723e+00	1.128e+01	7.187e+00	1.198e+00	**0.0508**	2
	WOA	**3.394e-01**	**2.042e+00**	**1.224e+00**	**3.597e-01**	0.0858	**1**
	GWO	1.996e+01	1.996e+01	1.996e+01	5.085e-04	0.3364	4
	MFO	1.868e+01	2.031e+01	1.997e+01	3.279e-01	0.174	5
	BFO	1.992e+01	2.062e+01	2.027e+01	1.277e-01	0.3319	6
	MBFO	1.969e+01	2.079e+01	2.049e+01	1.642e-01	0.2494	7

TABLE 6.10: Suites des résultats de la Table. 6.9

f	Algorithme	Best	Worste	Mean	StD	CPU	Rang
f_7	***UBFO***	**1.115e-02**	**2.208e-02**	**1.635e-02**	**2.084e-03**	4.3252	**1**
	PSO	1.052e+00	1.368e+01	2.427e+00	2.616e+00	**0.0543**	3
	WOA	2.648e-02	8.903e-01	1.676e-01	9.611e-02	0.0862	2
	GWO	3.741e-01	2.485e+01	9.374e+00	4.887e+00	0.3378	4
	MFO	1.323e+02	6.178e+02	3.389e+02	9.961e+01	0.1897	5
	BFO	8.919e+02	1.323e+03	1.114e+03	8.842e+01	0.4376	7
	MBFO	8.838e+02	1.295e+02	1.099e+02	8.022e+01	0.2641	6
f_{13}	***UBFO***	9.908e+01	9.287e+02	1.849e+02	1.176e+02	3.4554	2
	PSO	3.927e+03	1.072e+06	1.213e+05	1.277e+05	**0.0457**	4
	WOA	**6.935e+01**	**2.237e+02**	**1.233e+02**	**3.113e+01**	0.0597	**1**
	GWO	8.437e+01	5.665e+03	1.388e+03	1.134e+03	0.3092	3
	MFO	4.943e+05	2.892e+06	1.254e+06	4.553e+05	0.1445	5
	BFO	1.762e+06	4.389e+06	3.118e+06	5.335e+05	0.2312	7
	MBFO	1.612e+06	3.898e+06	3.002e+06	5.491e+05	0.2271	6
f_{14}	***UBFO***	1.389e+01	5.626e+01	2.771e+01	8.737e+00	2.4571	2
	PSO	1.481e+02	4.469e+03	1.176e+03	8.594e+02	**0.0275**	4
	WOA	**7.667e+00**	**4.991e+01**	2.145e+01	8.334e+00	0.0592	**1**
	GWO	7.547e+00	3.833e+02	1.536e+02	9.073e+01	0.3091	3
	MFO	8.846e+03	2.523e+04	1.575e+04	3.337e+03	0.1442	5
	BFO	2.249e+04	3.428e+04	2.871e+04	2.285e+03	0.2801	7
	MBFO	2.091e+04	3.345e+04	2.847e+04	2.455e+03	0.2496	6
f_{15}	***UBFO***	1.745e+01	5.446e+01	3.936e+01	7.882e+00	3.7636	3
	PSO	4.852e+00	5.368e+01	1.679e+01	8.352e+00	**0.0986**	2
	WOA	**2.530e-04**	**3.096e-02**	**3.125e-03**	**3.765e-03**	0.1133	**1**
	GWO	9.047e+00	2.508e+04	2.743e+02	2.485e+03	0.4009	4
	MFO	1.631e+08	1.054e+09	5.971e+08	1.775e+08	0.2550	5
	BFO	7.351e+08	1.396e+09	1.091e+09	1.572e+08	0.2988	6
	MBFO	6.552e+08	1.505e+09	1.147e+09	1.799e+08	0.3001	7
f_{19}	***UBFO***	**0.000e+00**	**1.800e-05**	**4.000e-06**	**3.008e-06**	3.9287	**1**
	PSO	2.588e+02	7.349e+05	7.444e+04	1.176e+05	0.1248	4
	WOA	**0.000e+00**	3.067e-03	1.690e-04	4.603e-04	0.1252	2
	GWO	4.182e-03	2.612e+04	9.557e+02	2.976e+03	0.3713	3
	MFO	1.826e+09	2.606e+10	1.324e+10	5.175+09	0.2110	5
	BFO	2.477e+09	2.627e+10	1.592e+10	3.701e+09	0.3205	6
	MBFO	6.434e+10	2.362e+10	1.595e+10	3.714e+09	0.3529	7

De plus, une analyse statistique utilisant le test de la somme des rangs (Run Sum) de Wilcoxon (Wilcoxon, 1945) est effectuée à un niveau de signification de 5%. Les valeurs des fonction benchmarks obtenues par UBFO sont comparées à celles obtenues par PSO, WOA, GWO, MFO, BFO et MBFO. Tous les algorithmes sont exécutés 100

fois pour l'analyse statistique. L'hypothèse nulle $(P = 0)$ indique qu'il n'y a pas de différence significative entre les algorithmes. Tandis que l'hypothèse alternative considère qu'il y a une différence significative entre les trois algorithmes. Les valeurs de p sont présentées dans les tableaux 6.4, 6.11,6.12 et 6.13. Une valeur de $p > 0,05$ indique que l'hypothèse nulle ne peut être rejetée. D'autre part, une valeur de p<0,05 signifie que l'hypothèse nulle peut être rejetée à un niveau de signification de 5%. Pour les fonctions de dimension fixe (tableau 6.11), l'algorithme UBFO produit un meilleur résultat dans tous les cas comparé aux algorithmes GWO, M FO,BFO et MBFO, 9 cas sur 10 comparé à PSO, et 8 cas sur 10 par rapport à WOA.

Le tableau 6.11 montre les valeurs de p en utilisant 10 comme dimension. D'après ces valeurs, *UBFO* produit de meuilleurs résultats dans 11 fonctions sur 11 comparé au MFO, mais dans 10 cas sur 11 par rapport aux algorithmes GWO, BFO et MBFO et dans 9 cas sur 11 comparé aux algorithmes PSO et WOA.

Selon le tableau 6.12, lorsque la dimension est égale à 30, nous observons que *UBFO* dépasse tous les algorithmes dans les 11 cas excepté PSO, où nous avons 10 cas sur 11. Par contre, quand la dimension est égale à 50 (tableau 6.13), l'algorithme UBFO génère de meilleurs résultats dans les 11 cas sur 11 comparé à tous les autres algorithmes.

Ces résultats indiquent qu'il existe une différence significative entre les 7 algorithmes. Dans la plupart des cas, l'algorithme ***UBFO*** est le *plus performant* relativement à tous les autres algorithmes.

6.3.3 Temps d'exécution

Les algorithmes utilisés ont été programmés sous Visual Studio en utilisant le langage C. L'exécution des tests a été faite sur une machine ayant : un processeur Intel® Core ™ i5_5300U CPU @ 2.30 GHz, 8 Go de RAM et un système d'exploitation Win10 (64) Pro.

Les résultats expérimentaux montrent que l'algorithme PSO est plus rapide par rapport aux autres algorithmes. En revanche, l'algorithme *UBFO* nécessite plus de temps dû à son complexité de calcul, ce qui pourrait-être amélioré en utilisant le calcul parallèle.

TABLE 6.11: Résultats statistiques basés sur le test de Wilcxon D= 10

f	UBFO vs					
	PSO	WOA	GWO	MFO	BFO	MBFO
f_1	*2.888e-034	6.311e-001	*4.725e-034	* 2.562e-034	* 2.562e-034	* 2.562e-034
f_2	_	_	* 3.489e-039	* 2.338e-014	* 5.338e-039	* 4.989e-039
f_3	*2.561e-034	* 3.499e-001	* 2.464e-034	* 2.562e-034	* 2.562e-034	* 2.562e-034
f_4	* 1.663e-026	* 2.562e-034	* 6.292e-034	* 2.562e-034	* 2.562e-034	* 2.562e-034
f_5	# 8.255e-002	* 1.537e-033	* 9.002e-034	* 1.371e-016	# 5.388e-001	# 5.951e-001
f_6	* 1.938e-003	* 2.595e-005	* 2.562e-034	* 2.562e-034	* 2.562e-034	* 2.562e-034
f_7	* 1.575e-009	#6.796e-002	* 1.320e-029	*8.119e-008	* 2.562e-034	* 2.562e-034
f_{13}	* 1.265e-007	* 2.773e-015	* 6.749e-014	* 8.232e-034	* 2.562e-034	* 2.562e-034
f_{14}	* 1.575e-032	* 1.048e-007	* 2.128e-034	* 2.5624e-034	* 2.562e-034	* 2.562e-034
f_{15}	* 1.897e-018	* 2.561e-034	* 9.810e-006	* 9.979e-005	* 2.562e-034	* 2.562e-034
f_{19}	_	* 4.036e-003	_	* 5.6405e-039	* 5.640e-039	* 5.640e-039

Une valeur p où UBFO est plus performant que les autres algorithmes est marquée avec '*' en exposant et avec '_' où la performance de UBFO est similaire et '#' où la performance de UBFO est pire que les autres algorithmes.

TABLE 6.12: Résultats statistiques basés sur le test de Wilcxon D=30

f	UBFO vs					
	PSO	WOA	GWO	MFO	BFO	MBFO
f_1	* 2.562e-034	* 7.481e-021	* 2.562e-034	* 2.562e-034	* 2.563e-034	* 2.562e-034
f_2	* 9.492e-036	#8.274e-002	* 7.159e-029	* 1.600e-038	* 1.597e-038	* 1.586e-038
f_3	* 2.562e-034	* 7.397e-017	* 2.562e-034	* 2.562e-034	* 2.562e-034	* 2.5624e-034
f_4	# 7.332e-001	* 2.562e-034	* 2.471e-033	* 2.562e-034	* 2.562e-034	* 2.562e-034
f_5	* 2.412e-008	* 2.641e-034	* 2.562e-034	* 2.640e-034	# 3.576e-001	# 6.259e-001
f_6	* 2.562e-034	* 2.562e-034	* 2.558e-034	* 1.457e-004	* 1.283e-032	* 2.641e-034
f_7	* 2.561e-034	* 2.561e-034	* 2.561e-034	* 2.561e-034	* 2.561e-034	* 2.567e-034
f_{13}	* 6.293e-034	* 4.638e-008	* 1.755e-004	* 2.562e-034	* 2.562e-034	* 2.562e-034
f_{14}	* 2.640e-034	* 1.665e-006	# 2.721e-001	* 2.562e-034	* 2.5621e-034	* 2.5624e-034
f_{15}	* 3.161e-034	* 2.562e-034	* 1.137e-023	* 2.562e-034	* 2.562e-034	* 2.562e-034
f_{19}	* 3.021e-035	* 1.506e-019	* 3.262e-024	* 3.021e-035	* 3.021e-035	* 3.027e-035

Une valeur p où UBFO est plus performant que les autres algorithmes est marquée avec '*' en exposant et avec et '#' où la performance de UBFO est pire que les autres algorithmes.

6.4 Application de *UBFO* dans la segmentation d'image

Afin de valider la performance de notre algorithme *UBFO*, nous l'avons appliqué dans la résolution du problème de segmentation des images en niveaux de gris *(Lena, Pepper et Cameraman*. Cet algorithme est appliqué au problème de multi-seuillage en utilisant la fonction d'Otsu (Eq (3.15)). La méthode proposée est testée sur un ensemble d'images de test standard. Les performances de la méthode proposée sont ensuite comparées avec l'algorithme WOA, qui s'est montré très compétitif lors des tests sur les fonctions benchmarks vu précédemment. Les résultats ont été analysés sur la base des meilleures valeurs de fitness, PSNR et SSIM.

TABLE 6.13: Résultats statistiques basés sur le test de Wilcxon D=50

f	UBFO vs					
	PSO	WOA	GWO	MFO	BFO	MBFO
f_1	* 2.564e-034	* 7.348e-027	* 2.564e-034	* 2.562e-034	* 2.563e-034	* 2.562e-034
f_2	* 2.641e-034	* 7.492e-029	* 1.295e-015	* 2.164e-034	* 2.156e-034	* 2.152e-034
f_3	* 2.564e-034	* 5.743e-025	* 2.564e-034	* 2.564e-034	* 2.564e-034	* 2.562e-034
f_4	* 4.989e-017	* 2.562e-034	* 4.105e-024	* 2.562e-034	* 2.564e-034	* 2.562e-034
f_5	* 6.236e-019	* 2.569e-034	* 2.562e-034	* 2.562e-034	* 3.752e-030	* 5.715e-030
f_6	* 2.564e-034	* 2.564e-034	* 5.012e-033	* 2.355e-019	* 2.977e-034	* 1.212e-033
f_7	* 2.562e-034	* 2.564e-034	* 5.018e-033	* 2.562e-034	* 2.562e-034	* 2.562e-034
f_{13}	* 2.564e-034	* 9.112e-005	* 6.822e-028	* 2.562e-034	* 2.562e-034	* 2.564e-034
f_{14}	* 2.562e-034	* 1.241e-006	* 9.145e-031	* 2.564e-034	* 2.562e-034	* 2.562e-034
f_{15}	* 1.517e-029	* 2.562e-034	* 3.481e-020	* 2.562e-034	* 2.562e-034	* 2.544e-034
f_{19}	* 1.871e-034	* 8.008e-024	* 1.871e-034	* 1.879e-034	* 1.871e-034	* 1.869e-034

Une valeur p où UBFO est plus performant que les autres algorithmes est marquée avec '*' en exposant .

6.4.1 Résultats et discussion

Les valeurs de la fonction objectif obtenues par les méthodes Otsu-exhaustive, UBFO , WOA et sont présentées dans le tableau 6.14, accompagnées de leurs seuils optimaux correspondants. Sachant que dans un problème de maximisation, plus la valeur de la fonction objective est élevée, plus la solution est meilleure. Ces résultats indiquent que la méthode basée sur *UBFO* produit de meilleurs résultats que WOA sur les trois images testées (avec une différence remarquable). De plus, UBFO trouve souvent des solutions identiques à celles produites par la recherche exhaustive.

Le tableau 6.15 présente les valeurs SSIM et PSNR, celles-ci indiquent que la performance de la méthode proposée (UBFO) est meilleure que celle de WOA pour tous les seuils utilisés.

TABLE 6.14: Les valeurs de la fonction objective obtenues par les méthodes recherche-exhaustive, UBFO et WOA avec leurs seuils correspondants.

		Otsu value			Thresholds		
Images de test	TH	Exhaustive	UBFO	WOA	Exhaustive	UBFO	WOA
Lena	2	1961.58	***1961,58***	1961.41	93,151	***93,151***	94,152
	3	2128.26	***2128,26***	2127.77	81,127,171	***81,127,171***	79,127,170
	4	2191.60	***2191,60***	2180.68	75,114,145,180	***75,114,145,180***	78,112,134,175
	5	2217.50	***2216***,99	2212.55	73,109,136,160,188	***74,107,135,157,189***	79,110,140,167,188
Pepper	2	2866.33	***2866.33***	2433.36	93,177	***93,177***	56,152
	3	3066.24	***3066.09***	2493.18	73,126,178	***73,126,179***	72,119,165
	4	3151.99	***3151.93***	2647.67	46,84,130,179	***45,84,130,179***	47,74,132,155
	5	3195.93	***3195.91***	2669.47	43,76,111,144,181	***43,76,111,144,181***	29,85,97,133,157
Cameraman	2	3650.33	***3650.33***	3649.36	70,144	***70,144***	
	3	3725.71	***3725.65***	3690.12	59,119,156	***59,117,156***	59,143,208
	4	3780.68	***3780.67***	3760.51	42,95,140,170	***42,95,140,170***	77,135,200,222
	5	3812.00	***3811.92***	3795.86	36,82,122,149,173	***36,83,122,149,173***	3,53,111,166,184

TABLE 6.15: La moyenne de la mesure PSNR et SSIM de toutes les méthodes.

		SSIM			PSNR		
Images de test	TH	Exhaustive	UBFO	WOA	Exhaustive	UBFO	WOA
Lena	2	0.9256	***0.9256***	0.9254	22.96	***22.96***	22.95
	3	0.9466	***0.9467***	0.9463	26.03	***26.02***	26.00
	4	0.9805	***0.9804***	0.9789	28.17	***28.18***	27.83
	5	0.9846	***0.9844***	0.9811	29.47	***29.44***	29.06
Pepper	2	0.9306	***0.9305***	0.6478	21.80	***21.80***	16.27
	3	0.9619	***0.9618***	0.6581	24.52	***24.53***	18.16
	4	0.9834	**0.9834**	0.7152	26.57	***26.56***	20.63
	5	0.9851	***0.9850***	0.7511	28.18	***28.15***	21.42
Cameraman	2	0.9705	***0.9705***	0.6455	24.39	***24.39***	18.28
	3	0.9807	***0.9792***	0.6935	26.06	***26.06***	19.90
	4	0.9867	***0.9865***	0.7384	27.88	***27.87***	21.18
	5	0.9925	***0.9914***	0.7668	29.37	***29.34***	22.16

6.5 Conclusion

Dans notre travail, nous avons proposé une version améliorée de l'algorithme BFO. Les modifications apportées se résument à : une modification du calcul de la taille de pas ($c_{(i)}$) de sorte à le rendre adaptatif au nombre des itérations de l'étape chimiotaxie ainsi qu'une modification dans l'étape Elimination/Dispersion qui consiste à utiliser une probabilité plutôt dynamique afin de favoriser l'élimination des mauvaises bactéries et de préserver celles jugées efficaces.

Les performances de l'algorithme proposé ont été testées avec diverses fonctions de tests standard et comparées aux algorithmes PSO, WOA, GWO , MFO, BFO et MBFO.

Les résultats obtenus pour la majorité des fonctions tests montrent que l'algorithme ***UBFO*** est meilleur que les algorithmes BFO, MBFO et MFO en termes de qualité de solution, d'efficacité de calcul mais aussi de stabilité. En revanche nous remarquons que *UBFO* est très compétitif par rapport aux algorithmes PSO et WOA.

De plus, la méthode proposée ***UBFO*** a été appliquée dans la résolution du problème de multi-seuillage, et comparée à WOA. Tous les résultats expérimentaux indiquent que ***UBFO*** produit de meilleures solutions avec une différence remarquable.

Cependant, l'algorithme *UBFO* est plus lent en calcul par rapport aux autres algorithmes, chose qui peut être améliorée en utilisant un calcul parallèle surtout avec l'évolution des processeurs et du nombre de cores.

Chapitre 7

Conclulion générale

Dans cette thèse, nous avons proposé une série de solutions aux problèmes liés au traitement d'image particulièrement la compression et la segmentation d'image en se basant sur l'utilisation des méta-heuristiques. Nous avons aussi proposé une amélioration de l'algorithme BFO.

Comme première contribution, nous avons testé l'efficacité de trois algorithmes méta-heuristiques (notamment, WOA, PSO et FA) à résoudre le problème de la quantification vectorielle, qui consiste à générer un dictionnaire optimal global meilleur que celui généré par l'algorithme LBG. Notons que l'étude expérimentale a montré que l'application des méta-heuristiques dans la QV n'apporte aucun gain ni en termes de la qualité de la solution ni en temps d'exécution. Bien au contraire, l'utilisation de l'algorithme LBG basé sur la technique de dichotomie vectorielle (Splitting) produit un dictionnaire plus optimal et en un temps très réduit comparé aux LBG-Random, WOA, PSO et FA. Cela n'est pas étonnant vu que le théorème NFL a confirmé que les algorithmes méta-heuristiques ne peuvent pas être bénéfiques pour tous les problèmes d'optimisation. Ainsi, nous nous sommes penchés sur un autre traitement d'image très important dans l'analyse et l'interprétation de l'image qui est la segmentation.

Notre deuxième contribution consistait à proposer une nouvelle fonction objective pour résoudre le problème du multi-seuillage, ce dernier joue un rôle principal dans la segmentation d'image. Cela dit, la qualité des processus succédant la segmentation dépendent fortement de la qualité des seuils trouvés. Afin de remédier à ce problème nous avons utilisé la métrique SSIM comme fonction objective optimisée par PSO, BA et FA avec des paramètres préalablement ajustés. Nos résultats obtenus ont montré que l'algorithme PSO produit des solutions quasi-identiques à identiques aux solutions exactes (obtenues par la recherche exhaustive). De plus la fonction objective

SSIM a permis de générer des images segmentées de meilleures qualités comparés à l'utilisation de méthode classique : Otsu.

Bien que les algorithmes méta-heuristiques progressent vers la solution optimale en un temps raisonnable, la fonction SSIM s'est avérée un peu couteuse en temps de calcul. De ce fait, la complexité de calcul a été considérablement réduite en adoptant un paradigme de programmation parallèle à mémoire partagée pour tous les algorithmes que nous avons implémentés.

Quant à notre troisième contribution, une étude comparative de notre algorithme BFO amélioré (Upgraded-BFO) a été présentée. Nous avons apporté des modifications à l'algorithme BFO dans le but d'améliorer sa convergence et la qualité des solutions obtenues, puis nous l'avons comparé aux algorithmes : PSO, WOA, GWO, MFO, BFO et MBFO. Les expérimentations ont montré que notre algorithme UBFO produit de meilleurs résultats par rapport aux algorithmes BFO, MBFO et MFO et qu'il très compétitif avec PSO et WOA. De plus, les tests statistiques ont montré que, bien que relativement plus lent, UBFO est l'algorithme le plus précis et le plus stable par rapport aux autres algorithmes. Toutefois, nous avons appliqué UBFO dans notre problème principale (le multi-seuillage), et les résultats ont montré que UBFO est nettement plus performant que WOA qui était très compétitif avec celui-ci lors des tests benchmarks.

En perspectives, nous envisageons d'utiliser l'approche concernant le multi-seuillage sur d'autres types d'image (images médicales, images satellitaires). Nous prévoyons également de proposer une version parallèle de l'algorithme UBFO et de l'appliquer à des problèmes réels liés au traitement de données multimédia.

Bibliography

Abd El Aziz, Mohamed, Ahmed A Ewees, and Aboul Ella Hassanien (2017). "Whale optimization algorithm and moth-flame optimization for multilevel thresholding image segmentation". In: *Expert Systems with Applications* 83, pp. 242–256.

– (2018). "Multi-objective whale optimization algorithm for content-based image retrieval". In: *Multimedia tools and applications* 77.19, pp. 26135–26172.

Abd Elaziz, Mohamed, Ahmed A Ewees, and Diego Oliva (2020). "Hyper-heuristic method for multilevel thresholding image segmentation". In: *Expert Systems with Applications* 146, p. 113201.

Agrawal, Sanjay et al. (2013). "Tsallis entropy based optimal multilevel thresholding using cuckoo search algorithm". In: *Swarm and Evolutionary Computation* 11, pp. 16–30.

Alba, Enrique (2005). *Parallel metaheuristics: a new class of algorithms*. Vol. 47. John Wiley & Sons, Inc.

Ali, Musrrat, Patrick Siarry, and Millie Pant (2012). "An efficient differential evolution based algorithm for solving multi-objective optimization problems". In: *European journal of operational research* 217.2, pp. 404–416.

Bae, Changseok et al. (2012). "A new simplified swarm optimization (SSO) using exchange local search scheme". In: *International Journal of Innovative Computing, Information and Control* 8.6, pp. 4391–4406.

Bakhshali, Mohamad Amin and Mousa Shamsi (2014). "Segmentation of color lip images by optimal thresholding using bacterial foraging optimization (BFO)". In: *Journal of Computational Science* 5.2, pp. 251–257.

Balabanian, Felipe, Eduardo Sant'Ana da Silva, and Helio Pedrini (2017). "Image Thresholding Improved by Global Optimization Methods". In: *Applied Artificial Intelligence* 31.3, pp. 197–208.

Ballard, Dana Harry and Christopher M. Brown (1982). *Computer Vision*. 1st. Prentice Hall Professional Technical Reference. ISBN: 0131653164.

Bardekar, Ms Asmita A and Mr PA Tijare (2011). "A review on LBG algorithm for image compression". In: *International Journal of Computer Science and Information Technologies* 2.6, pp. 2584–2589.

Bhandari, A.K. et al. (Nov. 2016). "A Novel Color Image Multilevel Thresholding Based Segmentation Using Nature Inspired Optimization Algorithms". In: *Expert Syst. Appl.* 63.C, pp. 112–133. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2016.06.044. URL: https://doi.org/10.1016/j.eswa.2016.06.044.

Bhandari, Ashish Kumar, Anil Kumar, and Girish Kumar Singh (2015). "Modified artificial bee colony based computationally efficient multilevel thresholding for satellite image segmentation using Kapur's, Otsu and Tsallis functions". In: *Expert Systems with Applications* 42.3, pp. 1573–1601.

Bhandari, Ashish Kumar et al. (2014). "Cuckoo search algorithm and wind driven optimization based study of satellite image segmentation for multilevel thresholding using Kapur's entropy". In: *Expert Systems with Applications* 41.7, pp. 3538–3560.

Birattari, Mauro et al. (2001). "Classification of metaheuristics and design of experiments for the analysis of components". In: *Teknik Rapor, AIDA-01-05*.

Blum, Christian and Andrea Roli (Sept. 2003). "Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison". In: *ACM Comput. Surv.* 35.3, 268–308. ISSN: 0360-0300. DOI: 10.1145/937503.937505. URL: https://doi.org/10.1145/937503.937505.

Boubechal, Ikram, Rachid Seghir, and Redha Benzid (2018). "Vector-Quantization Codebook Generation using LBG and Meta-Heuristic Algorithms". In: 7^{th} International Conference on Metaheuristics and Nature Inspired Computing.

– (2019). "A Generalized and Parallelized SSIM-Based Multilevel Thresholding Algorithm". In: *Applied Artificial Intelligence* 33.14, pp. 1266–1289.

Boussaïd, Ilhem, Julien Lepagnot, and Patrick Siarry (2013). "A survey on optimization metaheuristics". In: *Information sciences* 237, pp. 82–117.

Bovik, Alan C. (2009). *The Essential Guide to Image Processing*. USA: Academic Press, Inc. ISBN: 0123744571. DOI: https://doi.org/10.1016/B978-0-12-374457-9.X0001-7.

Canny, John (1986). "A computational approach to edge detection". In: *IEEE Transactions on pattern analysis and machine intelligence* 6, pp. 679–698.

Chai, Ruishuai (2021). "Otsu's Image Segmentation Algorithm with Memory-Based Fruit Fly Optimization Algorithm". In: *Complexity* 2021.

Chan, Tony and Jianhong Shen (2005). *Image Processing And Analysis: Variational, Pde, Wavelet, And Stochastic Methods*. USA: Society for Industrial and Applied Mathematics. ISBN: 089871589X.

Chang, Yian-Leng and Xiaobo Li (1994). "Adaptive image region-growing". In: *IEEE transactions on image processing* 3.6, pp. 868–872.

Chehdi, K. and D. Coquin (1991). "Binarisation of various images by detecting local thresholds with a validation test". In: *[1991] IEEE Pacific Rim Conference on Communications, Computers and Signal Processing Conference Proceedings*, 611–614 vol.2. DOI: 10.1109/PACRIM.1991.160813.

Chen, Kai et al. (2016). "Multilevel image segmentation based on an improved firefly algorithm". In: *Mathematical Problems in Engineering* 2016.

Chen, Qian, Jiangang Yang, and Jin Gou (2005). "Image compression method using improved PSO vector quantization". In: *International conference on natural computation*. Springer, pp. 490–495.

Cheriet, M., J. N. Said, and C. Y. Suen (1998). "A recursive thresholding technique for image segmentation". In: *IEEE Transactions on Image Processing* 7.6, pp. 918–921. ISSN: 1057-7149. DOI: 10.1109/83.679444.

Clerc, Maurice and James Kennedy (2002). "The particle swarm-explosion, stability, and convergence in a multidimensional complex space". In: *IEEE transactions on Evolutionary Computation* 6.1, pp. 58–73.

Crainic, Teodor (2019). "Parallel metaheuristics and cooperative search". In: *Handbook of Metaheuristics*. Springer, pp. 419–451.

Crainic, Teodor Gabriel (2008). "Parallel solution methods for vehicle routing problems". In: *The vehicle routing problem: Latest advances and new challenges*. Springer, pp. 171–198.

Crainic, Teodor Gabriel, Tatjana Davidović, and Dušan Ramljak (2014). "Designing parallel meta-heuristic methods". In: *Handbook of research on high performance and cloud computing in scientific research and education*. IGI Global, pp. 260–280.

Crainic, Teodor Gabriel and Nourredine Hail (2005). "1 PARALLEL META-HEURISTICS APPLICATIONS". In: *Parallel metaheuristics: A new class of algorithms* 47, p. 447.

Crainic, Teodor Gabriel and Michel Toulouse (1998). "Parallel metaheuristics". In: *Fleet management and logistics*. Springer, pp. 205–251.

Crane, Randy (1996). *Simplified Approach to Image Processing: Classical and Modern Techniques in C*. 1st. USA: Prentice Hall PTR. ISBN: 0132264161.

Cremers, Daniel, Mikael Rousson, and Rachid Deriche (2007). "A review of statistical approaches to level set segmentation: integrating color, texture, motion and shape". In: *International journal of computer vision* 72.2, pp. 195–215.

Cuevas, Erik, Daniel Zaldivar, and Marco Pérez-Cisneros (2010). "A novel multi-threshold segmentation approach based on differential evolution optimization". In: *Expert Systems with Applications* 37.7, pp. 5265–5271.

Cuevas, Erik et al. (2012). "A multi-threshold segmentation approach based on artificial bee colony optimization". In: *Applied Intelligence* 37.3, pp. 321–336.

Cui, Zhihua, Feixiang Li, and Wensheng Zhang (2019). "Bat algorithm with principal component analysis". In: *International Journal of Machine Learning and Cybernetics* 10.3, pp. 603–622.

Cung, Van-Dat et al. (2002). "Strategies for the parallel implementation of metaheuristics". In: *Essays and surveys in metaheuristics*. Springer, pp. 263–308.

Dagum, Leonardo and Ramesh Menon (Jan. 1998). "OpenMP: An Industry-Standard API for Shared-Memory Programming". In: *IEEE Comput. Sci. Eng.* 5.1, pp. 46–55. ISSN: 1070-9924. DOI: 10.1109/99.660313. URL: https://doi.org/10.1109/99.660313.

De Albuquerque, M Portes, Israel A Esquef, and AR Gesualdi Mello (2004). "Image thresholding using Tsallis entropy". In: *Pattern Recognition Letters* 25.9, pp. 1059–1065.

Deriche, Rachid (1987). "Using Canny's criteria to derive a recursively implemented optimal edge detector". In: *International journal of computer vision* 1.2, pp. 167–187.

Digalakis, Jason G and Konstantinos G Margaritis (2001). "On benchmarking functions for genetic algorithms". In: *International journal of computer mathematics* 77.4, pp. 481–506.

Du, Ke-Lin, MNS Swamy, et al. (2016). "Search and optimization by metaheuristics". In: *Techniques and Algorithms Inspired by Nature*.

El Aziz, Mohamed Abd, Ahmed A Ewees, and Aboul Ella Hassanien (2017). "Whale optimization algorithm and moth-flame optimization for multilevel thresholding image segmentation". In: *Expert Systems with Applications* 83, pp. 242–256.

Erdmann, H et al. (2015). "A study of a firefly meta-heuristics for multithreshold image segmentation". In: *Developments in medical image processing and computational vision*. Springer, pp. 279–295.

Farshi, Taymaz Rahkar, John H Drake, and Ender Özcan (2020). "A multimodal particle swarm optimization-based approach for image segmentation". In: *Expert Systems with Applications* 149, p. 113233.

Farshi, Taymaz Rahkar and Mohanna Orujpour (2021). "A multi-modal bacterial foraging optimization algorithm". In: *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–15.

Fayad, Hadi, Mathieu Hatt, and Dimitris Visvikis (2015). "PET functional volume delineation using an Ant colony segmentation approach." In: *Journal of Nuclear Medicine* 56.supplement 3, pp. 1745–1745.

Fister, Iztok et al. (2013). "A comprehensive review of firefly algorithms". In: *Swarm and Evolutionary Computation* 13, pp. 34–46.

Flynn, Michael J (1972). "Some computer organizations and their effectiveness". In: *IEEE transactions on computers* 100.9, pp. 948–960.

Fonseca, CS, Felipe ABS Ferreira, and Francisco Madeiro (2018). "Vector quantization codebook design based on Fish School Search algorithm". In: *Applied Soft Computing* 73, pp. 958–968.

Franti, Pasi et al. (2000). "Fast and memory efficient implementation of the exact PNN". In: *IEEE Transactions on Image Processing* 9.5, pp. 773–777.

Friedrich, Tobias et al. (2010). "Approximating covering problems by randomized search heuristics using multi-objective models". In: *Evolutionary Computation* 18.4, pp. 617–633.

Fu, King-Sun and JK Mui (1981a). "A survey on image segmentation". In: *Pattern recognition* 13.1, pp. 3–16.

– (1981b). "A survey on image segmentation". In: *Pattern recognition* 13.1, pp. 3–16.

Gao, Hao, Chi-Man Pun, and Sam Kwong (2016). "An efficient image segmentation method based on a hybrid particle swarm algorithm with learning strategy". In: *Information Sciences* 369, pp. 500–521.

Gao, Hao et al. (2009). "Multilevel thresholding for image segmentation through an improved quantum-behaved particle swarm algorithm". In: *IEEE transactions on instrumentation and measurement* 59.4, pp. 934–946.

Gao, YAN et al. (2011). "Optimal region growing segmentation and its effect on classification accuracy". In: *International journal of remote sensing* 32.13, pp. 3747–3763.

Gendreau, Michel and Jean-Yves Potvin (2005). "Tabu search". In: *Search methodologies*. Springer, pp. 165–186.

Gersho, Allen and Robert M Gray (2012). *Vector quantization and signal compression*. Vol. 159. Springer Science & Business Media.

Ghamisi, Pedram et al. (2012). "An efficient method for segmentation of images based on fractional calculus and natural selection". In: *Expert Systems with Applications* 39.16, pp. 12407–12417.

Glover, Fred (1986). "Future paths for integer programming and links to artificial intelligence". In: *Computers & operations research* 13.5, pp. 533–549.

– (1990). "Tabu search—part II". In: *ORSA Journal on computing* 2.1, pp. 4–32.

Glover, Fred and Manuel Laguna (1998). "Tabu search". In: *Handbook of combinatorial optimization*. Springer, pp. 2093–2229.

Gonzalez, Rafael C. and Richard E. Woods (2006). *Digital Image Processing (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc. ISBN: 013168728X.

– (2018). *Digital Image Processing (4rd Edition)*. 330 Hudson Street, New York, NY 10013: Pearson Education, Inc. ISBN: 9780133356724.

Goshtasby, A Ardeshir and Stavri Nikolov (2007). "Image fusion: advances in the state of the art". In: *Information fusion* 2.8, pp. 114–118.

Gras, Robin et al. (2003). "Cooperative metaheuristics for exploring proteomic data". In: *Artificial Intelligence Review* 20.1, pp. 95–120.

Hammouche, Kamal, Moussa Diaf, and Patrick Siarry (Feb. 2008). "A Multilevel Automatic Thresholding Method Based on a Genetic Algorithm for a Fast Image Segmentation". In: *Comput. Vis. Image Underst.* 109.2, pp. 163–175. ISSN: 1077-3142. DOI: 10.1016/j.cviu.2007.09.001. URL: http://dx.doi.org/10.1016/j.cviu.2007.09.001.

– (2010). "A comparative study of various meta-heuristic techniques applied to the multilevel thresholding problem". In: *Engineering applications of artificial intelligence* 23.5, pp. 676–688.

Hang, Hseuh-Ming, John W Woods, et al. (1995). *Handbook of visual communications*. Academic Press.

Haralick, Robert M and Linda G Shapiro (1985). "Image segmentation techniques". In: *Applications of Artificial Intelligence II*. Vol. 548. International Society for Optics and Photonics, pp. 2–10.

Hassanien, Aboul Ella and Diego Alberto Oliva (2017). *Advances in Soft Computing and Machine Learning in Image Processing*. Vol. 730. Springer.

Hastings, W. K. (Apr. 1970). "Monte Carlo sampling methods using Markov chains and their applications". In: *Biometrika* 57.1, pp. 97–109. ISSN: 0006-3444. DOI: 10.1093/biomet/57.1.97. eprint: https://academic.oup.com/biomet/article-pdf/57/1/97/23940249/57-1-97.pdf. URL: https://doi.org/10.1093/biomet/57.1.97.

Haupt, Randy L. and Sue Ellen Haupt (2004). *Practical Genetic Algorithms, Second Edition*. USA: John Wiley & Sons, Inc. ISBN: 0471455652.

Homayouni, S Mahdi and Dalila BMM Fontes (2018). *Metaheuristics for maritime operations*. Wiley Online Library.

Horng, Ming-Huwi (2011). "Multilevel thresholding selection based on the artificial bee colony algorithm for image segmentation". In: *Expert Systems with Applications* 38.11, pp. 13785–13791.

– (2012). "Vector quantization using the firefly algorithm for image compression". In: *Expert Systems with Applications* 39.1, pp. 1078–1091.

Horng, Ming-Huwi and Ting-Wei Jiang (2011). "The artificial bee colony algorithm for vector quantization in image compression". In: *2011 4th IEEE International Conference on Broadband Network and Multimedia Technology*. IEEE, pp. 319–323.

Horowitz, Steven L and Theodosios Pavlidis (1976). "Picture segmentation by a tree traversal algorithm". In: *Journal of the ACM (JACM)* 23.2, pp. 368–388.

Hussain, Kashif et al. (2017). "Common benchmark functions for metaheuristic evaluation: A review". In: *JOIV: International Journal on Informatics Visualization* 1.4-2, pp. 218–223.

Hussain, Kashif et al. (2019). "Metaheuristic research: a comprehensive survey". In: *Artificial Intelligence Review* 52.4, pp. 2191–2233.

Jain, Anil K, Robert P. W. Duin, and Jianchang Mao (2000). "Statistical pattern recognition: A review". In: *IEEE Transactions on pattern analysis and machine intelligence* 22.1, pp. 4–37.

Jain, Anil K, M Narasimha Murty, and Patrick J Flynn (1999). "Data clustering: a review". In: *ACM computing surveys (CSUR)* 31.3, pp. 264–323.

Jamil, Momin and Xin-She Yang (2013). "A literature survey of benchmark functions for global optimisation problems". In: *International Journal of Mathematical Modelling and Numerical Optimisation* 4.2, pp. 150–194.

Kapur, Jagat Narain, Prasanna K Sahoo, and Andrew KC Wong (1985). "A new method for gray-level picture thresholding using the entropy of the histogram". In: *Computer vision, graphics, and image processing* 29.3, pp. 273–285.

Karayiannis, Nicolaos B and Zhiying Liu (2000). "Split and merge codebook design algorithms for image compression". In: *Journal of Electronic Imaging* 9.4, pp. 509–520.

Karri, Chiranjeevi and Umaranjan Jena (2016). "Fast vector quantization using a Bat algorithm for image compression". In: *Engineering Science and Technology, an International Journal* 19.2, pp. 769–781.

Kass, Michael, Andrew Witkin, and Demetri Terzopoulos (1988). "Snakes: Active contour models". In: *International journal of computer vision* 1.4, pp. 321–331.

Kennedy, J. and R. Eberhart (1995). "Particle swarm optimization". In: *Proceedings of ICNN'95 - International Conference on Neural Networks*. Vol. 4, 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968.

Kirkpatrick, Scott, C Daniel Gelatt, and Mario P Vecchi (1983). "Optimization by simulated annealing". In: *science* 220.4598, pp. 671–680.

Kittler, J. and J. Illingworth (Jan. 1986). "Minimum Error Thresholding". In: *Pattern Recogn.* 19.1, pp. 41–47. ISSN: 0031-3203. DOI: 10.1016/0031-3203(86)90030-0. URL: http://dx.doi.org/10.1016/0031-3203(86)90030-0.

Koschan, Andreas and Mongi A. Abidi (2008). *Digital Color Image Processing*. USA: Wiley-Interscience. ISBN: 0470147083.

Kotte, Sowjanya, P Rajesh Kumar, and Satish Kumar Injeti (2018). "An efficient approach for optimal multilevel thresholding selection for gray scale images based on improved differential search algorithm". In: *Ain Shams Engineering Journal* 9.4, pp. 1043–1067.

Li, Jun et al. (2014). "Analysis and improvement of the bacterial foraging optimization algorithm". In: *Journal of Computing Science and Engineering* 8.1, pp. 1–10.

Li, Xiaodong et al. (2013). "Benchmark functions for the CEC 2013 special session and competition on large-scale global optimization". In: *gene* 7.33, p. 8.

Liao, Huilian et al. (2007). "A novel optimizer based on particle swarm optimizer and LBG for vector quantization in image coding". In: *Third International Conference on Natural Computation (ICNC 2007)*. Vol. 3. IEEE, pp. 416–420.

Linde, Yoseph, Andres Buzo, and Robert Gray (1980). "An algorithm for vector quantizer design". In: *IEEE Transactions on communications* 28.1, pp. 84–95.

Lloyd, Stuart (1982). "Least squares quantization in PCM". In: *IEEE transactions on information theory* 28.2, pp. 129–137.

Maitra, Madhubanti and Amitava Chatterjee (Feb. 2008). "A Hybrid Cooperative-comprehensive Learning Based PSO Algorithm for Image Segmentation Using Multilevel Thresholding". In: *Expert Syst. Appl.* 34.2, pp. 1341–1350. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2007.01.002. URL: http://dx.doi.org/10.1016/j.eswa.2007.01.002.

McInerney, Tim and Demetri Terzopoulos (2000). "T-snakes: Topology adaptive snakes". In: *Medical image analysis* 4.2, pp. 73–91.

Metropolis, Nicholas et al. (1953). "Equation of state calculations by fast computing machines". In: *The journal of chemical physics* 21.6, pp. 1087–1092.

Mirjalili, Seyedali (2015). "Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm". In: *Knowledge-based systems* 89, pp. 228–249.

Mirjalili, Seyedali and Andrew Lewis (2016). "The whale optimization algorithm". In: *Advances in engineering software* 95, pp. 51–67.

Mirjalili, Seyedali, Seyed Mohammad Mirjalili, and Andrew Lewis (2014). "Grey wolf optimizer". In: *Advances in engineering software* 69, pp. 46–61.

Monga, O and B Wrobel (1987). "Segmentation d'images: vers une méthodologie". In: *TS Traitement du signal* 4.3, pp. 169–193.

Muthukrishnan, R and Miyilsamy Radha (2011). "Edge detection techniques for image segmentation". In: *International Journal of Computer Science & Information Technology* 3.6, p. 259.

Nag, Sayan (2019). "Vector quantization using the improved differential evolution algorithm for image compression". In: *Genetic Programming and Evolvable Machines* 20.2, pp. 187–212.

Naik, P, Pedda Sadhu, and T Venu Gopal (2016). "Particle swarm optimization (PSO) based k-means image segmentation algorithm". In: *International Journal of Scientific Research* 5.1.

Nixon, Mark and Alberto Aguado (2019). *Feature extraction and image processing for computer vision*. Academic Press.

Nocedal, Jorge and Stephen Wright (2006). *Numerical optimization*. Springer Science & Business Media.

Ohlander, Ron, Keith Price, and D Raj Reddy (1978). "Picture segmentation using a recursive region splitting method". In: *Computer Graphics and Image Processing* 8.3, pp. 313–333.

Oliva, Diego, Mohamed Abd Elaziz, and Salvador Hinojosa (2019). *Metaheuristic algorithms for image segmentation: theory and applications*. Springer.

Oliva, Diego and Erik Cuevas (2016). *Advances and Applications of Optimised Algorithms in Image Processing*. 1st. Springer Publishing Company, Incorporated. ISBN: 3319485490, 9783319485492.

– (2017). *Advances and applications of optimised algorithms in image processing*. Springer.

Otsu, N. (1979). "A Threshold Selection Method from Gray-Level Histograms". In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1, pp. 62–66. ISSN: 0018-9472. DOI: 10.1109/TSMC.1979.4310076.

Pal, Nikhil R, James C Bezdek, and EC-K Tsao (1993). "Generalized clustering networks and Kohonen's self-organizing scheme". In: *IEEE transactions on Neural Networks* 4.4, pp. 549–557.

Pal, Nikhil R and Sankar K Pal (1993). "A review on image segmentation techniques". In: *Pattern recognition* 26.9, pp. 1277–1294.

Panda, Rutuparna and Manoj Kumar Naik (2012). "A crossover bacterial foraging optimization algorithm". In: 2012.

Pare, S et al. (2016). "A multilevel color image segmentation technique based on cuckoo search algorithm and energy curve". In: *Applied Soft Computing* 47, pp. 76–102.

Passino, Kevin M (2002). "Biomimicry of bacterial foraging for distributed optimization and control". In: *IEEE control systems magazine* 22.3, pp. 52–67.

Pedemonte, Martín, Sergio Nesmachnow, and Héctor Cancela (2011). "A survey on parallel ant colony optimization". In: *Applied Soft Computing* 11.8, pp. 5181–5197.

Prewitt, Judith MS (1970). "Object enhancement and extraction". In: *Picture processing and Psychopictorics* 10.1, pp. 15–19.

Pruthi, Jyotika and Gaurav Gupta (2016). "Image segmentation using genetic algorithm and OTSU". In: *Proceedings of fifth international conference on soft computing for problem solving*. Springer, pp. 473–480.

Radosavljević, Jordan (2018). *Metaheuristic optimization in power engineering*. Institution of Engineering and Technology.

Rajpoot, NM et al. (2004). "A novel image coding algorithm using ant colony system vector quantization". In:

Reddi, S. S., S. F. Rudin, and H. R. Keshavan (1984). "An optimal multiple threshold scheme for image segmentation". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-14.4, pp. 661–665. ISSN: 0018-9472. DOI: 10.1109/TSMC.1984.6313341.

Rego, César and Bahram Alidaee (2006). *Metaheuristic optimization via memory and evolution: tabu search and scatter search*. Vol. 30. Springer Science & Business Media.

Ridler, TW, S Calvard, et al. (1978). "Picture thresholding using an iterative selection method". In: *IEEE trans syst Man Cybern* 8.8, pp. 630–632.

Roberts, Lawrence G (1963). "Machine perception of three-dimensional solids". PhD thesis. Massachusetts Institute of Technology.

Russ, John C. (2011). *The Image Processing Handbook, Sixth Edition*. 6th. USA: CRC Press, Inc. ISBN: 1439840458.

Sahoo, Prasanna K, SAKC Soltani, and Andrew KC Wong (1988). "A survey of thresholding techniques". In: *Computer vision, graphics, and image processing* 41.2, pp. 233–260.

Salomon, David and Giovanni Motta (2010). *Handbook of data compression*. Springer Science & Business Media.

Sanyal, Nandita, Amitava Chatterjee, and Sugata Munshi (2013). "Modified bacterial foraging optimization technique for vector quantization-based image compression". In: *Computational Intelligence in Image Processing*. Springer, pp. 131–152.

Sathya, P. D. and R. Kayalvizhi (Nov. 2011a). "Optimal Multilevel Thresholding Using Bacterial Foraging Algorithm". In: *Expert Syst. Appl.* 38.12, pp. 15549–15564. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2011.06.004. URL: http://dx.doi.org/10.1016/j.eswa.2011.06.004.

Sathya, PD and R Kayalvizhi (2011b). "Modified bacterial foraging algorithm based multilevel thresholding for image segmentation". In: *Engineering Applications of Artificial Intelligence* 24.4, pp. 595–615.

Sethian, James A (1996). "A fast marching level set method for monotonically advancing fronts". In: *Proceedings of the National Academy of Sciences* 93.4, pp. 1591–1595.

Shehab, Mohammad et al. (2020). "Moth–flame optimization algorithm: variants and applications". In: *Neural Computing and Applications* 32.14, pp. 9859–9884.

Shi, Y. and R. C. Eberhart (1999). "Empirical study of particle swarm optimization". In: *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*. Vol. 3, 1945–1950 Vol. 3. DOI: 10.1109/CEC.1999.785511.

Shih, Frank Y (2010). *Image processing and pattern recognition: fundamentals and techniques*. John Wiley & Sons.

Snyder, Wesley et al. (Dec. 1990). "Optimal Thresholding&Mdash;a New Approach". In: *Pattern Recogn. Lett.* 11.12, pp. 803–810. ISSN: 0167-8655. DOI: 10.1016/0167-8655(90)90034-Y. URL: http://dx.doi.org/10.1016/0167-8655(90)90034-Y.

Sobel, Irvin (1978). "Neighborhood coding of binary images for fast contour following and general binary array processing". In: *Computer graphics and image processing* 8.1, pp. 127–135.

Sum, KW and Paul YS Cheung (2007). "Boundary vector field for parametric active contours". In: *Pattern Recognition* 40.6, pp. 1635–1645.

Talbi, El-Ghazali (2009). *Metaheuristics: From Design to Implementation*. Wiley Publishing. ISBN: 0470278587.

Tanenbaum, Andrew S. (2012). *Structured Computer Organization*. 6th. Pearson Education. ISBN: 0273769243.

Tang, Kezong et al. (2011). "An improved scheme for minimum cross entropy threshold selection based on genetic algorithm". In: *Knowledge-Based Systems* 24.8, pp. 1131–1138.

Tang, Kezong et al. (2017). "An improved multilevel thresholding approach based modified bacterial foraging optimization". In: *Applied Intelligence* 46.1, pp. 214–226.

Thakare, Punam (2011). "A study of image segmentation and edge detection techniques". In: *International Journal on Computer Science and Engineering* 3.2, pp. 899–904.

Tilton, James C et al. (2012). "Best merge region-growing segmentation with integrated nonadjacent region object aggregation". In: *IEEE Transactions on Geoscience and Remote Sensing* 50.11, pp. 4454–4467.

Tsai, Wen-Hsiang (1985). "Moment-preserving thresolding: A new approach". In: *Computer Vision, Graphics, and Image Processing* 29.3, pp. 377–393.

Tyagi, Vipin (Sept. 2018). *Understanding Digital Image Processing*. ISBN: 9781315123905. DOI: 10.1201/9781315123905.

Wan, S-Y and William E Higgins (2003). "Symmetric region growing". In: *IEEE Transactions on Image processing* 12.9, pp. 1007–1015.

Wang, Xiaolei et al. (2009). " Hybrid nature-inspired computation methods for optimization". In: *Faculty of Electronics, Communications and Automation. Department of Electrical Engineering*. ISSN: 1795-4584.

Wang, Z. and A. C. Bovik (2009). "Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures". In: *IEEE Signal Processing Magazine* 26.1, pp. 98–117. ISSN: 1053-5888. DOI: 10.1109/MSP.2008.930649.

Wang, Zhou et al. (2004). "Image quality assessment: from error visibility to structural similarity". In: *IEEE Transactions on Image Processing* 13.4, pp. 600–612. ISSN: 1057-7149. DOI: 10.1109/TIP.2003.819861.

Wolpert, David H, William G Macready, et al. (1995). *No free lunch theorems for search*. Tech. rep. Technical Report SFI-TR-95-02-010, Santa Fe Institute.

Yang, Xin-She (2008). *Nature-Inspired Metaheuristic Algorithms*. Luniver Press. ISBN: 1905986106, 9781905986101.

– (2010a). "Firefly algorithm, stochastic test functions and design optimisation". In: *International journal of bio-inspired computation* 2.2, pp. 78–84.

– (2012). "Nature-inspired mateheuristic algorithms: success and new challenges". In: *arXiv preprint arXiv:1211.6658*.

Yang, Xin-She and Xingshi He (2013). "Bat algorithm: literature review and applications". In: *International Journal of Bio-inspired computation* 5.3, pp. 141–149.

Yang, Xin—She (2010b). "A new metaheuristic bat-inspired algorithm Nature Inspired Cooperative Strategies for Optimization (NICSO 2010) 2010 284 Berlin". In: *Germany Springer* 65, p. 74.

Yanxia, Liang, Yang Jiawei, and Li Ye (2011). "One effective method to design LBG initial codebook". In: *2011 Fourth International Conference on Intelligent Computation Technology and Automation*. Vol. 2. IEEE, pp. 628–631.

Yen, Jui-Cheng, Fu-Juay Chang, and Shyang Chang (1995). "A new criterion for automatic multilevel thresholding". In: *IEEE Transactions on Image Processing* 4.3, pp. 370–378.

Yin, Peng-Yeng (2007). "Multilevel minimum cross entropy threshold selection based on particle swarm optimization". In: *Applied mathematics and computation* 184.2, pp. 503–513.

Zhang, Hui, Jason E. Fritts, and Sally A. Goldman (May 2008). "Image Segmentation Evaluation: A Survey of Unsupervised Methods". In: *Comput. Vis. Image Underst.* 110.2, pp. 260–280. ISSN: 1077-3142. DOI: 10.1016/j.cviu.2007.08.003. URL: http://dx.doi.org/10.1016/j.cviu.2007.08.003.

Zhang, Jian et al. (2014). "An improved quantum-inspired genetic algorithm for image multilevel thresholding segmentation". In: *Mathematical Problems in Engineering* 2014.

Zhang, YJ and Jan J Gerbrands (1994). "Objective and quantitative segmentation evaluation and comparison". In: *Signal processing* 39.1-2, pp. 43–54.

Zhang, Yu-Jin (2006). *Advances in image and video segmentation*. IGI Global.

Zhou, Zheng and Yuhui Shi (2011). "Inertia Weight Adaption in Particle Swarm Optimization Algorithm". In: *Proceedings of the Second International Conference on Advances in Swarm Intelligence - Volume Part I*. ICSI'11. Chongqing, China: Springer-Verlag, pp. 71–79. ISBN: 978-3-642-21514-8. URL: http://dl.acm.org/citation.cfm?id=2026282.2026293.