Liste des figures

Chapitre 5

Liste des tableaux

Table Des matières

Introduction générale

« Votre temps est limité, ne le gâchez pas en menant une existence qui n'est pas la vôtre ».

Steve Jobs

La taille des images augmente proportionnellement à leur qualité et leur stockage et transmission qui constituent donc les enjeux principaux dans le monde numérique. La compression s'impose comme une étape incontournable pour optimiser l'utilisation de ces grands volumes d'informations dans les réseaux informatiques. L'objectif principal de la compression d'image est de réduire la quantité d'information nécessaire à une représentation visuelle fidèle à l'image originale. En générale on différencie les méthodes de compression selon la perte d'informations. Les méthodes réversibles, utilisent uniquement le principe de la réduction de la redondance et n'engendrent pas de perte. Les méthodes irréversibles, définissent une représentation approximative de l'information.

Toutes ces méthodes présentent des avantages et des inconvénients ; le but et par contre de trouver un compromis entre les différents critères à vérifier après un algorithme de compression (temps de calcul ; taille de l'image ; dégradation totale…etc.), donc de concevoir un système de compression qui permet d'avoir une meilleure qualité de la compression. Pour cela on a utilisé des méthodes d'optimisation connues sous l'appellation de ***Métaheuristiques.***

Les métaheuristiques sont une famille d'algorithmes stochastiques. Apparues au début des années quatre-vingts, elles sont destinées à résoudre des problèmes d'optimisation difficile. Utilisées dans de nombreux domaines, ces méthodes

présentent l'avantage d'être généralement efficaces, sans pour autant que l'utilisateur ait à modifier la structure de base de l'algorithme qu'il utilise.

Ces algorithmes stochastiques d'optimisation peuvent être appliqués à tout problème, du moment qu'il est formulé sous la forme de l'optimisation de critère(s). Ces algorithmes sont inspirés par des analogies avec la physique (recuit simulé, recuit micro canonique), avec la biologie (algorithmes évolutionnaires) ou avec l'éthologie (colonies de fourmis, essaims particulaires). Ils se prêtent aussi à toutes sortes d'extensions, notamment en optimisation multiobjective.

D'autre part l'utilisation des métaheuristiques d'optimisation dans le domaine de la compression d'image est assez nouvelle. Dans ce contexte d'étude nous nous intéressons à la conception de métaheuristique d'optimisation pour la compression d'image. Le but est donc d'essayer d'appliquer plusieurs métaheuristiques d'optimisation sur un algorithme de compression d'image, afin d'améliorer les critères de ce dernier.

Dans cette thèse de doctorat, nous allons, et pour la première fois, procéder à une étude plus détaillée au sujet de l'Algorithme de la Meute de loups pour la compression fractale d'image.
L'image entière est considérée comme un espace de recherche où cet espace est divisé en blocs, les loups chercheurs explorent l'espace pour trouver d'autres plus petit blocs qui se ressemblent. Le processus sera arrêté après un nombre fixe d'itérations ou si aucune amélioration dans la solution du loup chef est trouvée. A la fin un mappage est dressé avec les meilleurs solutions trouvées afin de reconstruire l'image en question.
Notre méthode est originale dans le sens où elle utilise, et pour la première fois l'algorithme de la meute des loups dans la compression fractale. Les résultats ont montré l'amélioration du taux et du temps de compression tout en offrant une qualité acceptable de l'image reconstruite.

Ce mémoire de thèse est constitué en cinq chapitres comme suit :
Une **introduction générale** présentant le contexte de l'étude qui renferme deux grandes axes à savoir la compression d'image et les métaheuristique

d'optimisation, on passe ensuite à l'objectif de notre recherche qui est l'élaboration d'une nouvelle méthode de compression fractale basée sur l'utilisation d'une toute nouvelle métaheuristique appelée Algorithme du Pacte des loups (Wolf Pack Algorithm).

Un premier chapitre. Compression d'image : état de l'art.

Ce chapitre est consacré à la présentation des principaux concepts de la compression. Où on commencera par la définition de la compression de données en général, puis on va mettre l'accent sur la compression d'image en dressant en détail les différentes méthodes de compression d'image des deux famille (réversible et irréversible).

Cela va nous permettre de chercher les points faibles de quelques algorithmes (compression fractale) afin qu'on puisse introduire le mécanisme d'optimisation adéquat.

Le deuxième chapitre : Métaheuristiques d'optimisation

Dans ce chapitre nous aborderons le domaine d'optimisation des problèmes aux données incomplètes, incertaines, bruitées ou tout simplement ceux confrontés à une capacité de calcul limitée. Une taxinomie des Métaheuristiques d'optimisation est dressée à la fin de ce chapitre.

Le troisième chapitre. Métaheuristiques d'optimisation pour la compression d'image : travaux similaires.

Dans le troisième chapitre nous nous intéressons aux travaux abordés dans le contexte de l'utilisation des métaheuristiques d'optimisation pour la compression d'image et plus précisément la compression fractale améliorée par l'utilisation de ces méthodes. Ces travaux ont été bien inspectés aussi bien côté avantages que côté inconvénients, à la fin une comparaison des différents travaux est dressée.

Le quatrième chapitre. Approche proposée : Wolf Pack Algorithme pour la compression fractale d'image

Ce chapitre sera consacré à expliquer notre approche proposée pour mettre en évidence une nouvelle méthode de compression fractale en utilisant une métaheuristique d'optimisation nouvelle nommée Wolf Pack Algorithm.

Le cinquième chapitre. Résultats et discussion

Ce dernier chapitre exposera les différents résultats obtenus après l'expérimentation faites sur des images standards, accompagnés par des comparaisons et des commentaires pour mettre en valeur notre approche proposée.

Nous terminerons ce mémoire de thèse par une **conclusion générale** et quelques perspectives ouvrant la porte pour d'autres travaux de recherche.

Chapitre 1. Compression de données : état de l'art

« La théorie, c'est quand on sait tout et que rien ne fonctionne. La pratique, c'est quand tout fonctionne et que personne ne sait pourquoi ».

Albert Einstein

1.1 Introduction

Avec l'utilisation généralisée de l'Internet, avec les coûts décroissants des numériseurs et des disques, l'archivage, la transmission et la manipulation des documents se fait de plus en plus sur ordinateur et de moins en moins sur papier. L'écran de nos ordinateurs est en train de devenir le moyen privilégié de consultation de documents parce qu'il permet un accès immédiat à l'information.
La taille des images augmente proportionnellement à leur qualité et leur stockage et transmission constituent donc les enjeux principaux dans le monde numérique. La compression s'impose comme une étape incontournable pour optimiser l'utilisation de ces grands volumes d'informations dans les réseaux informatiques. L'objectif principal de la compression d'image est de réduire la quantité d'information nécessaire à une représentation visuelle fidèle à l'image originale. En générale, on différencie les méthodes de compression selon la perte d'informations. Les méthodes réversibles, utilisent uniquement le principe de la réduction de la redondance et n'engendrent pas de perte. Les méthodes irréversibles, définissent une représentation approximative de l'information.

Compression physique et compression logique

On considère généralement la compression comme un algorithme capable de comprimer énormément de données dans un minimum de place (compression

physique), mais on peut également adopter une autre approche et considérer qu'en premier lieu un algorithme de compression a pour but de recoder les données dans une représentation différente plus compacte contenant la même information (compression logique).

La distinction entre compression physique et logique est faite sur la base de comment les données sont compressées ou plus précisément comment est-ce que les données sont réarrangées dans une forme plus compacte.

La compression physique est exécutée exclusivement sur les informations contenues dans les données. Cette méthode produit typiquement des résultats incompréhensibles qui apparemment n'ont aucun sens. Le résultat d'un bloc de données compressées est plus petit que l'original car l'algorithme de compression physique a retiré la redondance qui existait entre les données elles-mêmes. Toutes les méthodes de compression dont nous allons traiter sont des méthodes physiques.

La compression logique est accomplie à travers le processus de substitution logique qui consiste à remplacer un symbole alphabétique, numérique ou binaire en un autre. Changer "United State of America" en "USA" est un bon exemple de substitution logique car "USA" est dérivé directement de l'information contenue dans la chaîne "United State of America" et garde la même signification. La substitution logique ne fonctionne qu'au niveau du caractère ou plus haut et est basée exclusivement sur l'information contenue à l'intérieur même des données. Un autre exemple, moins heureux, de substitution logique est de remplacer 1999 par 99...

Compression avec et sans pertes

C'est peut-être le critère de comparaison le plus important pour les algorithmes de compression.

Une bonne partie des schémas de compression utilisés sont appelés sans pertes (réversibles), cela signifie que lorsque des données sont compressées et ensuite décompressées, l'information originale contenue dans les données a été préservée. Aucune donnée n'a été perdue ou oubliée. Les données n'ont pas été modifiées.

La méthode de compression avec pertes (irréversible) quant à elle "jette", de façon sélective, quelques données d'une image dans le but d'effectuer la compression

avec un taux de compression meilleur que la plupart des méthodes de compression sans pertes. Les algorithmes avec pertes s'appliquent généralement aux données ayant de forts taux de redondance, comme les images, ou les sons. Certaines méthodes tirent partis d'algorithmes heuristiques élaborés qui s'ajustent eux-mêmes pour trouver le rapport de compression maximum possible en changeant aussi peu que possible les détails visibles d'une image. Autrement dit, d'autres algorithmes moins élégants suppriment carrément la portion la moins significative de chaque pixel.

1.2 Principales méthodes de compression

Ces méthodes se divisent en deux familles; à savoir les méthodes dites sans pertes et celles avec pertes de données.

1.2.1 Méthodes sans pertes

La compression sans perte ou codage entropique ou codage réversible permet de retrouver la valeur exacte du signal comprimé lorsqu'il n'y aucune perte de données sur l'information d'origine. En fait, la même information est réécrite d'une manière plus concise.

1.2.1.1 Codage d'Huffman

Huffman [Huf 52] a suggéré une méthode statistique qui permet d'attribuer un mot-code binaire aux différents symboles (pixel) à compresser. La probabilité d'occurrence du symbole dans l'image est prise en compte en attribuant aux plus fréquents des codes courts, et aux plus rares des codes longs, VLC - Variable Length Coding. La suite finale de pixels codés à longueurs variables sera plus petite que la taille originale. Le codeur Huffman crée un arbre ordonné à partir de tous les symboles et de leur fréquence d'apparition. Les branches sont construites récursivement en partant des symboles les plus fréquents. Le code de chaque symbole correspond à la suite des codes le long du chemin allant de ce caractère à la racine. Plus le symbole est profond dans l'arbre plus la quantité de bits pour le représenter est importante. La figure 3 présente un exemple de codeur d'Huffman. Le tableau de gauche montre que nous avons au total 21 symboles qui sont représentés en octets équivalant à 168 bits. Après la construction de l'arbre d'Huffman nous pouvons constater un taux de compression de $\sigma = 45/ (168) = 26.79\%$.

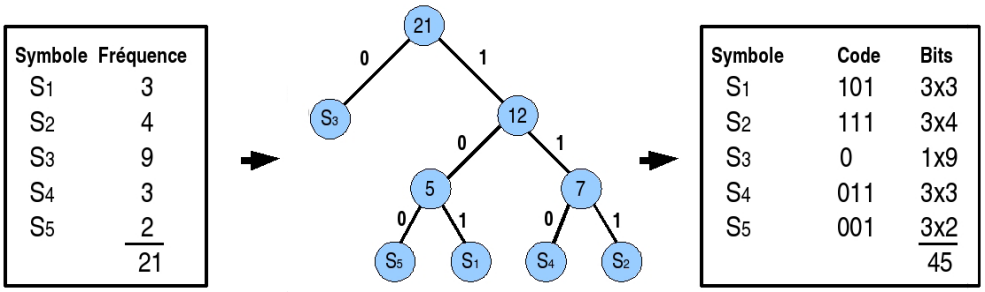


Figure 1.1 : Algorithme d'Huffman

1.2.1.2 Codage arithmétique

Le codage arithmétique (CA) [Pau 94] est un codage statistique qui attribue à une suite de symboles une valeur réelle. Il consiste à découper l'intervalle des réels [0, 1) en sous intervalles, dont les longueurs sont fonctions des probabilités des symboles. Le codage arithmétique n'attribue pas un code à chaque symbole comme Huffman et les autres codages par blocs, mais un code au message tout entier. Les tableaux suivants présentent un exemple de codage arithmétique avec le message AAOEU.

Alphabet	Probabilité	Probabilité Cumulée	Partition Initiale
A	**0,2**	**0,2**	**[0 0,2)**
E	**0 ,4**	**0,6**	**[0,2 0,6)**
I	**0,1**	**0,7**	**[0,6 0,7)**
O	**0,2**	**0,9**	**[0,7 0,9)**
U	**0,1**	**1,0**	**[0,9 1,0)**

Message	Gauche G	Taille T	Droite D
A	**0,0000**	**0,2000**	**0,2000**
A	**0,0000**	**0,0400**	**0,0400**
O	**0,0280**	**0,0080**	**0,0360**
E	**0,0296**	**0,0032**	**0,0328**
U	**0,0325**	**0,0003**	**0,0328**

Tableau 1.1 : Codage arithmétique

Soit l'alphabet {A,E,I,O,U} avec les probabilités {0,2 0,4 0,1 0,2 0,1}. Le codage arithmétique est fait à partir de l'intervalle initial [0, 1) et au fur et à mesure du codage, la longueur de l'intervalle diminue en tenant compte du sous-intervalle précédent.

Nous nous servons des formules (G = GPI +GM_TPI) et (T = TPI _TM) pour construire le nouvel intervalle où les lettres signifient : G-gauche, T-taille, M-message et P-précédant.

Le premier symbole A du message réduit l'intervalle initial à [0 0,2). Le deuxième symbole A du message réduit ce dernier intervalle à [0 0,04) (1/5 de l'intervalle précèdent).

Le symbole O réduit l'intervalle à [0,028 0,036). Le symbole E diminue l'intervalle à [0,0296 0,0328). Enfin, le symbole final U réduit à [0,03248 0,0328). Finalement, tout réel dans l'intervalle [0,03248 0,0328) codera le message AAOEU. Le codage arithmétique est présent dans la norme JPEG (dans les modes Extended DCT-based processes et Lossless processes) et JPEG2000.

Les méthodes de codage statistiques construisent les mots-codes à partir d'un dictionnaire prédéfini, basé sur les statistiques de l'image elle-même. Ce dictionnaire est indispensable pour le décodage. Des nouvelles études et améliorations pour cette approche ont été proposées, nous citons les travaux [Fon 02] [Guy 01] [Shi 01].

Les deux méthodes que nous allons présenter, codage par substitution, n'exigent pas de connaissance à priori de l'image, comme les probabilités d'apparition des pixels par exemple. Elles construisent des dictionnaires dynamiques dont les mots-codes créés sont indépendants de la source.

1.2.1.3 L'algorithme LZW

Lempel et Ziv [Ziv 77] ont présenté un schéma (LZ77) qui est à la base de tous les algorithmes à dictionnaire dynamique utilisés actuellement. Welch a amélioré leur algorithme et a déposé un brevet en créant l'algorithme LZW qui génère un dictionnaire dynamique qui contient des motifs du fichier. L'utilisation d'un dictionnaire dynamique a réglé le problème de le transmettre a priori pour les procédés de compression et décompression. Il est basé sur la multiplicité des occurrences de séquences de symboles.

Son principe consiste à substituer des motifs par un code d'affectation en construisant au fur et à mesure un dictionnaire. Celui-ci est initialisé avec les valeurs de la table ASCII. Chaque octet du fichier est comparé au dictionnaire. S'il n'existe pas, il est ajouté au dictionnaire. L'algorithme LZW fait partie du format d'image, aussi breveté, GIF (Graphics Interchange Format).

1.2.1.4 Codage par plage

Le codage par plage ou RLE Run Length Encoding est recommandé lorsque nous observons des répétitions de symboles consécutifs. Il est utilisé par de nombreux formats d'images (BMP, TIFF, JPEG) [Ric 01]. L'idée est de regrouper les pixels voisins ayant la même couleur. Chaque groupement définit un couple de valeurs

P = (plage, n) où plage est le nombre de points voisins ayant la même valeur, et n est cette valeur. Le RLE est d'autant plus performant que les groupements sont étendus, il n'est pas applicable dans tous les cas. Il est recommandé pour les images avec de larges zones uniformes. La compression d'une image peut être effectuée de manière adaptative : dans les régions uniformes le RLE est appliqué, et dans les zones non uniformes des règles particulières sont créées. Par exemple, au moins trois éléments se répètent consécutivement alors la méthode RLE est utilisée, sinon un caractère de contrôle est inséré, suivi du nombre d'éléments de la chaîne non compressée. D'autres caractères de contrôle peuvent aussi être utilisés pour définir la fin de ligne ou la fin de colonne.

1.2.1.5 Codage par prédiction linéaire

Les algorithmes qui utilisent le codage par prédiction exploitent la redondance spatiale. Il s'agit de prédire la valeur d'un pixel en fonction de la valeur des pixels voisins et de ne coder que l'erreur de prédiction. Le gain en compression est accompli par la variation faible entre pixels voisins, sauf pour les pixels situés sur les contours. Le voisinage peut être défini selon sa connexité (4-connexité ou 8-connexité) ou selon l'ordre du parcours choisi pour accéder aux pixels voisins. L'une des techniques de prédiction la plus simple est la DPCM (Differential Pulse Code Modulation) [Mor 95]. Cette technique effectue une prédiction à base d'une combinaison linéaire des valeurs des pixels voisins. Une version adaptative, ADPCM, qui utilise différentes formes de prédiction et de voisinage selon le contexte et le contenu de l'image a été présentée par Kyung et al. [Kyu 96]. Récemment, Babel et al. [Bab 03] ont proposé un codage progressif et multi résolution LAR - (Locally Adaptive Resolution). Il s'agit d'un codeur qui associe le DPCM à une décomposition multicouches suivi d'une transformée Mojette3. La profondeur de la décomposition détermine le type de compression, avec pertes ou sans perte pour le huitième niveau.

1.2.2 Méthodes avec pertes

Les méthodes avec pertes (lossy) ou irréversibles sont des méthodes qui tirent parti d'une corrélation (ou redondance) existante dans l'image. L'information perdue est due à l'élimination de cette redondance, ceci rend possible une compression plus importante.

La perte d'information est toujours discutable et nous nous posons alors la question de la limite acceptable. Cette limite est définie par le type d'application, comme les images médicales ou satellites par exemple. La quantification est un des mécanismes utilisé dans les algorithmes de compression, qui produit des pertes d'information.

1.2.2.1 Quantification

La quantification fait partie de plusieurs méthodes de compression d'image. L'objectif est de réduire la taille des coefficients de façon que cette réduction n'apporte pas de dégradations visuelles à l'image.

1.2.2.2 Scalaire

La quantification scalaire SQ - (Scalar Quantization) est une procédure qui associe à une variable continue X une variable discrète **x**. Pour cela on associe à **x** la valeur quantifiée $\mathbf{x_q}$=Q(X), où Q est une fonction (non linéaire) de quantification de R→Z

La quantification scalaire utilisée, en pratique, dans les images sont des quantifications basées en zone morte dans lesquelles l'intervalle de quantification est centré à l'origine et est de taille multiple de la taille des autres intervalles de quantification [Jos 06].

1.2.2.3 Vectorielle

Le principe de la quantification vectorielle VQ - (Vector Quantization) est issu du travail de Shannon qui montre qu'il était toujours possible d'améliorer la compression de données en codant non pas des scalaires, mais des vecteurs. Un quantificateur vectoriel Q associe à chaque vecteur d'entrée Xi = (x_j, j = 1 . . . k) un vecteur Yi = (y_j, j = 1 . . . k) = Q(Xi), ce vecteur Yi étant choisi parmi un dictionnaire (code-book) de taille finie. La VQ produit de meilleurs résultats que la SQ, néanmoins la VQ nécessite un codage complexe et de grandes capacités de mémoire.

1.2.2.4 Codage prédictif avec pertes

Il existe des techniques qui exploitent la redondance spatiale, cependant la prédiction est faite par approximation. Ces algorithmes ont comme objectif de rechercher un modèle de représentation le plus adéquat de l'information à coder

afin d'obtenir un coût de codage minimal. L'idée est de coder l'erreur de prédiction au-dessus d'un seuil. Ce seuil peut être défini par rapport à la qualité de l'image ou le niveau de compression espéré.

1.2.2.5 Codage par transformation

Les méthodes qui utilisent cette technique utilisent des transformations pour produire une décorrélation des redondances spectrales. Les pixels passent d'un espace où ils sont fortement corrélés dans un autre espace où leur corrélation est moindre. Lors de chaque transformation, le signal d'origine est remplacé par sa représentation dans un autre domaine. Dans divers algorithmes cette transformation d'espace est accompagnée d'une quantification et d'un codage entropique pour accomplir la compression de l'image. Ceci est le cas des normes standards de compression : l'algorithme JPEG qui utilise la transformation type DCT et l'algorithme JPEG2000 qui utilise la transformation en ondelettes DWT.

1.2.2.6 Domaine spatial

Les méthodes de compression d'images dans le domaine spatial exploitent la redondance entre un pixel et son voisinage, ou entre certaines régions de l'image.

1.2.2.7 Quadtree

La technique de décomposition par Quadtree est basée sur une approche récursive d'un codage arborescent. Une image hétérogène de taille $2n \times 2n$ est alors divisée en quatre sous-régions, de taille $2n-1 \times 2n-1$. Ce processus est refait jusqu'à ce que chacune des régions soit déclarée homogène selon un critère choisi. Les zones considérées homogènes seront découpées en grands blocs alors que les zones texturées ou contenant des contours seront découpées en blocs plus petits comme le montre la figure 1.2.

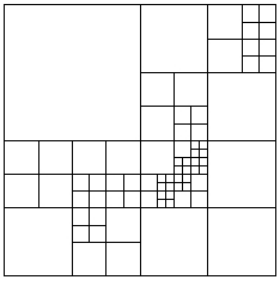


Figure 1.2 : Décomposition en Quadtree

1.2.2.8 Décomposition en plans binaire

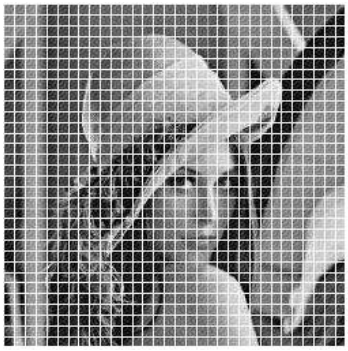
Cette technique est principalement employée pour les images en niveau de gris. L'idée est de décomposer l'image en huit images binaires, une pour chaque bit de niveau de gris, en commençant par les bits de poids les plus forts, voir figure 1.3.

Figure 1.3 : Décomposition en plans binaires des bits de poids forts jusqu'aux bits de poids faibles.

Sur chaque image binaire sont lancées des procédures particulières pour effectuer la compression. L'image binaire représentant le bit de poids le plus fort, première image (8), apporte le plus haut taux de compression, tandis que la compression sur l'image binaire de poids le plus faible, dernière image (1) a quasiment une compression nulle.

1.2.2.9 Fractales

La compression par fractale est une technique de compression avec pertes encore peu utilisée. Une fractale est une structure géométrique qui se reproduit, dans une boucle infinie, par transformation affine (translation, rotation et mise à l'échelle) [Fis 95]. Cette structure se refait à toutes les échelles de forme réduite et légèrement déformée. La compression par fractale est basée sur le principe qu'il existe des similarités entre différentes régions isolées d'image. Elle exploite les récurrences des motifs qui, après quelques traitements, peuvent permettre une compression. La figure 1.4 présente un exemple d'exploitation des motifs.



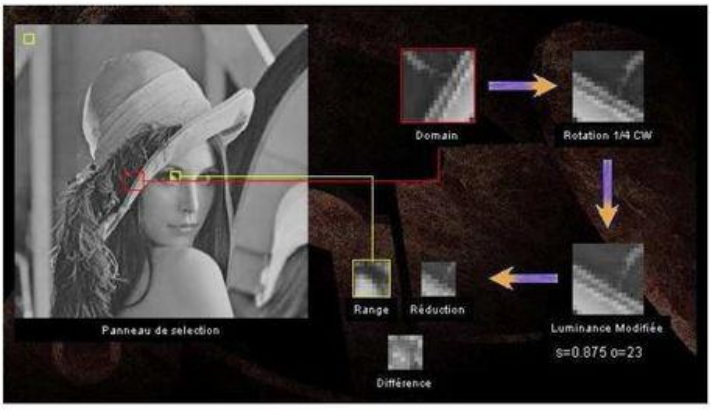


Figure 1.4 : Compression par fractales.

1.2.2.10 Domaine fréquentiel

Les techniques de compression dans le domaine fréquentiel s'appuient sur une transformation de l'image vers un nouvel espace de représentation d'énergie fortement décorrélée. Cette décorrélation provoque une nouvelle représentation de l'image par la redistribution de l'énergie dans un nombre restreint de coefficients transformés. Cette énergie de l'image transformée est distribuée sous la forme de tranches énergétiques de basse, moyenne et haute intensités. Les transformations d'espace les plus courantes sont la DCT détaillés dans l'annexe A et la DWT.

1.2.2.11 Les standards

Il existe plus d'une cinquantaine de types de formats d'image [Bab 03]. Pour chacun d'entre eux la structuration des données et les attributs sont différents. La standardisation d'un format d'image permet de régler l'utilisation, la divulgation et la production de logiciels et de hardware compatibles avec le format standard. Le format standard JPEG est le format d'image le plus populaire, et il est devant la scène depuis quelques années. Son successeur, le JPEG2000, semble s'établir dans le domaine de l'image numérique. Le JPEG2000 possède des fonctionnalités supplémentaires par rapport au format JPEG. Cependant, la plupart des appareils numériques (appareils photos, caméscopes, téléphones portable, etc.) et les logiciels qui capturent et traitent les images sont au format JPEG.

1.2.2.11.1 JPEG [Jos 06]

Le comité Joint Photographic Expert Group a été créé en 1986 par la jonction (Joint) de plusieurs groupes qui travaillaient sur la photographie. Ce comité a produit la norme de compression d'images photographiques qui a été standardisée (ISO/IEC/10918-1/1994) et a reçu son nom JPEG. Il est devenu le format le plus populaire très rapidement parce qu'il a été conçu avec différentes contraintes :

- L'algorithme JPEG doit être implémenté sur une grande variété de types de CPU (unité centrale de calcul) et sur des cartes plus spécialisées (appareil photo numérique et téléphone portable par exemple).
- Il doit pouvoir compresser efficacement tout type d'images réelles (images photographiques, médicales) avec pertes et sans perte.
- Il possède quatre modes de fonctionnement : séquentiel (Baseline), progressif (Extended DCT-based), sans perte Lossless, hiérarchique hierarchical.

Entre les 4 modes de compression de la norme JPEG, le séquentiel ou Baseline est le mode principal le plus répandu. Il est basé sur la transformation DCT, quantification scalaire et le codage d'Huffman sur pixels de 8 bits par plan de couleur. La figure 1.5 expose une synthèse du mode séquentiel.

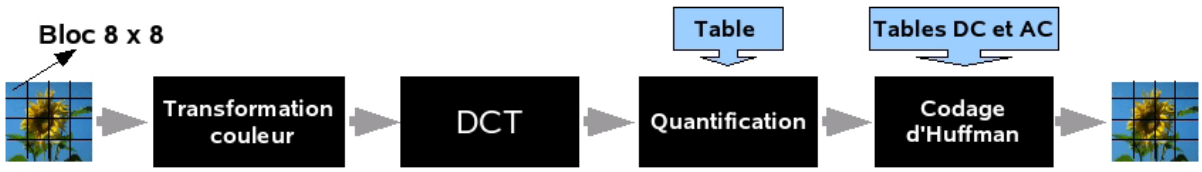


Figure 1.5: Compression JPEG.

- **Transformation d'espace couleur**

  Tout d'abord, l'image originale est soumise à un changement d'espace couleur YCbCr5. Les informations de chrominance, les plans Cb et Cr, sont sous-échantillonnées. Après cette intervention, les deux plans auront deux fois moins de lignes et de colonnes. Cette opération de sous-échantillonnage est fondée sur le principe que le SVH ne peut discerner des différences de chrominance au sein d'un carré de 2×2 points. Après le

sous-échantillonnage, chaque plan Y, Cb et Cr est traité de la même manière. Tout d'abord, ils sont découpés en blocs de 8 × 8 pixels.

- **DCT**

  Chaque bloc 8×8 est soumis à une transformation par DCT. Le premier coefficient de la DCT, le DC, est proportionnel à la moyenne des valeurs du bloc. Les 63 autres coefficients sont appelés AC. Ce nouveau domaine transformé permet une décorrélation très forte de l'information. Sur ce bloc de coefficients, les énergies sont groupées en basse (zone homogène de l'image originale), moyenne et haute fréquences (zone texturée ou zone de contour). Avec la DCT, chaque colonne est une fonction cosinus de fréquence différente. La variance est alors concentrée sur les composantes de basse fréquence, et les composantes de haute fréquence seront annulées par quantification.

- **Quantification**

  La compression avec pertes est faite dans l'étape de quantification qui est réalisée à l'aide d'une matrice Q de quantification de 8 × 8 éléments. L'image subit des distorsions selon le niveau de compression désiré. Chaque coefficient DCT est divisé par la valeur correspondante dans Q et le résultat est arrondi à l'entier le plus proche. L'acuité du SVH est plus faible à des hautes fréquences et plus sensible aux basses fréquences. Quelques tables standards, pour la quantification, ont été générées grâce à une série des caractéristiques psychovisuelles. Les valeurs prennent donc en compte cette caractéristique et introduit majoritairement de la distorsion dans les hautes fréquences.

- **Codage d'Huffman**

  Le codage entropique dans le mode séquentiel est un codage du type RLE. Après la quantification un grand nombre de coefficients sont nuls ou très proches de zéro. Le coefficient DC (composante continue) est codé séparément par rapport aux AC. Les coefficients AC sont parcourus en zigzag, figure 8, et sont codés par des couples (HEAD), (AMPLITUDE). L'entête HEAD contient des contrôleurs qui seront utilisés pour accéder aux tables d'Huffman. Le paramètre AMPLITUDE est un entier signé

correspondant à l'amplitude du coefficient AC non nul. La structure HEAD varie en fonction du type de coefficient. Pour les AC elle est composée de (RUNLENGTH, SIZE), alors que pour les DC elle est composée seulement de la taille SIZE. Les coefficients DC transportent une information visible importante et une corrélation locale significative. Ils sont hautement prédictibles, ainsi JPEG traite les coefficients DC séparément des 63 coefficients AC. La valeur des composants DC est importante et variée, mais est souvent très proche de celle de ses voisins. La seule valeur qui est donc encodée est la différence DIFF entre le coefficient DC_i quantifié du bloc courant et le précédent DC_{i-1}.

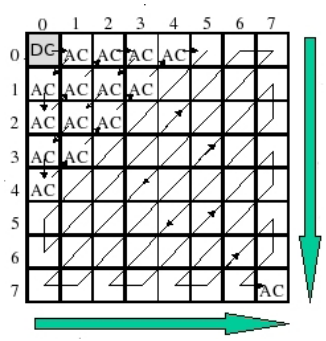


Figure 1.6 : Bloc DCT et parcours zigzag.

1.2.2.11.2 JPEG2000

Le JPEG2000 [Pau 94] remplace le JPEG comme le format standard pour la compression des images. Il a été réalisé dans la perspective de répondre aux exigences des nouvelles applications les plus diversifiées, comme la multi résolution par exemple. La compression JPEG2000 est composée de plusieurs étapes selon les schémas avec pertes et sans perte. Tout d'abord, un changement d'échelle est effectué dans chaque composante couleur RGB, l'échelle est changée de (0, 255) à l'échelle (-128, 127) par une simple soustraction de 128 de chaque valeur. Après le changement d'échelle, l'image est soumise à une transformation de plans couleurs, facultative, de RGB à YCbCr. Cette transformation peut être réversible (sans perte) ou irréversible (avec pertes). Chaque plan chromatique de l'image est découpé en petites images appelées tuiles. Chaque tuile est considérée comme une image et est traitée de façon indépendante. Du fait de la complexité du mécanisme du JPEG2000, la décomposition de l'image en tuiles rend possible l'application du JPEG2000 sur des images de taille importante. La

transformation de domaine est faite à l'aide d'une décomposition en ondelettes par schéma lifting. Chaque tuile de chaque composante subit des transformations selon les types compressions réversible et irréversible. Pour la compression réversible la norme utilise la paire (5,3) de LeGallles. Les coefficients des filtres d'analyse sont entiers et composés de 5 coefficients pour le filtre passe-bas et de 3 pour le passe-haut. Pour la compression irréversible est utilisée la paire (9,7) de Daubechies qui utilise des coefficients réels, avec 9 coefficients pour le passe-bas et 7 pour le passe-haut. Les lignes et les colonnes sont récursivement décomposées générant quatre sous-bandes pour chaque niveau de décomposition. Ceci produit des sous-bandes LL (basses fréquences horizontales et verticales), LH (basses fréquences horizontales et hautes fréquences verticales), HL (hautes fréquences horizontales et basses fréquences verticales) et HH (hautes fréquences horizontales et verticales).

Ensuite, une étape de quantification est faite selon le taux de compression. La quantification est faite pour rendre nuls les coefficients les plus faibles et faciliter le codage des autres. Chacune des sous-bandes peut-être quantifiée avec des étapes différentes. Plusieurs techniques permettant la quantification des sous-bandes ont été proposées, linéaire, non linéaire, avec masquage en fonction des voisins ou codage en treillis (TCQ), mais seule la quantification linéaire a été retenue par la norme.

Avant le codage entropique, les coefficients quantifiés sont divisés en plusieurs blocs appelés code-blocks de taille 64x64 ou 32x32. Le codage entropique du JPEG2000 est du type arithmétique adaptatif avec contexte, EBCOT - (Embedded Block Coding with Optimized Truncation) [Pau 94]. Le codage d'un bloc consiste à parcourir les coefficients du code-block par plan de bits, du bit de poids le plus fort (MSB) au bit de poids le plus faible (LSB). Les bits sont séparés en trois groupes en fonction de leur voisinage. Ensuite, ils sont codés en trois passes (coding passes). Le codage débute lorsqu'un plan de bits devient significatif (signifiant).

La résistance aux erreurs est une caractéristique particulière du JPEG2000. Après le codage entropique plusieurs caractères de contrôle (segment marks, resynchronising marks) sont insérés dans le flux de bits. Cette démarche est faite

pour synchroniser les informations, limiter la taille du segment et éviter la propagation des erreurs.

Une autre fonctionnalité importante du JPEG2000 est la compression par région d'intérêt (ROI). Ceci permet d'avoir des taux de compression différents dans certaines régions de l'image. Les zones importantes peuvent être compressées quasi sans pertes et les zones moins importantes avec un fort taux de compression.

Malgré ces nombreuses fonctionnalités, le JPEG2000 possède quelques inconvénients. Il nécessite entre deux et six fois plus de cycles de CPU que JPEG6 et il n'est pas indiqué pour les machines avec faibles ressources comme les appareils photos numériques par exemple. L'algorithme JPEG est beaucoup moins complexe et il peut être implémenté en hardware.

1.2.2.12 Compression par Ondelettes

La compression par ondelettes est une technique relativement nouvelle, La transformation par Ondelettes est une technique inventée par Summus Ltd et qui consiste à décomposer une image en une myriade de sous-bandes, c'est à dire des images de résolution inférieure [Sal 07]. On distingue 4 étapes différentes pour procéder à la transformation:

1. Moyenner les pixels de l'image originale deux à deux suivant l'axe horizontal ; par exemple **: (H(x) = (X_n+X_n+1) / 2)**

2. Calculer l'erreur entre l'image originale et l'image sous-échantillonnées dans le sens horizontal; par exemple : (G(x) = (X_n-X_n+1) /2)

3. Pour chacune des deux images intermédiaires, moyenner les pixels deux à deux suivant l'axe vertical ; par exemple : (H(y) = (Y_n+Y_n+1) / 2)

4. Pour chacune des deux images intermédiaires, calculer l'erreur suivant l'axe vertical ; par exemple : (G(y) = (Y_n-Y_n+1) / 2)

Le résultat est une image d'approximation qui a une résolution divisée par deux et trois images de détails qui donnent les erreurs entre l'image originale et l'image d'approximation. Cette transformation est répétée autant de fois que nécessaire pour obtenir le nombre voulu de sous-bandes.

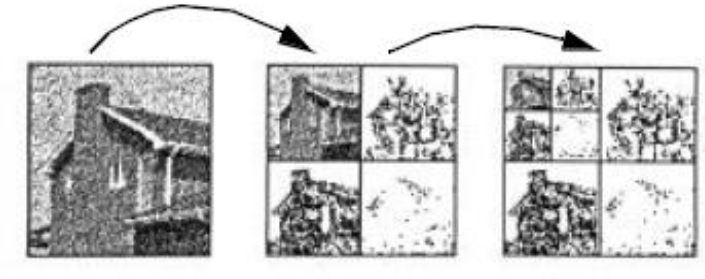


Figure 1.7 : Transformation par ondelettes

Il n'y a pas de pertes à ce stade de la transformation. Les pertes subviendront lors de la compression. Les étapes de compression sont les suivantes :

1. Transformations par ondelettes.
2. Quantification : les valeurs des images de détails inférieures à un certain niveau sont éliminées, en fonction de l'efficacité recherchée. C'est cette étape qui introduit des pertes.
3. Codage des valeurs restantes.

La transformation inverse par ondelettes reconstruit une image originale. La construction de l'image à partir des sous-bandes restitue l'image en mode progressif. L'affichage de l'image peut s'effectuer en deux modes :

- Soit la taille de l'image augmente au fur et à mesure de la lecture du fichier compressé.
- Soit la résolution de l'image augmente au fur et à mesure de la lecture du fichier compressé.

1.3 Conclusion

La compression consiste à réduire la taille des fichiers afin de faciliter leur transfert ainsi que limiter l'espace de sauvegarde de ces derniers.

Comme nous l'avons vu précédemment, il existe deux grandes familles de compression, celles qui restent fidèles au fichiers originaux et celles qui entrainent plus au moins des dégradations engendrant un effet non désirable lors de leur restauration.

On a vu aussi que quelques approches énumérées précédemment utilisent d'une manière ou d'une autre les corrélations entre pixels voisins dans l'image. Ces corrélations sont le fondement de la compression car elles sont liées à la notion de redondance.

L'idée maintenant est de savoir laquelle des méthodes citées convient le plus à un contexte donné. Cela n'est pas permis qu'on sachant le degré de conservation de la qualité d'image reconstruite.

D'autre part il est possible (pas toujours évident) de faire introduire des techniques d'optimisation sur quelques méthodes de compression qui présentent des pertes d'informations, afin de pallier aux différentes lacunes considérées par ces méthodes et de proposer des méthodes optimisées basées principalement sur ces techniques.

Le second chapitre présentera un axe très prometteur en optimisation des problèmes complexes.

Chapitre 2. Métaheuristiques et optimisation: état de l'art

« La vérité scientifique sera toujours plus belle que les créations de notre imagination et que les illusions de notre ignorance ».

Claude Bernard

2.1 Introduction

L'optimisation est un sujet capital dans la recherche opérationnelle, un grand nombre de problèmes pouvant en effet être transformés en des problèmes d'optimisation. Les problèmes d'identification, l'apprentissage supervisé de réseaux de neurones, problème du voyageur de commerce ou encore le problème de plus court chemin sont, par exemple, des problèmes d'optimisation.

Utilisées dans de nombreux domaines, ces méthodes présentent l'avantage d'être généralement efficaces, sans pour autant que l'utilisateur ait à modifier la structure de base de l'algorithme qu'il utilise.

Ces algorithmes stochastiques d'optimisation globale peuvent être appliqués à tout problème. Ce sont en général des problèmes aux données incomplètes, incertaines, bruitées ou confrontés à une capacité de calcul limitée. Ces algorithmes sont inspirés par la physique (recuit simulé, recuit micro canonique), la biologie (algorithmes évolutionnaires) ou l'éthologie (colonies de fourmis, essaims particulaires). Ils se prêtent aussi à toutes sortes d'extensions, notamment en optimisation multi objectif.

Ce chapitre présente un état de l'art des métaheuristiques d'optimisation pour les problèmes mono-objectif aussi bien que pour les problèmes multiobjectifs. Les métaheuristiques sont une famille d'algorithmes stochastiques. Apparues au

début des années quatre-vingts, elles sont destinées à résoudre des problèmes d'optimisation difficile.

2.2 Optimisation

Un problème d'optimisation en général est défini par un espace de recherche S et une fonction objectif *f*. Le but est de trouver la solution $\boldsymbol{s*} \in \boldsymbol{S}$ de meilleure qualité f ($\boldsymbol{s*}$). Suivant le problème posé, on cherche soit le minimum soit le maximum de la fonction *f*. L'équation (2.1) montre bien l'aspect de minimisation, maximiser une fonction *f* étant équivalent à minimiser *f*.

$$\boldsymbol{s} = \mathbf{min}(\boldsymbol{f}(\boldsymbol{s}) \setminus \boldsymbol{s} \in \boldsymbol{S}) \qquad (2.1)$$

De plus, un problème d'optimisation peut présenter des contraintes d'égalité et/ou d'inégalité sur les solutions candidates s ∈ S, être multiobjectif si plusieurs fonctions objectifs doivent être optimisées.

Il existe de nombreuses méthodes déterministes (exactes) qui permettent de résoudre certains problèmes d'optimisation en un temps fini. Néanmoins, ces méthodes sollicitent que la fonction objectif présente un certain nombre de caractéristiques, telles que la convexité, la continuité ou encore la dérivabilité.

Certains problèmes restent cependant trop compliqués à résoudre pour les méthodes exactes. Certaines caractéristiques peuvent poser des soucis pour ces méthodes. Dans ce cas, le problème d'optimisation est dit difficile, car aucune méthode déterministe ne peut résoudre ce problème en un temps raisonnable [Dré 04].

Les problèmes d'optimisation difficile se divisent en deux catégories : les problèmes à variables discrètes et les problèmes à variables continues. De façon générale, un problème d'optimisation à variables discrètes, ou combinatoire, consiste à trouver, dans un ensemble discret, la meilleure solution réalisable, au sens du problème défini par (2.1). Le problème majeur réside ici dans le fait que le nombre de solutions réalisables est généralement très élevé, donc il est très difficile de trouver la meilleure solution dans un temps raisonnable.

Les problèmes à variables continues sont, eux, moins formalisés que les précédents. En effet, la grande majorité des métaheuristiques existantes ont été créées pour résoudre des problèmes à variables discrètes [Sia et al 06].

Cependant, la nécessité croissante de méthodes pouvant résoudre ce type de problèmes a poussé les chercheurs à adapter leurs méthodes au cas continu. Les difficultés les plus courantes de ce type de problèmes sont les corrélations non identifiées entre les variables, le caractère bruité des fonctions ou encore des fonctions objectives qui ne sont accessibles que par dispositif expérimental. Il est à noter qu'il existe des problèmes à variables mixtes, c'est-à-dire que le problème présente à la fois des variables discrètes et continues.

2.3 Algorithmes d'optimisation

2.3.1 Heuristiques

On appelle une heuristique d'optimisation toute méthode approximative se voulant primitive, rapide et adaptée à un problème donné. Sa capacité à optimiser un problème avec un minimum d'informations est contredite par le fait qu'elle n'offre aucune assurance quant à l'optimalité de la solution trouvée. Du point de vue de la recherche opérationnelle, cette anomalie n'est pas toujours un obstacle, tout particulièrement quand une seule approximation de la solution optimale est recherchée [Dré 04].

2.3.2 Métaheuristiques

Certaines heuristiques sont applicables à de nombreux différents problèmes sans apporter des changements majeurs dans l'algorithme, c'est le cas des métaheuristiques. La plupart des heuristiques et des métaheuristiques utilisent des processus aléatoires comme moyens de récolter de l'information et de faire face à des problèmes comme l'explosion combinatoire. En plus de cette base stochastique, les métaheuristiques sont habituellement itératives, Elles tirent distinctivement leur gain de leur aptitude à éviter les optimums locaux, soit en acceptant une dégradation de la fonction objective au cours de leur progression, soit en utilisant une population de points comme méthode de recherche. Souvent inspirées de la physique, biologie et éthologie [Dré 04].

Finalement, on ne doit pas oublier qu'un des intérêts majeurs des métaheuristiques est leur facilité d'utilisation dans des problèmes palpables. Un des investissements des métaheuristiques est donc de faciliter le choix d'une méthode et de simplifier son réglage pour l'adapter à un problème d'optimisation donné.

2.3.2.1 Métaheuristiques pour l'optimisation mono-objectif

2.3.2.1.1 Algorithme de recuit simulé

La méthode du recuit simulé est inspirée de la métallurgie, où, pour atteindre des états de basse énergie d'un solide, il faut monter la température du solide à des valeurs très élevées, ensuite le laisser refroidir doucement. Ce processus est appelé "recuit". La métaheuristique est dite "recuit simulé" établie individuellement par Kirk Patrick et al, en 1983 [Kir 83], et Cerny en 1985 [Cer 85]. Elle repose sur la procédure de Metropolis [Met 53]. Cette procédure permet de sortir des minimums locaux avec une probabilité d'autant plus grande que la température T est élevée. Quand on atteint les très basses températures, les états les plus probables forment d'excellentes solutions. L'algorithme de Metropolis (Algorithme 2.1) échantillonne la fonction objective par le biais d'une distribution de Boltzmann de paramètre T. Le point décisif de l'algorithme est la loi de diminution de la température.

Initialiser un point de départ $x_0$et une température T
Pour *i = 1* à *n* faire
  Tant que x_i n'est pas accepté faire
    Si $\boldsymbol{f(x_i) \leq f(x_{i-1})}$: accepter x_i
    Si $\boldsymbol{f(x_i) > f(x_{i-1})}$:
    accepter x_i avec la probabilité $\boldsymbol{e^{-\frac{f(x_i)-f(x_{i-1})}{T}}}$
  Fin Tan que
Fin Pour

Algorithme 2.1 : Algorithme de Metropolis.

L'algorithme de recuit simulé est résumé dans l'Algorithme 2.2. Il est efficace et facile à s'adapter à un nombre important de problèmes. En contrepartie, cet algorithme présente l'inconvénient de disposer d'un nombre élevé de paramètres (température initiale, règle de décroissance de la température, durée des paliers de température, etc.) qui rendent les réglages de l'algorithme assez empiriques. L'autre inconvénient majeur de cette méthode est sa lenteur.

Définir la fonction objectif (f).
Choix des mécanismes de perturbation d'une configuration ΔS.
Tirer une configuration aléatoire S.
Initialiser la température (T_0).
Tant que Conditions d'arrêts pas satisfaites **faire**
  Tant que l'équilibre thermodynamique pas atteint **faire**
    Tirer une nouvelle configuration S'.
    Appliquer la règle de Metropolis.
    Si $f(S') < f(S)$
    $f_{min} = f(S')$
    $S_{opt} = S'$
    Fin de si
  Fin tant que
  Décroître la température.
Fin tant que
Afficher la solution optimale

Algorithme 2.2 : Algorithme de recuit simulé.

2.3.2.1.2 Recherche Tabou

L'algorithme de Recherche Tabou a été introduit par Glover en 1986 [Gol 86]. Le but de cette méthode est d'introduire aux méthodes de recherche locale un supplément d'intelligence. L'idée ici est d'intégrer une mémoire au processus de recherche permettant une recherche plus intelligente dans l'espace des solutions. Semblablement au recuit simulé, la recherche taboue fonctionne à une seule configuration courante mise à jour au cours des itérations successives. La nouveauté ici est que, pour échapper à une configuration déjà passée, on tient à jour une liste de déplacements non permis, appelée « liste tabou ». Cette liste contient m déplacements (t → s) qui sont les inverses des m derniers mouvements (s → t) effectués. L'algorithme modélise ainsi une forme de mémoire.

L'algorithme de recherche tabou peut être résumé par l'Algorithme 2.3.

Dans sa forme de base, l'algorithme présente l'avantage d'utiliser moins de paramètres que celui du recuit simulé. Néanmoins, l'algorithme n'étant pas toujours performant, il est souvent adéquat de lui joindre des processus

d'accroissement et/ou de discrimination, qui intègrent de nouveaux paramètres de contrôle [Gol 97].

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

Algorithme 2.3 : Algorithme de Recherche Tabou.

2.3.2.1.3 Algorithmes évolutionnaires

Ces méthodes, créés pendant les années 1950 [Fra 57], constituent une famille d'algorithmes qui s'inspirent de l'évolution biologique des espèces. L'idée ici est de s'inspirer de la théorie de Darwin de sélection naturelle pour résoudre des problèmes d'optimisation.

Courant les années 1970, avec l'invention des puissantes machines, de nombreuses tentatives ont été réalisées pour modéliser cette approche. Trois approches se sont détachées :

- Les politiques d'évolution [Rec 65] [Bey 01] qui ont été créées comme une méthode d'optimisation à variables continues ;
- La programmation évolutionnaire [Fog 66], qui tenait à faire évoluer les structures d'automates à états finis par des successions de mutations et de croisements ;
- Les algorithmes génétiques [Hol 73] [Hol 92] [Gol 89], qui ont été conçus pour résoudre des problèmes d'optimisation à variables discrètes.

Les méthodes évolutionnaires s'articulent toutes sur un modèle commun présenté par l'Algorithme 2.4. Les individus soumis à l'évolution sont des solutions possibles du problème posé. L'ensemble de ces individus constitue une population. Cette population évolue durant une succession d'itérations, appelées

générations. Au cours de chaque génération, une succession d'opérateurs est appliquée aux individus de la population pour engendrer une nouvelle population, en vue de la génération suivante. Chaque opérateur utilise un ou plusieurs individus de la population appelé(s) parent(s) pour engendrer de nouveaux individus, appelés enfants. A la fin de chaque génération, un certain nombre d'individus de la population sont remplacés par des enfants créés durant la génération. Un algorithme évolutionnaire dispose donc de trois opérateurs principaux : un opérateur de sélection, un opérateur de croisement et un opérateur de mutation.

- **L'opérateur de sélection** consiste à favoriser la propagation des meilleures solutions parmi la population, tout en préservant la diversité génétique, afin d'explorer de nouvelles régions de l'espace de recherche. Plusieurs opérateurs de sélection ont été proposés dans la littérature [Sia et al 06], comme par exemple, celui proposé par Goldberg [Gol 89] qui sélectionne les individus proportionnellement à leur performance.
- **L'opérateur de croisement**, appelé "*crossover"*, est mis en œuvre dans la phase de recombinaison. Le but de cette opération est de maintenir la diversité en manipulant les composantes des individus (chromosomes).

  Une fois la sélection effectuée, de nouveaux individus, appelés *enfants*, sont créés par croisement, c'est-à-dire en échangeant des parties des individus de la population (*parents*). Cet échange se fait en sélectionnant un point de découpage, aléatoirement (avec une densité de probabilité uniforme), et en échangeant ensuite les deux sous chaînes de chacun des deux parents. Considérons, par exemple, deux chaines de parents A et B :

  A= 1 1 0 : 1 0

  B= 0 1 1 : 0 1

  et supposons que le point de découpage est celui indiqué par (:). Le résultat du croisement permet d'avoir deux nouveaux individus faisant partie de la nouvelle génération :

  A'= 1 1 0 : 0 1

  B'= 0 1 1 : 1 0

Cette procédure peut s'étendre à un découpage comportant plus de deux parties. Le croisement à découpage est très rapide à mettre en œuvre dans le cas où le problème utilise des entiers.

- **L'opérateur de mutation** consiste à tirer aléatoirement un gène (par exemple, un bit, dans le cas du codage binaire) dans le chromosome et à le remplacer par une valeur aléatoire. La mutation est l'opérateur qui apporte aux algorithmes génétiques l'aléa nécessaire à une exploration efficace de l'espace de recherche. Elle garantit la possibilité d'atteindre tout l'espace d'état. Ainsi l'algorithme génétique peut converger sans croisement. L'utilisation de la mutation, en tant qu'opérateur de recherche locale, suggère de le combiner avec d'autres techniques de recherche locale. Dans ces cas, les algorithmes génétiques (AG) sont dits *hybrides*.
- **L'élitisme** consiste à conserver au moins un individu de la population de la génération précédente, dans la génération suivante. L'algorithme évolutionnaire génétique se présente dans l'Algorithme 2.4.

Initialisation de la population de _ individus

Évaluation des μ individus

Tant que le critère d'arrêt n'est pas atteint **faire**

  Sélection de λ individus en vue de la phase de reproduction

  Croisement des λ individus sélectionnés

  Mutation des λ individus sélectionnés

  Évaluation des λ' enfants obtenus

  Sélection pour le remplacement

Fin Tant que

Algorithme 2.4 : Algorithme évolutionnaire générique.

2.3.2.1.4 Algorithme de colonies de fourmis

Cette métaheuristique est inspirée des comportements collectifs de dépôt et de suivi de pistes observés dans les colonies de fourmis. Un tel comportement est possible car les fourmis communiquent entre elles de manière indirecte par le dépôt de substances chimiques, appelées *phéromones*, sur le sol. Ce type de

communication indirecte est appelé *stigmergie,* résolvant ainsi naturellement des problèmes complexes. La figure 2.1 illustre bien ce comportement.

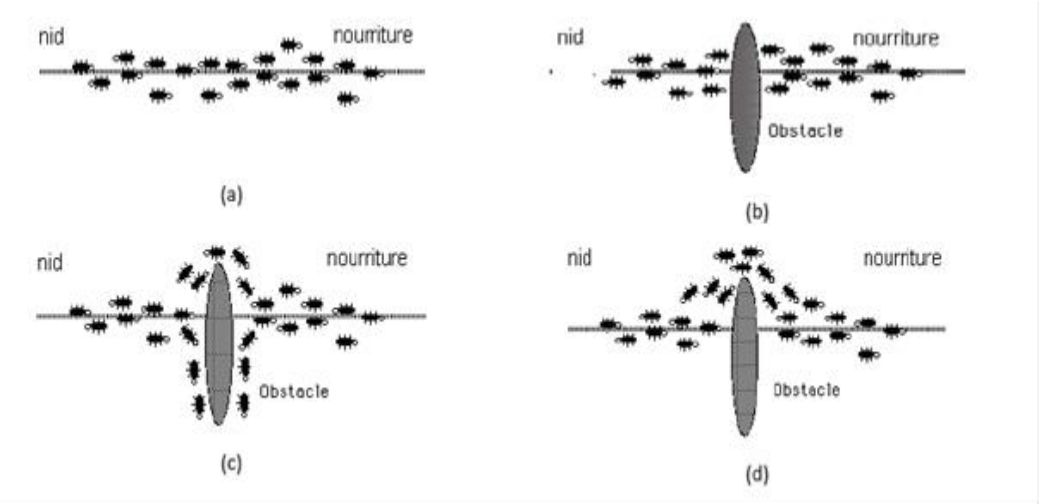


Figure 2.1 : Comportement naturel des fourmis pour chercher de la nourriture en minimisant leur parcours.

(a) Recherche initiale, (b) Découverte d'un obstacle, (c) Recherche du plus court chemin, (d) Plus court chemin trouvé [Dor 06].

Au départ, les fourmis partent de leur nid en direction de la nourriture (figure 2.1 (a)). Dès la découverte de l'obstacle, les fourmis modifient leur trajectoire pour le contourner (figure 2.1 (b) et (c)). La colonie se divise alors en deux groupes, le premier groupe choisit le plus long chemin, tandis que le second choisit le plus court. Puisque toutes les fourmis déposent leurs phéromones sur leur trajectoire, le chemin le plus court finira par avoir la densité de phéromones la plus élevée. Par conséquent, le nombre de fourmis suivant cette trajectoire augmente. Au fil du temps, la quantité de phéromones déposée sur le plus long chemin diminue et finit par disparaître ; toutes les fourmis suivent alors le chemin le plus court (figure 2.1 (d)).

Le premier algorithme d'optimisation par colonies de fourmis ("*Ant colony optimization*", ACO) a été proposé par Colorni, Dorigo et Maniezzo [Dor et al 92] [Dor et al 96]. Le premier algorithme, appelé "*Ant System*" (AS), a été développé spécialement pour résoudre le problème du voyageur de commerce (Algorithme 2.5).

Tant que le critère d'arrêt n'est pas atteint **faire**
  Pour *k=1* à *m* **faire**
    Choisir une ville au hasard
      Pour chaque ville non visitée *i* **faire**
        Choisir une ville j, dans la liste J_i^k des villes restantes selon (2.2)
      Fin Pour
    Déposer une piste $\Delta \boldsymbol{\tau_{ij}^k(t)}$ sur le trajet $\boldsymbol{T^k}$ (t) conformément à (2.3)
  Fin Pour
  Évaporer les pistes selon (2.4)
Fin Tant que

Algorithme 2.5 : Algorithme de colonies de fourmis pour le problème du voyageur de commerce.

Si l'on considère un problème de voyageur de commerce à *N* villes, à chaque itération *t*, chaque fourmi *k* parcourt le graphe et construit un trajet complet de *n* étapes (n = Card(*N*)). Pour chaque fourmi, le trajet entre une ville *i* et une ville *j* dépend de :

- La liste des villes déjà visitées. Elle définit les mouvements possibles à chaque pas, quand la fourmi *k* est sur la ville $\boldsymbol{i}$: $\boldsymbol{J_i^k}$
- La ***visibilité*** de chaque fourmi est définie par l'inverse de la distance entre les villes : $\boldsymbol{\eta_{ij}} = \frac{\mathbf{1}}{\boldsymbol{d_{ij}}}$ Ce qui permet aux fourmis de se déplacer vers les villes les plus proches.
- L'***intensité de la piste*** correspond à la quantité de phéromone déposée sur le chemin reliant deux villes. Ce qui définit une pseudo mémoire globale du système.

La règle de déplacement est définie par la formule : [Dor et al 05]

$$\boldsymbol{P\left(C_i^j \middle| S^p\right)} = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\Sigma_{l \in J_i^k}[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}} & \boldsymbol{si\ j \in J_i^k} \\ \mathbf{0} & \boldsymbol{si\ j \notin J_i^k} \end{cases} \qquad (2.2)$$

Où α et β sont deux paramètres contrôlant l'importance de l'intensité de la piste$\boldsymbol{\tau_{ij}(t)}$, et la visibilité$\boldsymbol{\eta_{ij}}$. A la fin d'un tour complet, la quantité de

phéromone $\Delta \boldsymbol{\tau}_{ij}^{k}(\boldsymbol{t})$ laissée par chaque fourmi dépend de la qualité de la solution trouvée, et est donnée par la formule (2.3) :

$$\Delta \boldsymbol{\tau}_{ij}^{k}(\boldsymbol{t}) = \begin{cases} \frac{\boldsymbol{Q}}{\boldsymbol{L^{k}(t)}} & \boldsymbol{si}\ (\boldsymbol{i},\boldsymbol{j}) \in\ \boldsymbol{T^{k}}\ (\boldsymbol{t}) \\ \boldsymbol{0} & \boldsymbol{si}\ (\boldsymbol{i},\boldsymbol{j}) \notin\ \boldsymbol{T^{k}}\ (\boldsymbol{t}) \end{cases} \qquad (2.3)$$

Où T^{k} (t) est le trajet effectué par la fourmi k à l'itération t, L^{k} (t) est la longueur du tour et Q un paramètre fixé. Pour que l'algorithme évite les pièges des solutions locales, une mise à jour des pistes est effectuée (2.4):

$$\boldsymbol{\tau_{ij}(t+1) = (1-\rho).\tau_{ij}(t) + \Delta\tau_{ij}(t)} \qquad (2.4)$$

Où $\boldsymbol{\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t)}$ et m est le nombre de fourmis. Au départ, la phéromone est initialisée par une petite quantité $\tau_0 \geq 0$.
La figure 2.2 illustre bien le principe de base d'un algorithme d'optimisation par la méthode des colonies de fourmis.

Depuis, la démarche initiée par cette analogie a été étendue à la résolution d'autres problèmes d'optimisation, discrets et continus [Dor et al 05] [Sia et al 06]. Les algorithmes de colonies de fourmis présentent plusieurs caractéristiques intéressantes, telles que le parallélisme intrinsèque élevé, la robustesse (une colonie peut maintenir une recherche efficace même si certains de ses individus sont défaillants) ou encore la décentralisation (les fourmis n'obéissent pas à une autorité centralisée). De ce fait, ces algorithmes semblent être idéalement adaptés aux problèmes dynamiques, pour lesquels seule une information locale est disponible [Tfa 07].

2.3.2.1.5 Optimisation par Essaim Particulaire

Kennedy et Eberhart en 1995 [Ken et al 95] ont introduit l'Optimisation par Essaim Particulaire (OEP) en s'inspirant du comportement social des animaux évoluant en essaim.

En effet, on peut observer chez ces animaux des déplacements dynamiques relativement complexes, alors qu'individuellement chaque individu a une intelligence limitée et une connaissance seulement locale de sa situation dans l'essaim. Un individu de l'essaim n'a pour connaissance que la position et la vitesse de ses plus proches voisins. Chaque individu utilise donc, non seulement,

sa propre mémoire, mais aussi l'information locale sur ses plus proches voisins pour décider de son propre déplacement.

L'algorithme commence avec une initialisation aléatoire de l'essaim de particules dans un espace de recherche de dimension *D*. Chaque particule est caractérisée par sa position dans l'espace de recherche et par sa vitesse. A chaque instant, toutes les particules ajustent leurs positions et vitesses, donc leurs trajectoires, par rapport (1) à leurs meilleures positions, (2) à la particule ayant la meilleure position dans l'essaim et (3) à leur position actuelle.

La position et la vitesse d'une particule dans l'espace de recherche sont définies par :

$\overrightarrow{P_i} = (p_{i1}, p_{i2}, \dots, p_{iD})$ et $\overrightarrow{V_i} = (v_{i1}, v_{i2}, \dots, v_{iD})$, respectivement. Chaque particule est caractérisée aussi par sa meilleure position $\overrightarrow{L_i} = (l_{i1}, l_{i2}, \dots, l_{iD})$à l'instant $\boldsymbol{t}$. La meilleure position qu'atteint l'essaim est sauvegardée dans le vecteur $\vec{g} = (g_1, g_2, \dots, g_D)$. La vitesse de chaque particule est mise à jour selon l'expression suivante [Cle et al 02] :

$$\boldsymbol{v_{ij}(t+1) = K\left[w.v_{ij}(t) + c_1.r_1.\left(l_{ij} - v_{ij}(t)\right) + c_2.r_2.\left(g_j - v_{ij}(t)\right)\right]} \quad (2.6)$$

Et

$$\boldsymbol{K = \frac{2}{\left|2-\varphi-\sqrt{\varphi^2-4\varphi}\right|}} \quad (2.7)$$

où $\boldsymbol{\varphi}$ = c1 + c2 et $\boldsymbol{\varphi} > 4$. Dans les travaux de M. Clerc, J. Kennedy [Cle et al 02], ils suggèrent les valeurs suivantes j = 1,..., D, w est une constante appelée facteur d'inertie, c1 et c2 sont des constantes appelées coefficients d'accélération, r1 et r2 sont des nombres aléatoires uniformément distribués dans l'intervalle [0,1]. Si la vitesse calculée fait sortir une particule de l'espace de recherche, sa fitnesse n'est pas calculée. Compte tenu de la nouvelle vitesse, obtenue à partir de (2.11) et (2.12), la position à l'itération t+1 est alors calculée :

$$\boldsymbol{P_{ij}\,(t+1) = P_{ij}(t) + v_{ij}(t+1) \quad pour\ j = 1; \dots; D} \quad (2.8)$$

Le paramètre d'inertie *w* contrôle l'influence de l'ancienne vitesse sur la vitesse courante, afin de permettre aux particules d'éviter les minimas locaux. De la même façon, *c*1 contrôle le comportement de la particule dans sa recherche autour de sa meilleure position et *c*2 contrôle l'influence de l'essaim sur le comportement de la particule. Les différentes étapes de l'algorithme sont

présentées sur la figure 2.3. La méthode est analysée avec plus de détails dans **[Cle et al 02].**

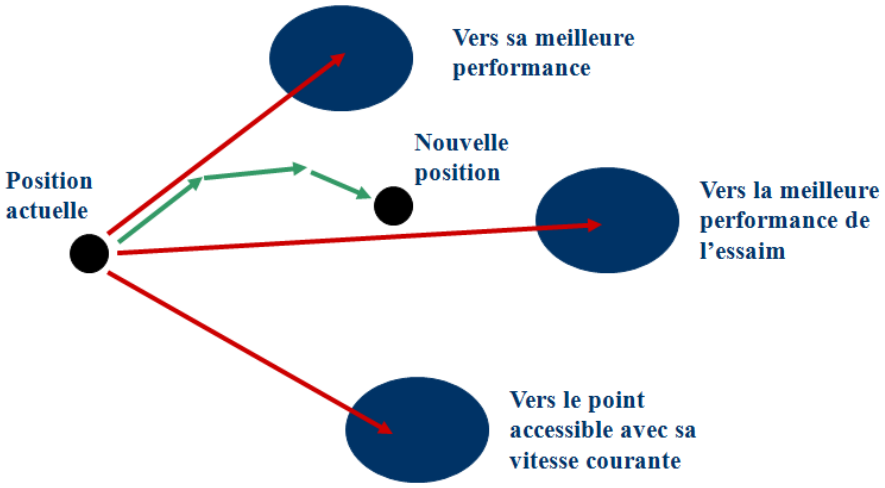


Figure 2.3 : Déplacement d'une particule.

Initialiser la population de particules avec des positions et vitesses aléatoires.

Evaluer la fonction objective pour chaque particule et calculer *g*.

Pour chaque individu *i*, *Li* est initialisée à *Pi*

Répéter jusqu'au critère d'arrêt

Mettre à jour les vitesses et les positions des particules.

Evaluer la fonction objective pour chaque individu.

Calculer les nouveaux *L*i et *g*.

Afficher le meilleur état rencontré au cours de la recherche.

Algorithme 2.6 : Algorithme d'optimisation par essaim particulaire.

2.3.2.2 Optimisation multiobjective

L'optimisation multiobjective est apparue dans le but de satisfaire plusieurs critères contradictoires (simultanément) en industrie. Ce type d'optimisation repose sur les bases posées par Pareto et Edgeworth [Par 1896].

2.3.2.2.1 Définitions

Un problème d'optimisation multiobjectif est un problème du type :

$$\text{Minimiser } \vec{f}(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), \dots f_k(\vec{x}))$$

Sous les contraintes suivantes :

$$3 \quad g_i(\vec{x}) \leq 0, \ i = 1,2, \dots, m$$
$$4 \quad h_i(\vec{x}) = 0, \ i = 1,2, \dots, p$$

Où $\vec{x} = (x_1, x_2, \dots, x_D)$ est une position dans l'espace de recherche, $f_i : \mathbb{R}^D \to \mathbb{R}$, $i = 1,2, \dots, k$ sont les fonctions objectifs du problème, $g_i : \mathbb{R}^D \to \mathbb{R}$, $i = 1,2, \dots, m$ sont les contraintes d'inégalité du problème et $h_i : \mathbb{R}^D \to \mathbb{R}$, $i = 1,2, \dots, p$ sont les contraintes d'égalité du problème.

Dans le cas multiobjectif, le concept d'optimum est différent que dans le cas mono-objectif. En effet, on n'est plus ici à la recherche d'un unique optimum global, mais plutôt d'une surface de solutions qui offrent un bon compromis entre les différents objectifs. Pour bien comprendre la notion d'optimalité dans le cas multiobjectif, on introduit les définitions suivantes, basées sur les travaux de Pareto [Par 1896]:

- **Définition 1** : Étant donné $\vec{x}, \vec{y} \in R^k$, on dit que $\vec{x} \leq \vec{y}$ si $x_i \leq y_i$ pour $i = 1,2, \dots, k$ et que $\vec{x}$ ***domine*** $\vec{y}$ (noté $\vec{x} \prec \vec{y}$) si $\vec{x} \leq \vec{y}$ et $\vec{x} \neq \vec{y}$.
- **Définition 2** : On dit qu'un vecteur solution $\vec{x} \in \chi \subset \mathbb{R}^D$ est ***non dominé*** dans χ s'il n'existe pas d'autre vecteur $\overrightarrow{x'} \in \chi$ tel que $\vec{f}(\overrightarrow{x'}) \prec \vec{f}(\vec{x})$.
- **Définition 3** : On dit qu'un vecteur solution $\overrightarrow{x*} \in \mathcal{F} \subset \mathbb{R}^D$ est ***Pareto optimal*** s'il est non dominé dans $\mathcal{F}$.
- **Définition 4** : l'ensemble de Pareto optimal est défini par :

$$P* = \{\vec{x} \in \mathcal{F} | x \ est \ Pareto \ optimal\}.$$

- **Définition 5** : le front optimal de Pareto est défini par :

$$\mathcal{F}_{p*} = \left\{\vec{f}(\vec{x}) \in \mathbb{R}^k \mid \vec{x} \in P*\right\}.$$

Le but d'un algorithme d'optimisation multiobjective est donc de trouver les positions de l'espace de recherche qui correspondent au front de Pareto $\mathcal{F}_{p*}$.

Les méthodes de résolution des problèmes multiobjectif se divisent en trois catégories : approches basées sur la transformation du problème en un problème mono-objectif (agrégation), approches non Pareto et approches Pareto.

2.3.2.2.2 Approche agrégative

Approche naïve de l'optimisation multiobjective, consiste simplement à transformer un problème multiobjectif en un problème mono-objectif [Col et al

02], ce dernier va subir l'une des méthodes déjà existantes pour sa résolution. Le but est de faire combiner plusieurs critères f_i en un seul critère$F$:

$$\boldsymbol{F}(\vec{x}) = \sum_{i=1}^{N} \boldsymbol{a_i f_i}\ (\vec{x}) \quad (2.9)$$

Où $\boldsymbol{a_i} \geq \mathbf{0}$; représentent les poids affectés différents critères de la fonction objectif, $\sum_{i=1}^{N} a_i = 1$ et N désigne le nombre de critères, En faisant varier les paramètres, différentes solutions supportées sont produites. La même solution peut être aussi produite en utilisant des paramètres différents.

La méthode agrégative a pour avantage la simplicité de la mise en œuvre ainsi elle ne fournit qu'une seule solution, ce qui évite l'intervention d'un décideur. Par contre on trouve deux problèmes à savoir la détermination des $\boldsymbol{a_i}$ et l'interaction entre les différents critères.

Le premier problème est résolu du fait qu'on doit donner plusieurs valeurs aléatoires aux paramètres, des stratégies « aveugles » ont été introduites pour modifier de façon aléatoire ces paramètres [Ish et al 98].

La figure 2.4 montre bien le fonctionnement de la méthode d'agrégation dans un cas bi-objectif L'espace des solutions réalisables est noté A. Le calcul des paramètres revient à trouver l'hyperplan tangent au front de Pareto (figure 2.4 (a)). La Figure 2.4 (b) illustre les limites de la méthode. Si le front de Pareto est concave, les solutions y et z peuvent être trouvées, alors que les autres points ne le seront jamais.

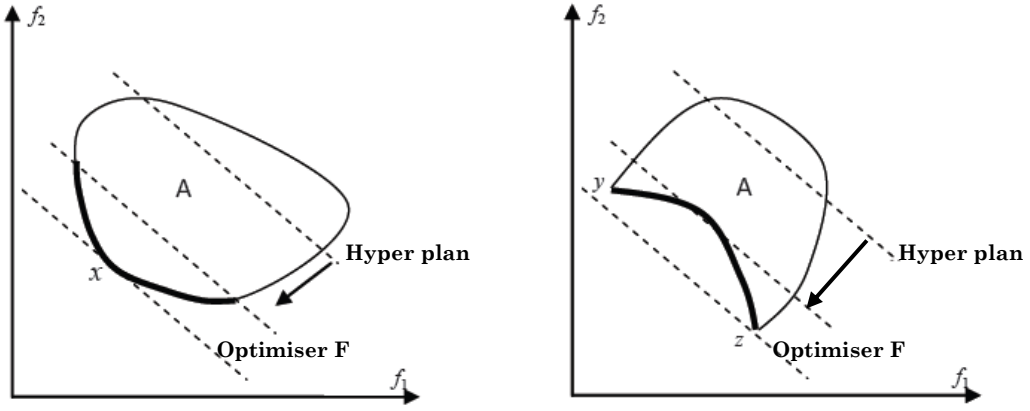


Figure 2.4 : Illustration de la méthode d'agrégation, dans un cas bi-objectif. (a) Une frontière de Pareto convexe, (b) Une frontière de Pareto non convexe. A : domaine réalisable, *x* : solution Pareto.

2.3.2.2.3 Approche non-Pareto :

Les méthodes basées sur une approche non-Pareto ont pour caractéristique de traiter les objectifs séparément. Deux groupes de méthodes existent : les méthodes à sélection lexicographique et les Méthodes à sélection parallèle.

Dans une approche classique de sélection lexicographique, les fonctions sont optimisées séquentiellement, suivant un ordre défini a priori. Cet ordre permet de définir le poids des objectifs. Plusieurs métaheuristiques ont été utilisées pour la résolution des problèmes multiobjectif à sélection lexicographique [Tal 99].

L'approche à sélection parallèle a été proposée par Schaffer dans [Sch 85]. Son algorithme, inspiré d'un algorithme évolutionnaire et nommé VEGA, sélectionne les solutions courantes du front de Pareto suivant chaque objectif, indépendamment des autres (d'où le parallélisme). L'analyse du comportement de cet algorithme a montré qu'il se comportait d'une manière similaire à un algorithme utilisant une méthode agrégative. D'autres travaux ont été menés sur ce sujet [Ric et al 89] [Ber 01].

2.3.2.2.4 Approche Pareto

Contrairement aux deux précédentes approches, l'approche Pareto utilise la notion de dominance pour sélectionner des solutions faisant converger la population vers un ensemble de solutions qui approchent avec justesse le front de Pareto. Cette approche a été initiée par Goldberg en 1989 [Gol 89]. Cette approche assure un traitement équitable de chaque critère, car il n'y a pas de classement a priori de l'importance des critères. Ainsi, enfin de traitement, l'algorithme fournit un ensemble de solutions qui approchent le front de Pareto. Le choix de la solution finale revient donc à l'utilisateur, qui doit choisir parmi l'ensemble fourni la solution qui lui convient le mieux [Tal 99].

Ces méthodes se sont avérées être les plus efficaces donc, de nos jours, la majorité des algorithmes utilisent une approche Pareto pour traiter les problèmes multiobjectifs.

2.3.2.3 Métaheuristiques pour l'optimisation multiobjective

2.3.2.3.1 Algorithme de recuit simulé

La méthode de recuit simulé en optimisation multiobjective a d'abord été abordée sous l'angle agrégatif [Ser 92] [Fre et al 93]. Les deux méthodes les plus

populaires sont la méthode MOSA (Multiple Objective Simulated Annealing) proposée par Ulungu et al. [Ulu et al 99] et la méthode PASA (Pareto Archived Simulated Annealing) proposée dans [Eng 97]. MOSA utilise les caractéristiques du recuit simulé pour rechercher le plus efficacement possible les solutions non-dominées. PASA utilise une fonction d'agrégation des fonctions objectifs, couplée avec un système d'archivage des solutions non-dominées.

2.3.2.3.2 Algorithmes évolutionnaires

Les algorithmes évolutionnaires sont très largement utilisés pour résoudre des problèmes multiobjectifs. Parmi les méthodes utilisant l'approche Pareto, on distingue deux familles d'algorithmes : les non-élitistes et les élitistes. Parmi les méthodes non-élitistes, l'algorithme MOGA (Multiple Objective Genetic Algorithm) proposé par Fonseca et al. [Fon et al 95] est l'un des plus connus. Cet algorithme se base sur le classement des individus suivant le nombre d'individus dominés. NSGA (Non Dominated Sorting Genetic Algorithm) [Sri 95] propose de diviser la population en plusieurs groupes, fonctions du degré de domination de chaque individu. Enfin, l'algorithme qui fait référence dans le domaine est NSGA-II [Deb et al 02], qui est une évolution de NSGA, mais en version élitiste. Parmi les méthodes élitistes, on peut citer SPEA (Stength Pareto Evolutionary Algorithm) [Zit et al 02], où l'on profite du passage de génération pour mettre à jour les différentes sauvegardes. Les individus non-dominés sont sauvegardés et les individus dominés déjà présents sont éliminés.

2.3.2.3.3 Algorithme de colonies de fourmis

En optimisation mono-objectif, les algorithmes de colonies de fourmis sont très appréciés pour résoudre des problèmes combinatoires. En revanche, peu de travaux existent dans le cas multiobjectif.

Dans l'état de l'art réalisé par Blum en 2005 [Blu 05] on ne retrouve que trois références à des algorithmes multiobjectifs. Doerner et al. [38] [39] ont proposé un algorithme nommé P-ACO dédié à la résolution du problème d'allocation de portefeuilles. Cet algorithme présente de bons résultats, notamment face à NSGA et au recuit simulé, mais soufre de ne pas avoir été comparé à NSGA-II, la référence en la matière. L'algorithme OCF de Gagné [Cag et al 04] repose sur le même principe que P-ACO. A chaque itération, les fourmis changent de fonction

objectif à optimiser. A la fin de l'itération, la fourmi ayant la meilleure performance met à jour le tracé de phéromone.

2.3.2.3.4 Optimisation par Essaim Particulaire

L'Algorithme 2.7 présente le schéma global d'un algorithme d'OEP adapté pour l'optimisation multiobjective. Les étapes ajoutées par rapport à une OEP mono-objectif sont indiquées en italique. La résolution d'un problème multiobjectif revient à sélectionner les positions non-dominées trouvées par l'algorithme au cours de son exécution. De ce fait, à la fin de l'exécution, les solutions retenues se doivent d'être des solutions non-dominées vis-à-vis de toutes les positions atteintes par les particules au cours des itérations successives. Il apparaît donc nécessaire de tenir à jour une liste des solutions non-dominées trouvées au cours du traitement et de mettre celle-ci à jour à la fin de chaque itération. La solution la plus couramment utilisée pour résoudre ce problème est d'utiliser une archive extérieure. Une solution potentielle est alors acceptée dans l'archive si, et seulement si, elle est non-dominée vis-à-vis du contenu de l'archive. De plus, si elle domine des solutions contenues dans l'archive, celles-ci sont retirées de l'archive. Le problème d'une telle approche est qu'elle peut être très rapidement coûteuse en complexité. En effet, la taille de l'archive peut devenir rapidement importante. De ce fait, le temps de calcul pour mettre à jour les solutions non-dominées peut s'avérer excessif. Dans le pire des cas, toutes les positions de l'essaim sont ajoutées à l'archive, ce qui induit une complexité de O(kMN2) où k est le nombre d'objectifs, M est le nombre d'itérations et N est la taille de l'essaim. Pour pallier à cet inconvénient, il est commun de définir une taille maximale pour l'archive [Coe et al 02]. L'introduction de ce paramètre implique aussi la définition d'une règle d'inclusion de solutions dans l'archive, dans le cas où celle-ci est pleine.

L'OEP pour l'approche de transformation vers le mono-objectif

Dans cette approche, quelques algorithmes dédiés à la résolution d'un problème multiobjectif, en le transformant en un problème mono-objectif, ont été proposés dans la littérature ; la plupart sont recensés dans [Sie et al 06]. L'une des approches, utilisant l'agrégation linéaire d'objectifs, consiste à diviser l'essaim en sous essaims, de taille égale. Chaque sous essaim utilise un vecteur de

paramètres différents et évolue dans sa propre direction. Enfin, les solutions Pareto optimales sont choisies en utilisant la méthode du gradient.

L'OEP pour l'approche non Pareto

Plusieurs algorithmes ont été développés. Nombreux sont inspirés des AGs, à l'image de l'algorithme VEPSO dérivé de l'algorithme VEGA [Sch 85]. VEPSO utilise une approche dite multi essaims, où chaque essaim est évalué en utilisant une des fonctions objectives du problème. Les différents essaims communiquent entre eux à travers l'échange de leur meilleure position [Sie at al 06].

L'OEP pour l'approche Pareto

C'est dans cette approche d'optimisation multiobjective que le nombre de travaux publiés est le plus important. Ces approches utilisent les méthodes de sélection de "*leader"* basées sur la dominance au sens de Pareto. Le principe de base de toutes ces approches consiste à considérer des particules comme étant des leaders, si elles sont non dominées dans l'essaim. Plusieurs schémas de sélection des leaders sont possibles [Sie et al 06].

2.4 Conclusion

Dans ce chapitre nous avons présenté le problème d'optimisation, les algorithmes de résolution de ce problème à savoir les heuristiques et les métaheuristiques.
Les métaheuristiques sont des méthodes stochastiques qui visent à résoudre un large panel de problèmes, sans pour autant que l'utilisateur ait à modifier leur structure. Les métaheuristiques sont la plupart du temps issues de métaphores provenant de la nature, et notamment de la biologie. Les algorithmes de colonies de fourmis sont inspirés par le comportement d'insectes sociaux, mais ce ne sont pas les seuls algorithmes à être issus de l'étude du comportement animal (éthologie). En effet, l'optimisation par essaim particulaire (Particle Swarm Optimization), est issue d'une analogie avec les comportements collectifs de déplacements d'animaux, tels qu'on peut les observer dans les bancs de poissons ou les vols d'oiseaux. On a vu tout au long de ce chapitre que, suivant la nature du problème posé (continu/discret, monoobjectif/multiobjectif, etc.), les métaheuristiques sont facilement adaptables et/ou hybridables en vue d'obtenir les meilleures performances possibles.

Chapitre 3. Optimisation de la compression fractale : Travaux Similaires

« Le signe premier de la certitude scientifique, c'est qu'elle peut être revécue aussi bien dans son analyse que dans sa synthèse ».

Gaston Bachelard

3.1 Introduction

La compression fractale est une méthode de compression d'images encore peu utilisée aujourd'hui. Elle repose sur un principe simple : toute image comporte des similarités et des redondances. Cette méthode de compression consiste donc à détecter la récurrence des motifs à différentes échelles, et tend à éliminer la redondance d'informations dans l'image.

La compression fractale est une méthode de compression destructive puisque l'ensemble des données de départ ne se retrouve pas dans l'image finale. Nous expliquerons plus loin en détail pourquoi.

Plusieurs techniques de compression fractale d'image existent dans la littérature. Elles impliquent une codification d'image permettant une haute sécurité et un bruit réduit : la compression fractale de Huber [Jen 09], les ondelettes [Kuo 99] ou la classification floue [Han 08]. Les techniques les plus citées ont montré que le codage entraine une perte d'information considérable. Ainsi un temps de compression/ décompression très considérable et aussi une vulnérabilité aux statistiques de la cryptanalyse.

Ces dernières années, les chercheurs ont trouvé de nouvelles techniques basées sur une combinaison entre la compression fractale d'image et les métaheuristiques d'optimisation tels que les algorithmes Génétiques [Mit 98], les

colonies de fourmis [Yan 08] et l'optimisation par essaim de particule [Tse 08]. Dans le but de réduire le temps du codage et améliorer la qualité des images.

Dans ce chapitre nous effectuons un survol sur la plupart des méthodes proposées pour améliorer la compression fractale en utilisant les méta heuristiques d'optimisation.

3.2 Métaheuristiques d'optimisation pour la compression fractale d'images

Nous présentons, dans cette section, les différents travaux proposés dans le domaine de l'optimisation de la compression fractale par l'utilisation des métaheuristiques d'optimisation. Nous avons essayé de détailler quelques travaux que nous avons jugé les plus proches de notre méthode proposée, les autres sont brièvement discutés.

3.2.1 Compression fractal d'images et Algorithmes génétiques

3.2.1.1 Travaux de Lucia Vences et al. [Ven 97]

Dans ce travail la méthode proposée utilise les algorithmes génétiques pour trouver le système de fonctions itérées locales qui codifie une seule image. Le temps est réduit par 30% approximativement en comparaison avec la méthode de Barnsley si la qualité de l'image est désirée.

Si la qualité est moins acceptable, en utilisant un algorithme génétique, on peut varier le temps que le codage prendra en changeant des paramètres tel que la dimension de la population et nombre de générations permises.

L'algorithme a été étendu ensuite pour s'adapter avec les séquences d'images.

Les résultats de cette méthode sont un peu dépassés, puisque le matériel utilisé à l'époque est assez long par rapport à ce qui existe maintenant. Par exemple une simple image en niveau de gris de taille 296 * 240 peut prendre jusqu'à 33 minutes de temps de compression.

3.2.1.2 Travaux de Suman K. Mitra et al. [Mit 98]

Ici la méthode de compression fractale proposée utilise les algorithmes génétiques avec le modèle élitiste, cette technique diminue d'une façon remarquable l'espace de recherche pour trouver l'autosimilarité dans une image donnée.

Dans une image de Lena 256*256 8 bits/pixel, ils procèdent à la division de cette dernière en 4 sous-images de 128*128 dont chacune est encodée séparément (procédure fractale).

Les algorithmes génétiques sont implémentés comme étant une méthode de recherche. Pour chaque sous image, le nombre total des blocs de destination parents est de 256 et le nombre des blocs de domaine recherché (m) est de (128-16) *(128-16) = 112*112 et (128-8)*(128-8) = 120*120 pour les blocs destinations parents et fils respectivement. Donc la cardinalité M de l'espace de recherche pour ces deux cas est 112*112*8 et 120*120*8 respectivement. La longueur de la chaine a été pris pour être 17 (7 + 7 + 3) dans les deux cas.

Les résultats sont présentés dans le tableau suivant :

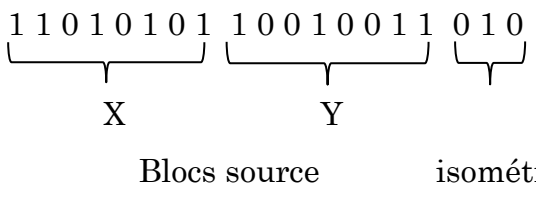
Type de compression	Algorithmes génétiques			Recherche exhaustive		
	CR	PNSR	Bloc de domaine	CR	PNSR	Bloc de domaine
1 niveau 8*8	21.7	26.16	4073160	21.7	26.20	85939200
2 niveaux 8*8 & 4*4	10.59	30.22	8102640	11.24	28.32	161869952

Tableau 3.1 : Travaux de Suman K. Mitra et al.

3.2.1.3 Travaux de Y.Chakrapani et al. [Cha 09]

Les Algorithmes Génétiques (GAs) sont utilisés pour la Compression Fractale d'Image de (FIC). Avec l'aide des algorithmes évolutionnaires pour réduire la complexité de la recherche entre bloc source et bloc de domaine. Pour améliorer le temps de calcul et aussi la qualité acceptable de l'image décodée, les algorithmes Génétiques sont proposés.

Dans ce travail une image de 256*256 de niveau de gris est considérée, les blocs sources sont de taille 4*4 et les blocs destination de taille 8*8. Dans l'algorithme génétique le chromosome individuel est codé comme suit :



14 bits pour l'emplacement de bloc de source (coordonnée horizontale et verticale), 3 bits pour les isométries. Pour chacun des blocs destination fractal coder inclut l'allocation de 17 bits.

Les résultats sont comme suit :

Figure3.1. (a) Image originale. (b) Image reconstruite avec AG.

Figure 3.2. (a) Image originale. (b) Image reconstruite avec AG.

Le tableau ci-dessus présente les résultats de cette méthode :

	Image	**FIC avec recherche exhaustive**	**FIC avec Algorithme Génétique**
Taux de compression	Barbara	1.2:1	1.2:1
	Lena	1.3:1	1.3:1
PSNR (dB)	Barbara	32.84	28.34
	Lena	32.69	26.22
Temps (sec)	Barbara	8400	2500
	Lena	8400	2370

Tableau 3.2 : Travaux de Y. Chakrapani et al.

3.2.1.4 Travaux de Wang Xing-yuan et al. [Wan 09]

Pour résoudre la haute complexité du plan du codage conventionnel pour la compression fractale, une corrélation spatiale de l'algorithme génétique hybride basée sur les caractéristiques de fractal et a divisé le système de la fonction répété (PIFS) est proposée. Il y a deux étapes pour l'algorithme :

(1) Faire usage de la corrélation spatiale dans les images pour les blocs sources et les blocs de domaine pour exploiter des optima locaux.

(2) Adopter la recuite simulée à algorithme génétique (SAGA) pour explorer les optima globaux.

Pour éviter la convergence prématurée, l'algorithme adopte l'opérateur de la mutation dyadique. Les résultats de l'expérience montrent que l'algorithme convergent rapidement. À la prémisse de bonne qualité de l'image reconstruite, la technique a amélioré le temps du codage et elle a offert un haut taux de compression.

Les opérations élémentaires de SAGA incluent sélection, croisement, mutation, et simulent la recuite. L'algorithme est comme suit :

(1) définissez formation du chromosome et fonction de l'aptitude, initialisez le comptoir de la génération de l'évolution.

(2) créez la population initiale.

(3) évaluez des valeurs de l'aptitude et épreuve critère qui s'arrête.

(4) sélection, croisement, mutation et recuite simulée opération.

(5) évaluez des valeurs de l'aptitude après algorithme de la recuite simulé.

(6) épreuve critère qui s'arrête. Si la valeur de l'aptitude satisfait l'obturation critère, alors ajoutez on a la génération de l'évolution comptoir et aller à l'étape (4) ; autrement enregistrez le courant individu le mieux similaire.

Image	Temps d'exécution	Qualité (PSNR) (dB)
Lena	16.51 s	27.15
Pepper	17.33 s	27.83

Tableau 3.3: Travaux de Wang Xing-yuan et al.

3.2.1.5 Travaux de Vishvas V. Kalunge et al. [Vis 12]

Ici les algorithmes génétiques sont utilisés dans le cadre de l'optimisation du temps de compression.

Puisque la compression fractale d'image pose un problème de temps de calcul, ce travail utilise comme solution les algorithmes génétiques pour trouver les parties similaires dans l'image. Cette tâche devient très facile et peut se terminer dans un temps réduit.

L'algorithme est le suivant :

Entrée: image carrée NxN
Initialiser les paramètres de la compression
    Taille du range block, fonction fitnesse, limite d'erreur, no. d'itérations.
Initialiser les paramètres du AG
    mutation, croisement ;
Diviser l'image en un ensemble de range blocks ;
Diviser l'image en un ensemble de domain blocks ;
Générer une population aléatoire de chromosomes à partir de region blocks ;
While Loop (No. d'itérations)
While Loop (until all Regions not coded)
    Sélectionner sequentiellement Region Blocks
    While Loop (until last generation reached)
        - calculer la fitnesse pour toutes les régions ;
        - chercher le block de domaine optimal ;

Ces travaux ont été réalisés sur des images de 128*128 avec des range block de taille 4*4 et domain block de taille 8*8.

Les résultats obtenus sont présentés dans le tableau suivant :

Image	Taille de l'image d'entrée	Taille de l'image de sortie	Temps de Compression sans AG	Temps de Compression avec AG
Lena	48 KB	5120 Bytes	184 sec	67 sec
Barbara	48 Kb	5120 Bytes	243 sec	66 sec

Tableau 3.4: Travaux de Vishvas V. Kalunge et al.

3.2.1.6 Travaux de Mahesh G. Huddar [Mah 13]

Ces travaux présentent une méthode qui utilise les algorithmes Génétiques pour améliorer le temps du calcul dans la compression fractale d'image avec une qualité d'image acceptable et un taux de compression élevé. Ces améliorations sont obtenues en chiffrant toutes les régions dans l'image avec des blocs de différentes dimensions. Les résultats obtenus par cette expérience sont détaillés dans les tableaux suivants :

Image	Temps d'exécution (min)	Qualité (PSNR) (dB)	Taux de compression
Lena	02 :55	30.05	9.73 :1
Boat	03 :22	25.86	7.9 :1
Pepper	02 :45	21.07	10.16 :1

Tableau 3.5 : Travaux de Mahesh G. Huddar

3.2.1.7 Travaux de Ming-Sheng Wu [Min 14]

Un algorithme génétique (GA) basé sur la transformation discrète des ondelettes (DWT) est proposé pour vaincre l'inconvénient du temps pour l'encodeur fractal. En premier, deux coefficients des ondelettes sont utilisés pour trouver le bloc source le plus proche du bloc de domaine pour chaque bloc source. L'égal semblable est fait seulement avec le bloc le plus en bonne santé pour sauver sept huitièmes calculs MSE redondants.

Seconde, enfoncer le DWT dans le GA, qu'un GA a basé sur DWT est construite plus loin à vitesse évolutionnaire rapide et maintient la bonne qualité rapportée. Les expériences montrent que, sous le même nombre de calculs MSE, le PSNR de la méthode GA proposée est réduit 0.29 à 0.47 dB en comparaison de méthode SGA. De plus, la méthode GA proposée est plus vite que la méthode de la recherche pleine 100 fois au temps du codage, pendant que l'amende de qualité de l'image rapportée est relativement acceptable.

Image	Méthode	N° de MSE calculées	PSNR (dB)
Lena	Full search	475,799,552	28.91
	FIC using DWT	59,474,944	28.55
Peppers	Full search	475,799,552	29.84
	FIC using DWT	59,474,944	29.74

Tableau 3.6 : Travaux de Ming-Sheng Wu

3.2.1.8 Travaux de Anamika Pandey et al. [Pan 15]

L'objectif principal de ce travail est d'analyser quantitativement des schémas compression d'image basés sur des paramètres qui incluent des croisements. La technique est implémentée avec la compression fractal d'image sur plusieurs images et même avec l'algorithme génétique adaptative proposé qui utilise des variantes de Croisement. Ces deux techniques ont été comparées sur quelques images célèbres dans plusieurs scénarios et aussi en variantes paramètres. Les résultats montrent que le mécanisme proposé a devancé.

L'algorithme génétique adaptative proposé dans ce travail est résumé dans les phases :

- Phase initiale
  - En entrée : Image.
  - Prétraitement.
- Décomposition d'image
  - Décomposition fractal d'image en utilisant la technique Quadtree.
- Application de l'algorithme génétique
  - Calcule de la fitnesse.
  - Sélection de la population
  - Reproduction (croisement/mutation)
  - Finalisation.
- Phase finale
  - Compression et décompression.
  - Images résultantes et calcule des métriques

Les résultats obtenus sont exposés dans le tableau suivant :

Image	Paramètre	FIC	Algorithme Proposé	
			1 point de croisement	2 points de croisement
	CR	13.51	22.07	24.06
	CT	1.78 s	1.39 s	1.11 s
	DcT	17.64 s	8.39 s	8.15 s
	SNR	16.53	13.96	14.97
	MSE	577.13	398.7	499.73
	PSNR	21. 34	22.99	24.43
	CR	14.85	34.75	36.75
	CT	1.70 s	0.79 s	0.79
	DcT	14.81 s	4.69 s	4.72 s
	SNR	18.73	16.6	15.22
	MSE	237.90	174.38	134.47
	PSNR	20.78	21.49	24.29

Tableau 3.7 : Travaux de Anamika Pandey et al.

3.2.2 Compression fractal d'images et les algorithmes bio-inspirés

3.2.2.1 Travaux de Cristian Martinez [Mar 06]

En 2006, Cristian Martinez a introduit un algorithme de l'Optimisation de la Colonie de la Fourmi pour compression de l'image. Il est basé sur le fait que les fourmis trouvent le chemin le plus court pour aller du nid à la source de la nourriture. Au sujet du problème de la compression de l'image du fractal, Phéromone sont déposés sur bloc de la gamme i et bloc de domaine j. La matrice de la phéromone est rectangulaire (pas symétrique) où les lignes indiquent des blocs de la gamme (blocs de l'image) et les blocs du domaine des colonnes (blocs transformer). Alors, chaque fourmi construit son chemin qui choisit un bloc de domaine j pour chaque bloc de la gamme i. La solution est basée en mettant à jour phéromone et information heuristique [Mah 13]. Ce la solution offre des

images avec qualité semblable à cela obtenu avec une méthode déterministe, dans approximativement 34% moins de temps.

3.2.2.2 Travaux de Chun-Chieh Tseng et al. [Tse 08]

Pour ces travaux l'optimisation par essaim particulaire (particle swarm optimization) (PSO) est utilisée avec l'information visuelle de la propriété de bord est proposée qui conserve speed up l'encodeur et conserve la qualité de l'image. Au lieu de la recherche pleine, une carte de la direction est construite d'après le bord-type de blocs de l'image qui dirigent les particules dans l'essaim aux régions qui consistent en candidats de plus haute ressemblance. Par conséquent, l'espace de recherche est réduit et les speed up peuvent être accompli. Aussi, depuis que la stratégie est exécutée d'après la propriété de bord, mieux l'effet visuel peut être conservé.

Image originale

full search PSNR = 28.91

PSO-kPSNR = 27.77

PSO-kIPSNR = 28.02

Figure 3.3 : Travaux de Chun-Chieh Tseng et al

3.2.2.3 Travaux de Dervis Karaboga [Kar 05]

En 2005, l'optimisation de la Colonie de l'Abeille Artificielle (ABC), une technique itération-basée, a été définie largement par Dervis Karaboga. C'est un outil de l'optimisation qui fournit à une population basée que la procédure de la recherche

où chaque individu a appelé des places de la nourriture est changée par les abeilles artificielles avec visée du temps pour trouver la source de la nourriture avec grand montant du nectar. L'ABC consiste en trois types d'abeilles (a) a employé abeille, (b) abeille du spectateur et (c) abeille du scout. Les abeilles du spectateur qui attendent dans la ruche reçoivent de l'information des abeilles employées concernant les sources du nectar avant qui ont été découvertes. Les abeilles du spectateur choisissent une source de la nourriture exploitable basée sur l'information reçue des abeilles employées. Allez en reconnaissance les abeilles recherchent au hasard une source de la nourriture dans l'environnement pour trouver nourriture.

3.2.2.4 Travaux de Y. Chakrapani et al. [Cha 10]

Dans ces travaux, la technique d'optimisation par essaim de la particule (PSO) est sollicitée pour la compression fractal d'image (FIC). Avec l'aide de cet Algorithme évolutionnaire on a pu réduire la complexité de la recherche de similarité entre bloc de destination et bloc de domaine.

Pour améliorer le temps d'exécution avec une qualité acceptable de l'image décodée, l'algorithme.

Dans une exécution typique du PSO, pour chaque bloc de domaine, un essaim initial de valeurs aléatoires qui correspondent aux coordonnées gauches supérieures du bloc de domaine et son isométrie est produit. Chaque valeur aléatoire correspond à l'emplacement du bloc de domaine et est utilisé pour évaluer le bloc de domaine et trouver le MSE. Le bloc de domaine avec le MSE minimale dans l'essaim est identifié et ses coordonnées sont notées comme meilleurs valeurs. Et ainsi de suite jusqu'à ce que l'image soit entièrement parcourus. Les images utilisées dans les tests sont celles de Barbara et Lena avec une taille de 256*256. Les résultats comparés à la compression fractal avec une recherche exhaustive sont donnés par le tableau suivant :

	Image	**FIC avec recherche exhaustive**	**FIC PSO**
Taux de compression	Barbara	0.98	1.89
	Lena	0.98	1.89

PSNR (dB)	Barbara	34.674	34.39
	Lena	35.26	32.98
Temps (sec)	Barbara	8400	6500
	Lena	8600	6620

Tableau 3.8 : Travaux de Y. Chakrapani et al.

3.2.2.5 Travaux de Wang Xing-Yuan et al. [Wan et al 15]

Dans ces travaux, un nouvel algorithme de la compression fractal d'image basé sur l'optimisation par essaim particulaires (PSO) et un partitionnement Quadtree hybride(QP) est proposé. Une méthode appelée stratégie PSO basée sur la classification par blocs de distance (PSO-RC) est présentée au lieu d'utiliser la méthode PSO dans l'ensemble du pool de plages. Cette nouvelle idée peut améliorer considérablement le taux de compression et accélérer le codeur. De plus, un système de QP hybride PSO-RC (PSO-RCQP) est adopté afin d'améliorer la qualité de l'image récupérée plus loin.

Tout d'abord, les blocs de gamme sont divisés en deux catégories en fonction de l'écart-type fonctionnalité. Deuxièmement, les blocs de plage sont codés en utilisant l'approche PSO ou en stockant directement les valeurs de pixel moyennes.

Troisièmement, les blocs de plage avec des erreurs de correspondance importantes lors de l'utilisation du schéma PSO utilisent la méthode QP proposée. Les résultats de la simulation montrent que l'algorithme proposé peut obtenir une bonne qualité et un taux de compression plus élevé aussi le raccourcissement le temps d'encodage.

3.3 Comparaison des différentes méthodes d'optimisation

Dans cette section nous avons essayé d'établir une comparaison entre les méthode d'optimisation par méta heuristiques de la compression fractal en

mettant l'accent sur des facteurs de comparaison tels que le PSNR, le temps de calcul ainsi que le taux de compression. A partir de ces critères nous évaluant ces méthodes.

Images de Test	**Méthodes**	**PSNR (dB)**	**Temps (s)**	**Taux (%)**
Lena	FF/FD [Wan et al 09]	29.02	17.18	/
	DWSR [Wan & Zha 14]	25.8212	56.4247	15.6355
	PSO-RCQP [Wan et al 15]	27.089	6.453	16.392
	F/MET [Wan & Zou 09]	32.77	30.30	/
	IFIC [Wan et al 09]	31.23	1.86	/
	HFIC [Wan et al 13]	27.23	13.7101	28.2941
Pepper	FF/FD	/	/	/
	DWSR	25.7231	56.4247	15.5510
	PSO-RCQP	24.983	6.750	26.292
	F/MET	/	/	/
	IFIC	/	/	/
	HFIC	26.04	15.8200	22.8886
Cameraman	FF/FD	/	/	/
	DWSR	/	/	/
	PSO-RCQP	26.686	4.281	18.212
	F/MET	/	/	/
	IFIC	/	/	/
	HFIC	24.95	8.4700	39.699

Tableau 3.9 : Tableau comparatif des différentes méthodes d'optimisation.

3.4 Conclusion

Ce chapitre nous a vraiment bien servi pour se positionner dans le contexte de l'étude. Donc, on a détaillé les différentes méthodes d'optimisation introduite dans la compression fractale, on a remarqué que toutes les optimisations de cette compression partent du point de vue d'étape de recherche des blocs dans la compression fractale, cette étape est très gourmande en temps de calcul, alors toutes les méthodes d'optimisation interviennent à cette étape.

Dans le chapitre suivant, et à notre tour, nous allons essayer d'appliquer l'algorithme du pacte des loups et pour la première fois sur la compression fractale, et voir ce que va donner.

Chapitre 4 : Approche proposée

Wolf Pack Algorithme pour la Compression fractale d'image

« La recherche scientifique ressemble beaucoup à celle de l'or : vous sortez et vous chassez, armé de cartes et d'instruments, mais en fin de compte, ni vos préparatifs, ni même votre intuition ne comptent. Vous avez besoin de votre chance ».

Michael Crichton

4.1Introduction

Actuellement, un volume considérable d'information est manié et échangé, et en particulier, les images ont attiré surtout une grande importance dans le domaine de la reconnaissance. Ainsi, il est essentiel de réduire le volume de données en utilisant des techniques de compression qui peuvent autoriser son chargement et son transfert toute en utilisant des ressources limitées. Les tendances courantes de la compression tendent vers l'usage des algorithmes de la théorie fractale qui paraissent comme un outil puissant pour améliorer la qualité de l'image et réduire l'usage des ressources.

Différemment aux messages textuels, les fichiers images sont impliqués dans plusieurs logiciels caractérisés par d'énormes données, corrélation de pixels et redondance. Dans de telle situation, les algorithmes de compression traditionnels paraissent non appropriés pour cette tâche, dû à leurs besoins au temps du traitement étendu. Par contraste, les algorithmes de la compression fractale d'image ont été développés récemment et proposent de bonnes performances de compression [Gal 03] [Tha et al 13] [Sel et al 09]. Ils sont basés sur le fait que les fractales peuvent décrire des scènes naturelles mieux que les formes géométriques traditionnelles. Plusieurs techniques de compression fractale

d'image sont proposées dans la littérature, elles impliquent le codage de l'image en haute sécurité et la réduction du bruit :*la compression fractale d'image du d'Huber* [Jen et al 09], *la compression fractale par les ondelettes* [Li et al 99] ou *la classification par le modèle floue* [Han 08]. Les techniques les plus citées ont montrées le codage engendre une perte considérable d'information et vulnérabilité vis-à-vis les statistiques de la cryptanalyse.

D'autre part, les algorithmes heuristiques sont aussi utilisés dans ce contexte tel que les algorithmes Génétiques [Mit 98], les colonies de fourmis [Li et al 08] et l'optimisation par essaims particulaires [Tse et al 08]. Ils sont capables de construire une partition basée région qui augmente le taux de la compression et conserve la qualité de l'image décomprimée.

Le but de notre méthode est d'essayer pour la première fois, une nouvelle métaheuristique, à savoir l'algorithme du pacte des loups (Wolf Pack Algorithm) et comment il peut être utilisé pour la compression d'images. La raison principale pour utiliser un tel algorithme est sa propriété de trouver la solution globale et sa capacité de produire des résultats acceptables de façon plus rapide et moins couteuse comparée à d'autres méthodes. L'algorithme utilise aussi un nombre réduit de paramètres sans le besoin d'une approximation initiale de paramètres inconnus.

4.2 Techniques bio-inspirées

Les Méta heuristiques représentent des techniques qui permettent la génération de solutions en maximisant le profit et minimisant l'usage de ressources ; cependant, l'optimalité de la solution ne peut pas être garantie si dans l'espace de l'exploration un croisement entre la solution locale et la globale se produit [Ola et al 08]. Les méta heuristiques bio-inspirées, une sous-classe des heuristiques, s'inspirent du comportement social d'animaux qui habitent dans des communautés telles que les essaims d'oiseaux, les colonies de fourmis ou les troupes de poissons, elles sont basées sur le principe des populations d'individus qui réagissent réciproquement et évoluent d'après une règle commune. De telles méthodes paraissent comme des modèles célèbres qui sont utilisées avec succès comme des outils puissants pour résoudre des problèmes combinatoires complexes [Gan et al 11] avec une consommation rationnelle de ressources.

Plusieurs travaux [Blu et al 08] [Fel 07] montrent que les algorithmes ont une possibilité prospère pour manier une grande gamme de données et peuvent être adaptés pour créer des solutions pour des problèmes d'optimisation différents.

4.3 Compression Fractale d'image

4.3.1 Vue générale

La compression fractale est une technique moderne pour la compression d'image avec perte [Lu 93]. Elle a été introduite originairement par Hutson [Lu 93] et Barnsley [Bar 95]. Elle explore les ressemblances entre différentes régions isolées de l'image [Pei et al 04] et sauvegarde seulement les paramètres des transformations au lieu des pixels de l'image. Ce principe autorise de construire une approximation de l'image originale en détectant les redondances des modèles sur plusieurs échelles et a tendance à éliminer la redondance d'information dans l'image originale afin que le résultat peut être assez acceptable.

Figure 4.1 : Fractals

Bien qu'il existe maintenant d'autres méthodes plus efficaces en matière de facteur de compression, nous décrirons ici la méthode **Jacquin** pour la compression des images, celle-ci étant la plus connue et la plus utilisée.

Comme nous l'avons dit précédemment, la compression fractale repose sur le principe qu'une image n'est qu'un ensemble de motifs identiques en nombre limité, auxquels on applique des transformations affines (rotations, symétries, agrandissements, réductions). La première étape consiste donc à réaliser deux partitionnements sur l'image, c'est-à-dire deux segmentations de l'image en blocs : un partitionnement des blocs « source » et un partitionnement des blocs « destination ».

Un point essentiel dans les partitionnements Source et Destination est que le pavage destination doit être plus petit que le pavage source. En effet, dans le cas contraire, nous serions amenés à faire un agrandissement (et non une réduction) lors de la transposition des figures sources vers les figures destinations. Une fractale possède un motif se répétant à l'infini, en se rétrécissant. Aussi, nous perdons cette propriété si le partitionnement destination est plus grand que le partitionnement source, l'image ne pourra alors pas converger.

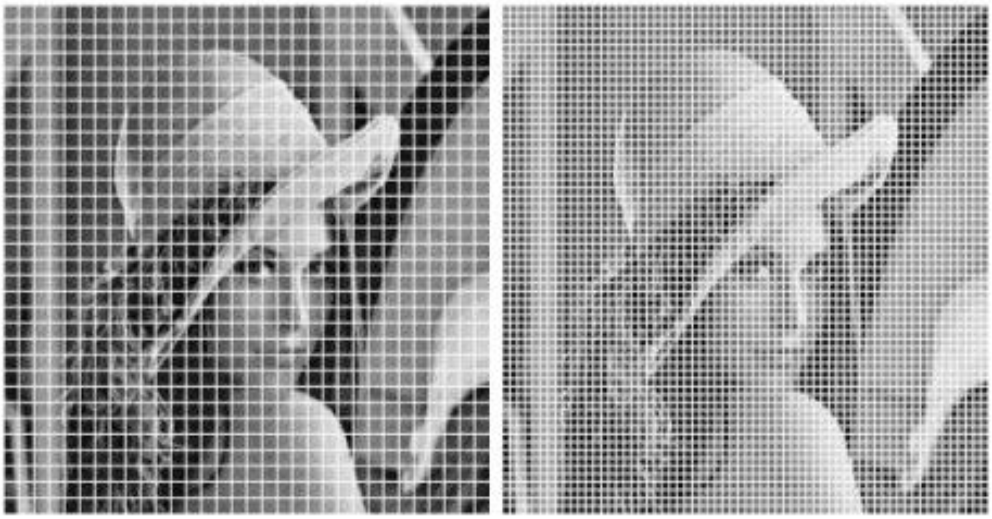


Figure 4.2 : partitionnement de l'image en blocs source et destination

L'étape suivante consiste à trouver, pour chaque bloc « source », le meilleur couple (bloc « source », bloc « destination ») minimisant l'erreur. L'erreur correspond à la différence entre les deux figures. On détermine ensuite les transformations nécessaires permettant de passer du bloc « source » au bloc « destination » : rotations, réductions, réflexions, variation du contraste et de la luminosité.

Ensuite, il suffit juste de coder le bloc « source » et la transformation permettant d'obtenir le bloc « destination ».

Au final, on peut considérer que l'image a été « fractalisée », puisqu'au prix de quelques pertes de données (l'erreur entre les blocs « source » et « destination »), on obtient une structure présentant des motifs similaires, à différentes échelles.

La compression fractale d'image est basée sur un système de fonction itérée *f*, un ensemble fini de contractions défini sur un espace métrique R^n par la relation :

$$f_i: R^n \rightarrow R^n \mid i < N \quad (2)$$

Une telle contraction peut prendre plusieurs formes d'après les limites techniques. Elle peut être appliquée à tous points dans l'image originale, ensuite les rapprocher dans l'image résultat. Cette idée, appelée transformation de l'affine, permet à chaque sous-bloc de l'image originale à se transformer selon une relation mathématique, comme suit :

$$W(x)= T(x)+b \qquad (3)$$

Où T est une transformation linéaire sur R^n et b$\in R^n$ est un vecteur.

Dans la pratique, le concept fondamental du codage fractal est illustré par le pseudo-algorithme suivant :

- Créer un partitionnement de figures Sources
- Créer un partitionnement de figures Destinations
- **Pour** toutes les figures Destinations **Faire**
  - **Pour** toutes les figures Sources **Faire**
    - **Pour** toutes les transformations définies **Faire**
      - Appliquer la transformation à la figure Source
      - Ajuster la moyenne des couleurs des pixels
      - Appliquer la réduction de la figure Source vers la figure Destination
      - Calculer l'erreur entre le résultat et la figure de destination
      - **Si** l'erreur est minimale pour la figure de destination **Alors**

        Sauver les modifications effectuées
      - **Fin Si**
    - **Fin Pour**
    - Ecrire dans le fichier de sortie les valeurs sauvées
  - **Fin Pour**
- **Fin Pour**

Ce processus peut se réaliser à l'aide de la formule de contraction suivante :

$$B_i = v(D_i) \qquad (4)$$

Où v () est la fonction de la contraction qui consiste à modifier D_i.

Pour calculer l'erreur entre Bi et Ri, on peut utiliser par exemple la distance de Hausdorff définit par la formule :

$$H(B_i, R_i)= max(d(B_i,R_i), d(R_i,B_i)) \qquad (5)$$

Où d (B, R) = $max_{b \in A} min_{r \in B}$ ||b-r || ou par la distance Euclidienne illustrée par l'équation:

$$d^2 (R,B)=\Sigma^n(r_i,b_i)^2 \qquad (6)$$

Où n est le nombre de pixels dans les blocs R_i et B_i. Le paramètre d devrait être minime pour les blocs semblables.

La décompression fractale consiste en la reconstruction des blocs R_i à partir des blocs Bi les plus semblable en répétant la transformation de la contraction utilisée dans la compression.

Ces dernières années, plusieurs études se concentrent sur le développement des algorithmes du déchiffrement rapide [Gei et al 94] [Moo et al 99] [Moo et al 97] avec un but de conserver la qualité d'image. Leur principe consiste à choisir une image arbitraire comme image originale et exécuter une transformation de l'affine basé sur les codes fractals obtenus d'elle-même. Cette action est répétée récursivement jusqu'à ce que l'image produite trouve la satisfaction de l'utilisateur.

4.3.2 Différentes approches pour la compression fractale

Actuellement, la compression fractale d'image devient une des techniques les plus prometteuses pour chiffrer des images dues à son taux de compression élevé et sa dégradation minime de l'image. L'histoire de cette méthode de compression remonte aux années 90 où la première technique a été proposée par ***Jacquin*** [Jac 92] [Jac 93]; son principe est bien détaillé dans la section 4.3.1. Depuis, les chercheurs ont introduit de nouvelles idées pour réduire le temps du codage énorme ; le travail de Thomas et Deravi [Tho et al 95] combine des blocs de destination et les rend plus adaptatifs avec contenu de l'image en utilisant la méthode région-croissante. Une idée semblable a été proposée par Cardinal [Car 01] ; elle est basée sur une partition géométrique de l'espace de bloc de l'image en niveau de gris. Les comparaisons expérimentales avec les méthodes publiées précédemment montrent une amélioration considérable dans la vitesse sans perdre de la qualité. He al. [He et al 06] ayez l'idée d'utiliser une norme de bloc normalisé pour éviter la recherche excessive dans la comparaison des blocs. Chong et Pi [Ton et al 01] ont présenté une nouvelle approche de recherche adaptative pour réduire la complexité computationnelle du codage fractal pour exclure un grand nombre de blocs de domaine non qualifié pour accélérer la

compression fractale d'image. Cependant, cette technique est caractérisée par un processus asymétrique, il passe principalement tant de temps dans le processus du codage que dans le processus de la décompression qui consiste dans la recherche du meilleur bloc sur les grandes images.

Plusieurs autres recherches ont introduit de nouveaux concepts pour améliorer la qualité de la recherche tel le codage par la transformée de Fourier [Har et al 00], special image features [Zha et al 07], DCT Inner Product [Tru et al 00].

La plupart des approches sont basées sur un taux d'erreur assorti pour restreindre l'espace de recherche. Récemment, Lin et Wu [Lin et al 11] ont proposé une stratégie de la recherche basée sur les propriétés de limites des blocs de l'image qui présente une performance acceptable.

En outre, les nombreux papiers de la recherche ont été publiés pendant la dernière décennie ; ils ont rehaussé la qualité d'image sans augmentation des ressources du processus de codage.

4.4 L'algorithme du pacte des loups « Wolf Pack Algorithm »

Le Wolf Pack Algorithm (WPA) [Wu et al 14] est l'un des algorithmes de cette famille (bio-inspirée) qui peuvent être employés pour se rapprocher de solutions pour plusieurs problèmes d'optimisation. WPA est une métaheuristique basée population caractérisée par le comportement de la chasse sociale des loups. Il consiste essentiellement à mettre les loups chasser, trouvez la trace de proie et capturez-la sous l'ordre d'un loup chef.

Le pacte des loups comprend un loup chef qui est le plus fort et le plus intelligent. Il prend à sa charge le contrôle de la meute. Ses décisions sont toujours basées sur l'environnement avoisinant : proies, loups de meute et d'autres prédateurs. Le pacte est donc divisé en deux classes de loups : chasseurs et furieux. La première classe bouge indépendamment dans l'environnement et règle sa direction d'après l'odeur des proies. Quand une proie est localisée, les loups de cette classe crient et rapportent l'information qui utilise le son au chef-loup qui estime la distance spatiale, invoque vite les furieux-loups et bougent vers le son. La proie est capturée alors ; ensuite elle est distribuée selon l'état de chaque loup : du plus fort au plus faible. Ainsi, les loups faibles peuvent mourir à cause

de manque de nourriture. C'est comment le pacte reste à tout moment dynamique et robuste.

Le WPA est réalisé comme suit :

- Dans un espace de la recherche $\mathbb{R}^n$, chaque loup i représente une solution de base du problème et a une place x_i. Initialement, les loups sont initialisés au hasard dans l'espace.
- À chaque instant t, le loup i déplace de la position x^t_i à la place x^{t+1}_i. Le choix de la nouvelle position est mis à jour selon l'équation suivante :

$$x_i^{t+1} = x_i^t + \lambda \left| x_g^t - x_i^t \right| \quad (1)$$

  Où λ est un vecteur aléatoire distribué dans l'intervalle [-1,1] et x_g est la position du chef-loup.
- Après un nombre fixe d'itérations qui correspondent à une phase détalant, le loup de la meilleure solution devient un chef-loup ; un nombre donné de loups faibles (mauvaises solutions) sera effacé et sera remplacé par une nouvelle génération de loups aléatoires.

L'algorithme suivant explique bien la fonction objective des mouvements des loups :

Algorithm 1. WPA

Fonction Objective *f(x), x=(x1,x2,..xd)T*

Initialiser la population des loups, $x_i (i=1,2,..,W)$

Définir et initialiser les paramètres :

r = radius of the visual range

s = l'étape

a = facteur de vélocité du loup

Pa = vecteur aléatoire [0..1].

Tanque non (*critère d'arrêt && t* < générations)

pour i=1:W // ***pour chaque loup***

Proie initiative ();

Génèrer_nouvelle_position ();

// tester si la seconde position suggérée par le numéro aléatoire est bien nouvelle. Sinon régénérer une nouvelle position aléatoire.

***Si*(dist (x_i^t, x_i^{t+1}) <*r* && x_i^{t+1} est meilleur $f(x_i^t) < f(x_i^{t+1})$)**

x_i^t déplace suivant x_i^{t+1} // x_i^{t+1} est meilleur que x_i^t

sinon

x_i = proie passive();

Fin si

Génèrer_nouvelle_position ();

Si(rand()>pa)

$x_i^t = x_i^t$ + rand() + v; // échapper pour la nouvelle position.

Fin si

Fin Pour

Fin Tanque

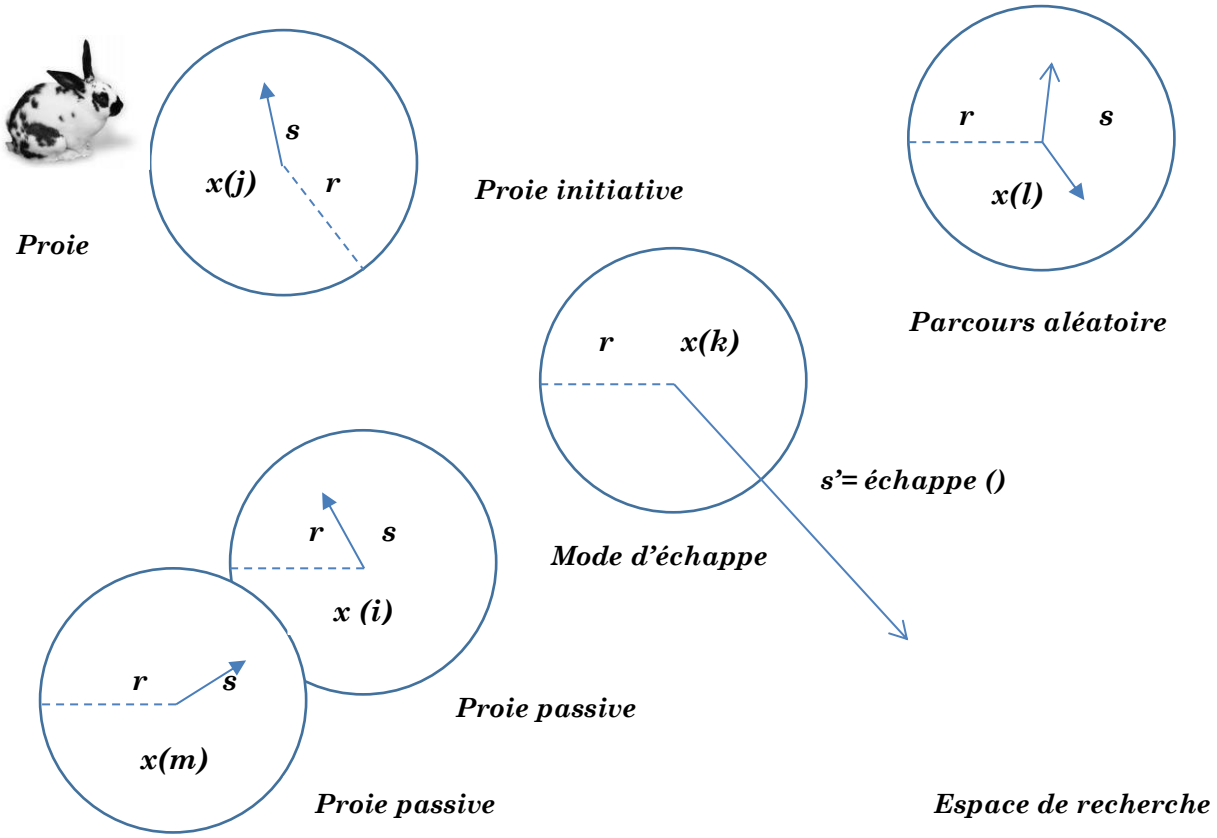


Figure 4.3 : illustration du comportement des loups de la meute

4.5 Compression fractale avec le Wolf Pack Algorithm

Dans le chapitre précédent on a présenté une large gamme d'algorithmes introduisant les métaheuristiques d'optimisation dans la compression fractale d'image, on a vu aussi les points forts et faibles presque de chaque méthode citée. Dans cette section, on va essayer d'impliquer une technique bio-inspirée, à savoir le Wolf Pack Algorithm *(WPA)* ; et pour la première fois, dans la compression fractale.

En assumant une image ordinaire de m x n pixels comme un espace de recherche, dénoté par un tableau P où chaque pixel est représenté par une cellule et a référencé par un octet (niveau de gris). L'image comprimée C de $m/2$ x $n/2$ pixels est obtenue d'après les étapes suivantes :

- Divisez le domaine de l'image en petits blocs sans aucun chevauchement r_i de taille s×s (avec s <<m). Pour simplifier, les blocs sont de dimension carrées de b x b et est représentée par $R_N = \{r_1, r_2, .., r_N\}$, un bloc source.
- Pour chaque r_i, les loups chercheurs explorent l'espace pour trouver un d_i bloc de destination 2b×2b lequel ayant une ressemblance avec ri basée sur ses paramètres.
- Pour chaque bloc d_i sera affectés une valeur la fitnesse $f(d_i)$ d'après l'équation (6). Le bloc d_i est considéré comme une proie.
- Si l'espace entier est parcouru par les loups chercheurs et, pour chaque r_i, le bloc d_i avec la meilleure fitnesse est sélectionnée. Un mappage est dressé d'après les équations (4) et (5).
- Le processus sera arrêté après un nombre fixe d'itérations ou si aucune amélioration n'est faite sur la solution du loup chef.

L'algorithme WPA pour la compression fractale d'image est illustré comme suit:

Algorithm 2. WPA pour la compression fractale

Input: Problem dimension (N >2, g), objective function *f*.

Output: {B}

Initialization :

Generate $\boldsymbol{r_i}$, (i=1..N)

For each $\boldsymbol{r_i}$, f($\boldsymbol{r_i}$) ← 0, (i=1.. N)

While not (*stopping criteria)*

it←0

While **not** (***Iter***$_{scoot}$<*It)*

Pick random numbers: $\lambda \in [-1,1]$

For each $\underline{r_i}$ do

$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \lambda |x_g - x_i|$

If $(f(b_{xi}) < f(g))$ $g \leftarrow i$ Endif

End for

$\boldsymbol{r}_i \leftarrow v(\boldsymbol{d}_i)$

Update ***It***

End While

End While

4.6 Conclusion

Dans ce chapitre on a bien détaillé la méthode de compression par des fractales. Une méthode irréversible de compression d'images basée principalement sur le principe système de fonctions itérées *f*, ce système prouve qu'on peut recouvrir toute forme de l'espace par des copies d'elles-mêmes.

Dans la pratique, le concept fondamental de la compression fractale d'images est de trouver la meilleure ressemblance des blocs source pour chaque bloc destination. Donc on est appelé à découper l'image en bloc (deux partitionnements), puis on cherche les motifs qui se répètent dans l'image. Cette deuxième étape est très lourde en calcul car si l'image est de résolution 320x240, il y a donc 1200 blocs (8x8) (par exemple) et chaque bloc doit être vérifié avec l'ensemble de tous les autres blocs.

Pour pallier le problème de lourdeur de calcul, on a introduit la notion de métaheuristiques d'optimisation, plus précisément, la technique du pacte des loups où les loups chercheurs partagent la tâche de recherche pour arriver à réduire le temps considérable de calcul, qui présente l'inconvénient majeur de cette méthode de compression.

Notre méthode va opérer dans un premier temps sur la division de l'image en petits blocs (blocs source et blocs destination), Ensuite les loups chercheurs explorent l'espace pour trouver une ressemblance entre les blocs source et destination jusqu'à ce que l'espace de recherche entier soit parcouru.

Pour chaque bloc *source*, un bloc *destination* avec la meilleure fitnesse est sélectionné. Un mappage est dressé pour la reconstruction de l'image.

Dans le chapitre suivant on présentera des détails sur l'implémentation de notre algorithme, les résultats obtenus, ainsi que des comparaisons avec d'autres méthodes citées dans le chapitre précédent.

Chapitre 5 : Résultats et Discussion

« Celui qui dit que deux et deux font quatre, a-t-il une connaissance de plus que celui qui se contenterait de dire que deux et deux font deux et deux ? ».

Jean le Rond d'Alembert

5.1 Introduction

Dans le chapitre 4, nous avons détaillé notre approche proposée, qui consiste à introduire une technique bio-inspirée nommée le Wolf Pack Algorithme dans la compression fractale d'image.

Dans ce chapitre, nous allons nous intéresser aux résultats expérimentaux obtenus tout en passant par la définition de quelques mesures de performance utilisées :

(i) ***Taux de compression*** : le taux de compression d'image (compression ratio) est une mesure de réduction du coefficient détaillé des données. CR est défini comme le nombre de bits de l'image originale (B_{org}) par le nombre de bits de l'image comprimée (B_{comp}). Ce taux peut être exprimé comme suit :

$$\boldsymbol{CR = \frac{B_{org}}{B_{comp}}}$$

(ii) ***Temps de compression et décompression :*** la compression peut être avec ou sans perte de données, la compression consiste à réduire les bits de données, le temps pris pour une telle réduction est le temps de compression, inversement le temps occupé pour reconstruire et restaurer l'image est le temps de décompression.

(iii) Taux de Signal/bruit : (signal to noise ratio) Le taux du signal-à-bruit (abrégé souvent SNR ou S/N) est une mesure utilisée en ingénierie pour mesurer combien un signal a été corrompu par un bruit. Il est défini comme le taux de force du signal propulsé à la force du bruit qui abîme le signal. Signal to Noise ratio est défini comme le taux de force entre un signal (information significative) et le bruit de l'origine (signal non désiré):

$$\boldsymbol{SNR = \frac{P_{signal}}{P_{noise}}}$$

(iv) L'erreur quadratique moyenne : (Mean Square Error « MSE ») c'est l'erreur carrée cumulative entre l'image originale est l'image comprimée. Une petite valeur de MSE est équivalente à une erreur minime.

$$\boldsymbol{MSE = \frac{1}{m\,n} \sum_{i=0}^{m-1} \sum_{i=0}^{n-1} [I(i,j) - K(i,j)]^{2}}$$

5.2 Processus de compression

Le processus du codage traverse dès le commencement des pas multiples avec la décomposition des loups et termine avec le codage Huffman.

Décomposition avec loups (Wolf decomposition)

Function S = wolfdcmp (Ima, Loudness, Number_wolfs, Frequency)

Cette fonction décompose l'image en de petits blocs selon les critères de ressemblance, alors elle affecte les résultats dans un S épars, Cet épar contient la place des blocs et leurs dimensions.

Le premier pas de la décomposition avec les loups est d'initialiser ces derniers et les étendre sur l'image.

Création des loups

wolfs = create_wolfs(SM, SN, Number_wolfs);

Cette fonction crée les loups et leurs positions aléatoires.

Les variables d'entrée dans cette fonction sont la largeur, la hauteur et nombre des loups.

```
function creation = create_wolfs(m,n,number_wolfs)
resultat = cell(m,n);
creation = cell(m*n/number_wolfs,number_wolfs);
for i = 1:m
    for j = 1:n
        resultat{i,j} = [i,j];
    end
end
Cell =resultat(randperm(numel(resultat))) ;
for i = 1:m*n/number_wolfs
    for j = 1:number_wolfs
    creation{i,j} = Cell{i+((j-1)*(m*n/number_wolfs))};
    end
end
```

Au début, une cellule vide est créée, dont la dimension est définie par le nombre de lignes = largeur * hauteur / nombre de Loups tandis que le nombre de colonnes=nombre de Loups.

Après que la cellule soit remplie de places commandées des pixels de l'image entière, le processus des loups scoutes commence. La cellule est transformée dans un vecteur qui est parcouru en utilisant la fonction du **randperm** puis retourné dans la cellule.

Début de décomposition

```
for Iteration = 1 : (SM*SN)/Number_wolfs
    for Number = 1:Number_wolfs
```

Dans cette étape, nous utilisons deux boucle imbriquées. La première est la boucle d'itération et la deuxième est une boucle pour exploiter les loups.

Après, les places de loups (m, n) seront restaurées de la cellule qui a été créée précédemment.

```
X = wolfs {Iteration,Number};
m = X (1);
n = X (2);
```

Après, une fonction **Search** est utilisée pour tester si cette zone est déjà explorée ou pas pour éviter les parcours répétés.

if (Search(m,n,Coordinates)==0)

Les variables d'entrée de cette fonction sont : **m**, **n** et **Coordinates**, qui contient les positions des pixels déjà visités.

```
function rechcherche = Search(m,n,Coordinates)
recherche = 0;
[L,C] = size(Coordinates);
for i = 1:L
      if (m == Coordinates (i,1) && n == Coordinates ( i,2))
            rech = 1;
            return
       end
end
```

Après, si cette position n'existe pas dans **Coordinates**, le loup commence à tester la similarité en utilisant la fonction **compare_gray**.

compare_gray(m,n,ima,Loudness,Frequency)

Les variables d'entrée de cette fonction sont **m**, **n**, **ima(l'image)**, **intensité** et **fréquence**.

```
function compare_den = compare_gray(m,n,ima,Loudness,Frequency)
compare_den = 1;
app = double(ima(m,n));
for i = 2:Loudness
      V1 =ima(m:(m+i)-1,(n+i)-1);
      V1 = V1';
      V = unique([V1,ima((n+i)-1,m:(m+i)-1)]);
      for j = 1 : length(V)
            x = double(V(j));
            comp = abs(app-x);
            if (comp <=Frequency)
                  compare_den = i;
```

```
            else
                if (compare_den == i)
                        compare_den = i-1;
                end
                return
            end
        end
end.
```

La première étape est de boucler sur l'intensité, en commençant avec la valeur minimum d'intensité 2, puis nous créons un vecteur V1 qui contient les pixels que nous voulons comparer avec ceux d'entrée. Après cela, nous testerons si chaque partie de 2x2 a une ressemblance avec le pixel d'entrée.

Si c'est vrai, la variable du compare_gray sera égale à 2.

On teste ensuite sur 3x3 dimension et ainsi de suite jusqu'à ce que la valeur du compare_gray soit égale à l'intensité, ou bien la ressemblance n'est pas trouvée et le compare_gray sera égale à la dernière dimension dans laquelle une ressemblance est trouvée.

Si c'est le cas contraire, la fonction arrêtera et la valeur du compare_gray sera 1.

```
if (comp~=1)
    for im = m:m+(comp-1)
        for jn = n:n+(comp-1)
            Coordinates =[ Coordinates; im,jn];
            if (im~=m || jn~=n)
                blksz = [blksz;-2];
            else
                blksz = [blksz;comp];
            end
        end
    end
else
    Coordinates =[ Coordinates; m,n];
    blksz = [blksz;1];
end
```

Maintenant, si le résultat de la variable (**comp**) de la fonction précédente est égale à 1 alors seulement la place du pixel courant sera sauvegardée dans **Coordinates** et la dimension du bloc sera aussi 1, autrement nous entreposons la place des pixels qui créent un bloc qui commence de la place courante avec **comp*comp** de la dimension et entrepose la dimension du bloc qui est égale à **comp**.

Après cette étape, les loups auront parcouru l'image entière et toutes les places et les dimensions des blocs auront été sauvegardées.

Choisir la meilleure solution

La matrice des coordonnées contient des positions redondantes. Alors, pour éliminer la plus mauvaise solution, on utilise la fonction **rech_repetition**.

Les variables de cette fonction sont **Coordinates** qui contient les positions redondantes et **blksz** qui contient la taille des blocs.

```
function rechrep = rech_repetition(Coordinates,blksz)
[L,C] = size(Coordinates);
for i = 1:L
  for j = 1:L
    if (i~=j)
      if (Coordinates(i,1) == Coordinates(j,1) && Coordinates(i,2) == Coordinates(j,2))
        if (blksz(i) < blksz(j))
          Coordinates(i,1) = 0;
          Coordinates(i,2) = 0;
        else
          Coordinates(j,1) = 0;
          Coordinates(j,2) = 0;
        end
      end
    end
  end
end
rechrep = Coordinates;
```

Comme nous pouvons le voir, cette fonction commence par un test. Si une position redondante est trouvée, on déterminera la meilleure solution en trouvant laquelle d'eux a la plus grande dimension du bloc. La position avec la plus mauvaise solution deviendra (0, 0) (case de départ).

Calculer la valeur moyenne

Cette étape consiste en le calcul de la valeur moyenne des blocs

```
[i,j,blksz] = find(S);
blkcount=length(i);
avg=zeros(blkcount,1);
for k=1:blkcount
   avg(k)=mean2(Ima(i(k):i(k)+blksz(k)-1,j(k):j(k)+blksz(k)-1));
end
avg=uint8(avg);
```

On extrait les positions et les tailles des blocs de l'espace crée précédemment, ensuite on boucle sur l'image et on calcul la valeur moyenne de chaque bloc domaine crée.

Codage Huffman

[sp,comp,symbols,data,dict] = Huffencod(i,j,blksz,avg);

Le codage Huffman est une fonction prédéfinie qui crée un arbre où le fond contient les valeurs les plus redondantes. Elles deviennent moins redondantes plus nous approchons le sommet de l'arbre.

5.3 Processus de décompression

Décodage Huffman

A travers le codage Huffman, nous retrouvons les données et leur dictionnaire pour retrouver les positions, la taille des blocs et la valeur moyenne.

[inew,jnew,blksznew,avgnew] = Huffdecod(comp,data,dict);

Reconstruction de l'image

En utilisant les positions (I, J), la taille des blocs et la valeur moyenne, on boucle sur la reconstruction de l'image.

```
for k=1:blkcount
    outim(i(k):i(k)+blksz(k)-1,j(k):j(k)+blksz(k)-1)=avg(k);
end
```

5.4 Expérimentation et résultats préliminaires

Dans cette section et pour montrer la performance de l'algorithme proposé, une expérimentation de base est menée pour tester son efficacité sur la compression d'images standards : Pepper, Building et Boat (figure 5.1) avec une basse résolution de 256 niveaux gris.

Les premières épreuves ont été traitées sur un laptop Intel Core 2.2 Ghz avec une mémoire de 4 Go.

Figure 5.1 : images standards

Le paramétrage de l'algorithme est comme suit:

- Le nombre de l'itération $it \in [10,100]$.
- La population des loups scouts est choisi à l'aide du tableau suivant :

Image de test	Nombre de loups	Temps de compression (s)	Temps de décompression (s)	Taux de compression	EQM	PSNR (dB)
Lena 256*256	8	668.810	0.850	1.596	9.282	33.305
	16	688.058	5.505	1.594	9.093	33.283
	32	646.653	1.013	1.596	9.424	32.924

Tableau 5.1 : Nombre de loups chercheurs

D'après le test illustré dans le tableau 5.1, le nombre de loups chercheurs qui offre un meilleur compromis temps/qualité est 8.

- Le blocs source sont dans l'intervalle [20,50].
- Le vecteur aléatoire dans l'intervalle de [0.5,1].

Les résultats numériques ont été faits en moyenne de plus de 5 exécutions pour chaque test.

Les résultats préliminaires sont illustrés dans le tableau 5.2 qui démontre une performance acceptable de l'algorithme proposé en comparaison avec la recherche exhaustive.

Méthode	Temps de Compression (s)			Taux (%)		
	Peppers	Building	Boat	Peppers	Building	Boat
Recherche Exhaustive	3,11	2,28	3,28	9,33	9,81	8,25
Recherche WPA	2.04	1.98	2,83	10,01	9,91	9,83

Tableau 5.2 : WPA vs Recherche Exhaustive.

La figure 5.2 présente les images reconstruites qui paraissent claires et presque proches des originales. Aussi elles présentent un taux de compression très intéressant, avec temps réduit par rapport à plusieurs algorithmes de compression fractale classiques.

Figure 5.2 : images reconstruites.

5.5 Résultats et analyse

Dans cette section, nous présentons des résultats plus détaillés de l'approche proposée sur des images issues d'une base de test, où on a pris d'autres exemples

d'images standards (Lena, Barbara, Cameraman). A la fin de cette partie une comparaison avec quelques méthodes de compression est dressée. Avant de faire cette comparaison, nous présentons notre approche avec des résolutions différentes d'image et testons les mesures de performance.

Ces tests sont maintenant réalisés sur un PC desktop dont les caractéristiques sont les suivants : AMD FX™-6100 Six-Core Processor 3.30 GHz, 32 GB RAM, Windows 10, 64 bit.

5.5.1 Résolution

La résolution présente un facteur qui ne doit pas être négligé lorsqu'on veut tester l'efficacité d'une approche. Pour notre cas, nous allons prendre en considération les trois images de test (Lena, Barbara et Cameraman) avec différentes résolutions (16*16, 32*32, 64*64, 128*128, 256*256) pour que nous puissions voir l'impact de ce facteur sur les grandeurs de qualité (Taux de compression, Temps de compression, EQM, PSNR).

(a) (b) (c)

Figure 5.3 : images de test (avant compression)

(a): Cameraman, (b): Lena, (c): Barbara.

(a) (b) (c)

Figure 5.4 : images de test décompressées (après compression WPA)

(a): Cameraman, (b): Lena, (c): Barbara.

A partir des images de test avec une résolution de 256*256 reconstruite, on voit que la qualité des images est acceptable à un très bon degré. Et pour être plus exacte, on doit se référer à des mesures calculables.

Le tableau ci-dessous présente les résultats obtenus par l'application de notre approche sur les images ci-dessus avec différentes résolutions :

Image de test	**Résolution**	**Temps de compression (s)**	**Temps de décompression (s)**	**Taux de compression**	**EQM**	**PSNR (dB)**
Cameraman	16*16	0.184	0.818	1.098	2.617	37.999
	32*32	0.446	0.802	1.174	4.079	36.173
	64*64	2.451	0.816	1.420	6.853	31.509
	128*128	41.284	0.810	1.653	8.931	30.096
	256*256	774.438	0.897	1.741	9.152	31.605
Lena	16*16	0.185	0.832	1.095	7.730	33.616
	32*32	0.507	0.801	1.125	5.861	34.451
	64*64	2.290	0.791	1.287	8.401	33.550
	128*128	34.257	0.821	1.480	9.547	32.641
	256*256	668.810	0.850	1.596	9.282	33.305
Barbara	16*16	0.225	0.812	1.019	1.352	38.070
	32*32	0.607	0.763	1.055	4.064	36.067
	64*64	1.868	0.781	1.152	6.259	35.060
	128*128	22.713	0.793	1.310	11.093	32.348
	256*256	509.406	0.861	1.447	11.115	33.288

Tableau 5.3 : variation de résolution des images.

Les résultats obtenus (détaillés dans le tableau 5.3) montrent bien en premier lieu la corrélation entre la résolution des images et le temps de compression, ce qui prouve que notre approche est sensible aussi au changement de résolution. On voit bien aussi que la qualité des images est inversement liée à la résolution (dès que la résolution augmente, la qualité des images se dégradent). Cela est bien exprimé par les grandeurs de PSNR et EQM.

Le taux de compression reste relatif à la résolution où on remarque que notre approche intervient dans cette mesure. Les loups se manifestent lorsque le

nombre de blocs est bien important (résolution plus grande) en offrant un taux de compression plus important que celui présent dans des résolutions plus basses.

Le temps de décompression reste optimisé est similaire pour presque toutes les méthodes. Cela est dû au processus de décompression qui ne comporte pas de complexité par rapport au processus de compression (p 5.3).

Les figures suivante montre bien la corrélation entre les différentes grandeurs et la résolution des images

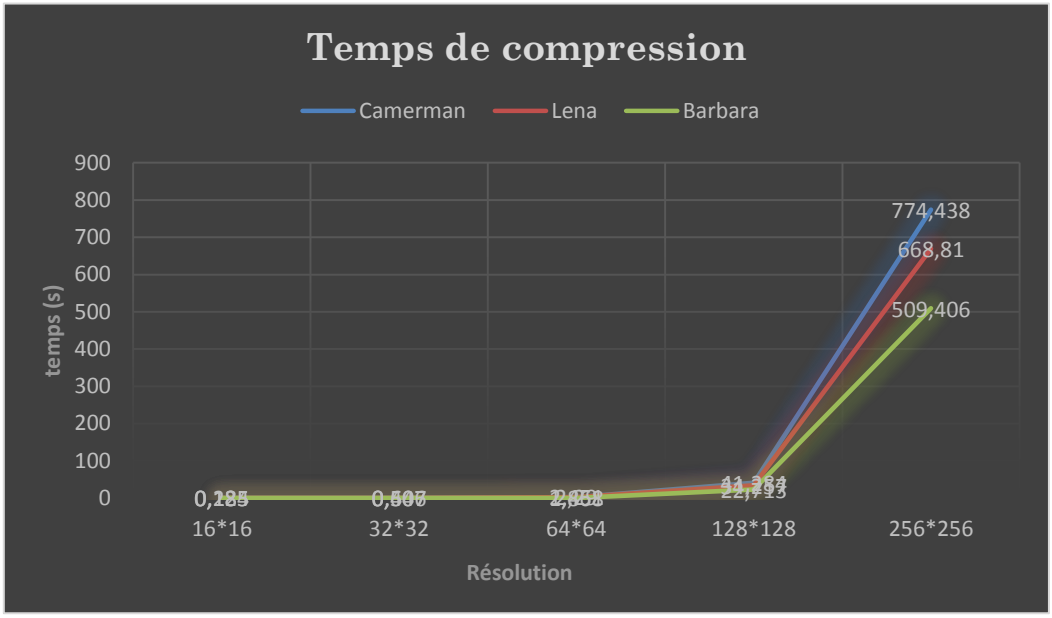


Figure 5.5: Temps de compression vs variation de résolution

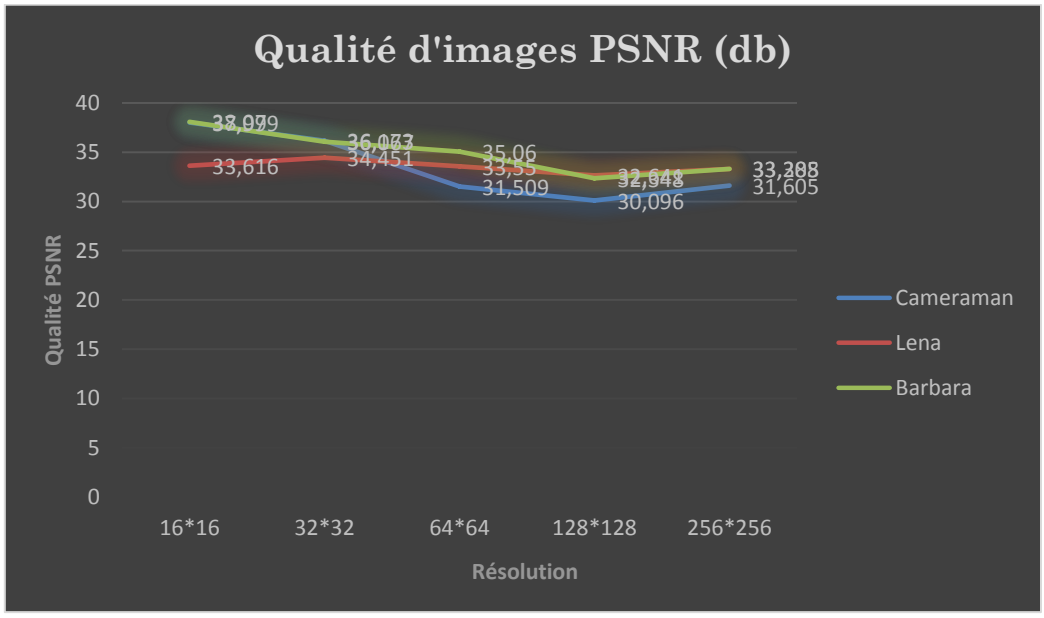


Figure 5.6: Qualité en PSNR vs variation de résolution

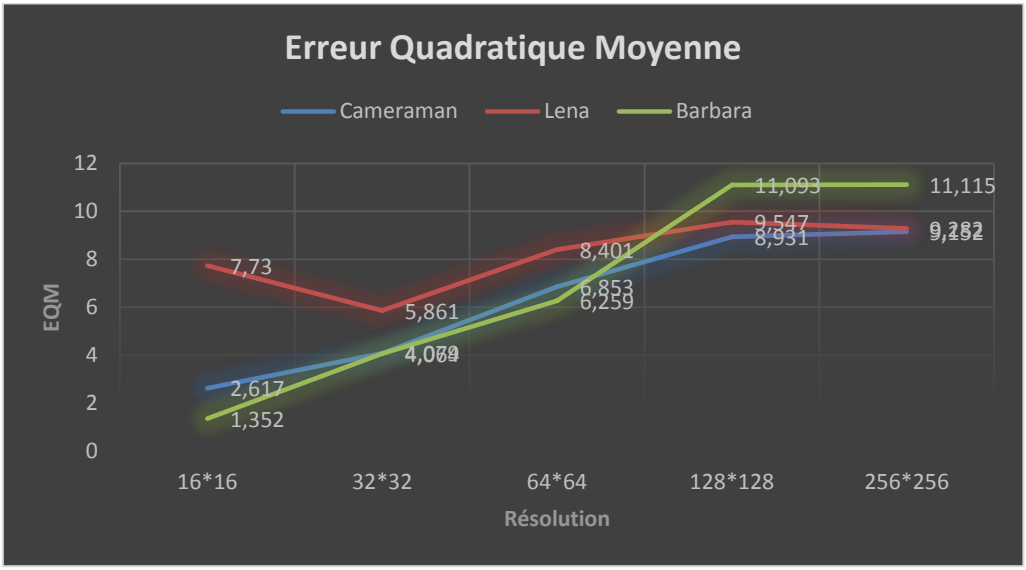


Figure 5.7 : Qualité en EQM vs variation de résolution

Les trois courbes précédentes montrent bien l'influence de la résolution de l'image en question sur les mesures de qualité de l'algorithme de compression, où on trouve que le temps de compression augmente proportionnellement avec l'accroissement de la résolution de l'image, contrairement au PSNR qui diminue indiquant une dégradation de plus dans la qualité d'image cette dégradation est confirmée par une augmentation dans la valeur de l'erreur quadratique moyenne. De ce qui précède on voit que notre méthode est en concordance parfaite avec celles qui existent dans la littérature

On va maintenant s'intéresser à la comparaison de notre approche avec les méthodes citées dans le chapitre 3. Le tableau suivant expose la différence remarquable entre notre méthode et les autres :

Images de Test	**Méthodes**	**PSNR (dB)**	**Temps (s)**	**Taux (CR)**
Lena 128*128	WPA	32.641	34.257	1.480
	Travaux de Suman K. Mitra et al. [Mit 98]	30.22	/	1.059
	Travaux de Vishvas V. Kalunge et al. [Vis 12]	/	67	/
Lena 256*256	WPA	33.305	668.810	1.596
	Travaux de Y. Chakrapani et al. [Cha 10]	26.22	2370	1.3

	Recherche exhaustive	32.69	8400	1.3
	DWSR [Wan & Zha 14]	25.8212	56.4247	1.56355
	PSO-RCQP [Xin et al 15]	27.089	6.453	1.6392
Cameraman 256*256	WPA	31.605	774.438	1.741
	PSO-RCQP [Xin et al 15]	26.686	268	1.8212
Barbara 128*128	WPA	32.348	22.713	1.310
	Travaux de Vishvas V. Kalunge et al. [Vis 12]	/	66	/
Barbara 256*256	WPA	33.288	509.406	1.447
	Travaux de Y. Chakrapani et al. [Cha 10]	28.34	2500	1.2
	Recherche exhaustive	32.84	8400	1.2

Tableau 5.4 : comparaison de notre approche avec les autres méthodes

Notre méthode constitue un travail innovateur qui contient des résultats assez satisfaisants. Elle démarque de la qualité offerte après compression ; le temps reste très satisfaisant et le taux de compression est meilleur que celui offert par la majorité des méthodes comme le tableau 5.4 le montre.

5.4 Conclusion

Dans ce chapitre, nous avons présenté la manière dont nous avons implémenté la compression fractale par l'utilisation d'une nouvelle méta heuristique (Wolf Pack Algorithm).

Les résultats prouvent l'efficacité de l'algorithme considéré qui peut être encore amélioré par un bon choix de paramètres de mise en œuvre. En outre, l'approche démontre la capacité de tel algorithme à mener la compression d'image et ouvre de nouvelles questions dans la quête des problèmes plus complexes.

Conclusion générale

« Ayez le courage de suivre votre cœur et votre intuition. L'un et l'autre savent ce que vous voulez réellement devenir. Le reste est secondaire ».

Steve Jobs

La quantité d'information augmente plus vite que la capacité de stockage. On a besoin donc de compresser les données avant de les utilisées (Acquisition des données scientifiques, médicales, industrielles, téléphonie, conservation du patrimoine, etc.). Donc, le domaine a une longue vie devant soi. Des nouveaux importants algorithmes naissent tous les ans.

Toute compression cherche à éliminer la redondance, soit par une structuration différente, mais réversible, permettant de restaurer l'original (compression sans pertes), soit en supprimant une partie d'information considérée inutile, ou sans importance (méthodes avec pertes, irréversibles). Les méthodes irréversibles offrent un taux de compression beaucoup plus élevé que méthodes sans pertes ! Bien sûr, parfois il est hors de question de perdre l'information. Ceci dit, les critères d'évaluation des techniques de compression sont très variés.

La compression fractale est une méthode de compression irréversible, elle repose sur un principe simple : toute image comporte des similarités et des redondances. Cette méthode de compression consiste donc à détecter la récurrence des motifs à différentes échelles, et tend à éliminer la redondance d'informations dans l'image. Cette méthode est largement utilisée ces dernières années en vue de sa fidélité approximative à l'image originale, néanmoins elle est beaucoup gourmande en temps de calcul.

D'autre part, les métaheuristiques d'optimisation est une famille d'algorithmes stochastiques qui tendent de résoudre les problèmes les plus complexe et de s'appliquer généralement à tous type de problème (ou presque).

Ces méthodes sont inspirées par rapports à la physique (recuit simulé, recuit micro canonique), à la biologie (algorithmes évolutionnaires) ou à l'éthologie (colonies de fourmis, essaims particulaires).

Ce travail de thèse, est divisé en deux grandes parties essentielles qui rentrent dans le cadre de l'optimisation de la compression d'image par les métaheuristiques d'optimisation:

- Dans un premier lieu, nous avons effectué une étude synthétique sur toutes les améliorations portées sur l'algorithme de la compression fractale par l'utilisation des algorithmes d'optimisation tout en mettant l'accent sur les techniques bio-inspirées et leur impact sur ce type de compression.
- La deuxième partie représente notre contribution dans le domaine de la compression d'image, où on a appliqué, et pour la première fois, l'une des nouvelles techniques à savoir l'algorithme du pacte des loups ***(Wolf Pack Algorithm)*** sur la compression fractale d'image afin de résoudre le problème de temps de calculs ainsi de garder une qualité acceptable de l'image comprimée.

Notre travail a été suivi par l'essai d'une autre nouvelle méthode inspirée du comportement social des chauves-souris. Cette méthode est en cours de finalisation et fera l'objet d'un article.
Nous envisageons aussi d'intégrer le concept de Deep Learning dans l'optimisation de la compression d'image tout en gardant l'apport des métaheuristiques.

Références bibliographiques

Chapitre 1

[Huf 52] DAVID A. HUFFMAN. A Method of the Construction of Minimum Redundancy Codes. The Institute of Radio Engineers, volume 40, No. 9, pp. 1098_1101, 1952.

[Pau 94] Paul G. Howard and Jeffery Scott Vitter. Arithmetic Coding for Data Compression. Technical Report, Proceedings of the IEEE. Volume. 82, No. 6, pp. 857-865, 1994.

[Fon 02] B. Fong, G.Y. Hong, and A.C.M Fong. Constrained error propagation for efficient image transmission over noisy channels. IEEE Transactions on Consumer Electronics, volume 48, No. 1 pp. 49-55, 2002.

[Guy 01] A. Guyader, E. Fabre, and C. Guillemot. Joint Source-Channel Turbo Decoding of Entropy-Coded Sources IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, Volume. 19, No. 9, pp. 1680-1696. 2001.

[Shi 01] B.-J. Shieh, Y.-S. Lee and C.-Y. Lee. A new approach of group-based VLC codec system with full table programmability. IEEE Transactions on Circuits and Systems for Video Technology, Volume. 11, No. 2, pp. 210-221, 2001.

[Ziv 77] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. IEEE Transactions on Information Theory, Volume. IT-23, No. 3, pp. 337-343, 1977.

[Ric 01] Richard E. Woods Rafael C. Gonzalez. Digital Image Processing. Addison-Wesley Pub Co, ISBN: 0201180758, Paris, January 2002.

[Mor 95] N. Moreau. Techniques de compression des signaux. Masson, Paris, 1995.

[Kyu 96] Kyung Sub Joo, D. R. Gschwind, and T. Bose. ADPCM encoding of images using a conjugate gradient based adaptive algorithm. IEEE International Conference on Acoustics, Speech, and Signal Processing. Conference Proceedings, volume 4, pp. 1942-1945, 1996.

[Bab 03] M. Babel, O. Déforges, and J. Ronsin. Décomposition pyramidale à redondance minimale pour compression d'images sans perte. Groupe d'Etudes du Traitement du Signal et des Images, volume 1, pp. 778-781, 2003.

[Jos 06] José Marconi M. Rodrigues, Transfert sécurisé d'images par combinaison de techniques de compression, cryptage et marquage, thèse de doctorat à l'université Montpellier II, Soutenue publiquement le 31 Octobre 2006.

[Fis 95] Yuval Fisher, Fractal Image Compression, Theory and Application, 1995 Springer-Verlag New York, Inc, ISBN-13: 1-4612-7552-7.

[Sal 07] David Salomon, Data Compression, the complete reference, fourth edition, chap 5, Springer-Verlag London Limited 2007, ISBN-10: 1-84628-602-6.

Chapitre 2

[Dré 04] J. Dréo, Adaptation de la méthode des colonies de fourmis pour l'optimisation en variables continues. Application en génie biomédical. Thèse de doctorat en sciences, Soutenue le 13 décembre 2004. Université Paris 12. Val de Marne. UFR de Sciences.

[Sia et al 06] P. Siarry, J. Dréo, A. Pétrowski, E. Taillard. (2006). Métaheuristiques pour l'optimisation difficile, 2003, Eyrolles. ISBN : 2-212-11368-4.

[Col et al 02] Collette, Y. and Siarry, P. Optimisation multiobjectif. Eyrolles, 2002. ISBN: 2-212-11168-1.

[Cer 85] Cerny, V., Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. JOURNAL OF OPTIMIZATION THEORY AND APPLICATIONS: Vol. 45, No. l, pp 41-51. 1985.

[Met 53] Metropolis, N., Rosenbluth, M. and Teller, A.,Teller, E., Equation of state calculation by fast computing machines. THE JOURNAL OF CHEMICAL PHYSICS. Vol. 21. No. 6, pp. 1087-1092. 1953.

[Gol 86] F. Glover. (1986). Future paths for integer programming and links to artificial intelligence Computers and Operations Research, Volume. 13. No. 5, pp. 533-549, 1986.

[Gol 97] F. Glover, M. Laguna. (1997). Tabu Search, Kluwer Academic Publishers 1997. ISBN: 0-7923-8187-4.

[Fra 57] A.S. Fraser. (1957). Simulation of genetic systems by automatic digital computers, Australian Journal of Biological Sciences, Volume. 10. No. 4 pp. 484-491, 1957.

[Rec 65] I. Rechenberg. (1965). Cybernetic Solution Path of an Experimental Problem, 1965, Royal Aircraft Establishment Library Translation, 1122.

[Bey 01] H.G. Beyer. (2001). The theory of evolution strategies, Natural Computing Series, 2001, Springer-Verlag Berlin Heidelberg,2001. ISBN: 978-3-642-08670.

[Fog 66] L.J. Fogel, A.J. Owens, M.J. Walsh. (1966). Artificial Intelligence through Simulated Evolution, 1966, John Wiley & sons.

[Hol 73] J.H. Holland. (1973). Genetic Algorithms and the optimal allocation of trials, SIAM Journal of Computing, Vol. 2, pp. 88-105, 1973.

[Hol 92] J.H. Holland. (1992). Adaptation in Natural and Artificial Systems, 2nd edition, 1992, MIT Press.

[Gol 89] D.E. Goldberg. (1989). Genetic Algorithms in Search, Optimization and Machine Learning, 1989, Addison-Wesley.

[Dor 06] Dorigo, M., About ACO: Behavior of real ants. Ant Colony optimization. IEEE COMPUTATIONAL INTELLIGENCE MAGAZINE. November 2006.

[Dor et al 92] A. Colorni, M. Dorigo, V. Maniezzo. (1992). Distributed Optimization by Ant Colonies, Proceedings of the 1st European Conference on Artificial Life, 1992, pp. 134-142, Elsevier Publishing.

[Dor et al 96] M. Dorigo, V. Maniezzo, A. Colorni. (1996). Ant System : optimization by a colony of cooperating agents, IEEE transactions on systems, man, and cybernetics part B cybernetics, Volume 26, No 1,1996.

[Dor et al 05] M. Dorigo, C. Blum. (2005). Ant colony optimization theory: A survey, Theoretical Computer Science, Vol. 344, pp. 243-278, 2005.

[Tfa 07] W. Tfaili. (2007). Conception d'un algorithme de colonie de fourmis pour l'optimisation continue dynamique, Thèse de doctorat de l'Université de Paris 12-Val de Marne, 2007.

[Ken et al 95] J. Kennedy, R.C. Eberhart. (1995). _Particle Swarm Optimisation, Proceedings of the IEEE International Conference On Neural Networks, 1995, pp. 1942-1948, IEEE Press.

[Cle et al 02] M. Clerc, J. Kennedy. (2002). The particle swarm: explosion, stability, and convergence in multi-dimensional complex space, IEEE Transactions on Evolutionary Computation, Volume. 6. No 1. pp. 58-73, 2002.

[Par 1896] Pareto, V., Cours d'économie politique. LAUSANNE F. ROUGE, Editeur. 1896.

[Ish et al 98] Ishibuchi, H. and Murata, T., A multiobjective genetic local search algorithm and its application to flow shop scheduling. IEEE Trans on Syst., Man., and Cyber. - Part C: App. and Reviews. Vol. 28, No. 3, pp. 392-403. 1998.

[Tal 99] Talbi, E.G., Métaheuristiques pour l'optimisation combinatoire multi-objectif: Etat de l'art. CNET, 1999.

[Sch 85] J.D. Schaffer. (1985). Multiple objective optimization with vector evaluated genetic algorithm, Proceedings of the 1rst International Conference on Genetic Algorithm, pp. 92-101.1985.

[Ric et al 89] J.T. Richardson et al. (1989). Some guidelines for genetic algorithms with penalty functions, Proceedings of the 3rd International Conference on Genetic Algorithms, pp. 191-197, 1989, Morgan Kaufmann Publishers.

[Fre et al 93] T.L. Friesz et al.. (1993). The multiobjective equilibrium network design problem revisited: a simulated annealing approach, European Journal of Operational Research, Volume. 65. No1, pp. 44-57, 1993.

[Ulu et al 99] E.L. Ulungu, J. Teghem, P. Fortemps, D. Tuyttens. (1999). MOSA method: a tool for solving multiobjective combinatorial optimization problems, Journal of Multicriteria Decision Analysis, Volume 8, No. 4, pp 221–236, 1999.

[Eng 97] P. Engrand. (1997). A multi-objective optimization approach based on simulated annealing and its application to nuclear fuel management, Proceedings of the 5th International Conference on Nuclear Engineering, volume 30, No 2, pp. 416 423, 1997, American Society Of Mechanical Engineers.

[Fon et al 95] Fonseca, C. M. and Fleming, P. J., An overview of evolutionary algorithms in multiobjective optimization. Evolutionary computation. Volume. 3. No 1, pp. 1-16. 1995.

[Sri 95] Srinivas, N. and Deb, K., Multiobjective optimization using non-dominated sorting in genetic algorithms. Evolutionary Computation, Volume 2, No.8, pp. 221-248. 1995.

[Deb et al 02] Deb, K., et al., A fast elitist non-dominated sorting genetic algorithm for multiobjective optimization: NSGA II. IEEE Transaction on Evolutionary computation, Volume 5, No.3, pp. 115-148. 2002.

[Zit et al 02] E. Zitzler, M. Laumanns, L. Thiele. (2002). _SPEA2 : Improving the Strengh Pareto Evolutionary Algorithm for Multiobjective Optimization_, Proceedings of the Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems Conference, pp. 95-100, CIMNE, Barcelona, Spain.

[Blu 05] C. Blum. (2005). Ant colony optimization: Introduction and recent trends, Physics of Life Reviews, Volume 2, No 4, pp. 353-373, 2005.

[Cag 04] C. Gagné, M. Gravel, W.L. Price. (2004). Optimisation multiobjectif à l'aide d'un algorithme de colonies de fourmis, Information System and Operational Research, Volume. 42, No1, pp. 23-42, 2004.

[Coe 02] C.A. Coello Coello, D.A. Veldhuizen, G.B. Lamont. (2002). Evolutionary Algorithms for Solving Multi-Objective Problems, Kluwer Academic Publishers. ISBN 978-0-387-33254-3.

[Sie 06] Reyes-Sierra, M. and Coello, C. A. C., Multi-objective Particle Swarm Optimizers: A survey of the State-of-the-Art. International Journal of Computer. Intelligence Research, Volume. 2, No 3, pp. 287-308. 2006.

Chapitre 3

[Jen 09] Jeng JH, Tseng CC, Hsieh JG. Study on Huber fractal image compression. IEEE Trans Image Process, Volume 18, No 5, pp 995–1003. 2009.

[Kuo 99] Li J, Kuo CCJ. Image compression with a hybrid wavelet-fractal coder. IEEE Transaction on Image Processing; volume 8, No 6, pp. 868–74. 1999.

[Han 08] Han JHJ. Fast Fractal Image Compression Using Fuzzy Classification. FSKD '08. Fifth International Conference on Fuzzy Systems and Knowledge Discovery, 2008, pp. 272–276.

[Mit 98] Mitra SK, Murthy C a, Kundu MK. Technique for fractal image compression using genetic algorithm. IEEE Trans Image Process. Volume 7, No 4, pp. 586–593. 1998.

[Li 08] Li J, Yuan D, Xie Q, Zhang C. Fractal Image Compression by Ant Colony Algorithm. In: 2008 The 9th International Conference for Young Computer Scientists. pp. 1890–1894. 2008.

[Tse 08] Tseng C-C, Hsieh J-G, Jeng J-H. Fractal image compression using visual-based particle swarm optimization. Image Visual Computation volume 26, No 8, pp. 1154 – 1162. 2008.

[Ven 97] Lucia Vences and Isaac Rudomin. Genetic Algorithms for Fractal Image and Image Sequence Compression. Comptacion Visual 1997.

[Cha 09] Y. Chakrapani and K. Soundara Rajan. GENETIC ALGORITHM APPLIED TO FRACTAL IMAGE COMPRESSION. ARPN Journal of Engineering and Applied Sciences. Volume 4, No. 1, pp. 53-58. 2009.

[Wan 09] Wang Xing-yuan, Li Fan-ping and Wang Shu-guo. Fractal image compression based on spatial correlation and hybrid genetic algorithm. Journal of Visual Communication and Image Representation, Volume 20, No 8, pp. 505-510. 2009.

[Vis 12] Vishvas V. Kalunge and Varunakshi Bhojane. Time Optimization Of Fractal Image Compression By Using Genetic Algorithm. International Journal of Engineering Research & Technology (IJERT). Volume 1 No 10, pp. 1-5. 2012.

[Mah 13] Mahesh G. Huddar. Genetic Algorithm based Fractal Image Compression. International Journal of Modern Engineering Research (IJMER). Volume 3, No 2, pp. 1123-1128. 2013.

[Min 14] Ming-Sheng Wu. Genetic algorithm based on discrete wavelet transformation for fractal image compression. Journal of Visual Communication and Image Representation. Volume 25, No 8, pp. 1835-1841. 2014.

[Pan 15] Anamika Pandey and Anshul Singh. Fractal Image Compression using Genetic Algorithm with Variants of Crossover. International Journal of Electrical, Electronics and Computer Engineering. Volume 4, No1, pp. 73-81. 2015.

[Mar 06] Cristian Martinez. An ACO algorithm for image compression. CLEI ELECTRONIC JOURNAL, Volume 9, No 2, pp. 1-17, 2006.

[Tse 08] Chun-Chieh Tseng , Jer-Guang Hsieh and Jyh-Horng Jeng . Fractal image compression using visual-based particle swarm optimization. Image Visual Computation volume 26, No 8, pp. 1154 –1162. 2008.

[Kar 05] Karaboga, D. Artificial bee colony algorithm. Scholarpedia. Turkey: Erciyes University, Computer Engineering Department. (2005).

[Cha 10] Y. Chakrapani and K. Soundara Rajan. Implementation of fractal Image compression employing particle swarm optimization. World journal of modeling simulation, volume.6 , No.1. pp. 40-46, 2010.

[Wan et al 15] Wang Xing-Yuan, Zhang Dou-Dou, Wei Na. Fractal image coding algorithm using particle swarm optimization and hybrid Quadtree partition scheme. IET Image Processing, Volume 9, No 2, pp. 153-161. 2015.

[Wan et al 09] Wang & Lang, 2009. A fast fractal encoding method based on fractal dimension. Fractals, Volume 17, No 4, pp. 459–465. 2009.

[Wan & Zou 09] Wang & Zou, 2009. Fractal image compression based on matching error threshold. Volume 17, No 01, pp. 109–115. 2009.

[Wan et al 09] Wang, Li, & Chen, 2009. An improved fractal image coding method. Fractals, volume 17, No 4, pp. 451–457. 2009.

[Wan et al 13] Wang, Zhang, & Guo, 2013. Novel hybrid fractal image encoding algorithm using standard deviation and DCT coefficients. Nonlinear Dynamics, volume 73, No (1–2), pp. 347–355. 2013.

Chapitre 4

[Gal 03] Galabov M. Fractal image compression. Proc 4th Int Conf Conf Comput Syst Technol e-Learning - CompSysTech '03 [Internet]. 2003;43(6): pp. 320-326.

[Tha et al 13] Thanushkodi KG, Bhavani S. Comparison of fractal coding methods for medical image compression. IET Image Process. volume 7, No 7, pp 686–693. 2013.

[Jen 09] Jeng JH, Tseng CC, Hsieh JG. Study on Huber fractal image compression. IEEE Trans Image Process, Volume 18, No 5, pp 995–1003. 2009.

[Kuo 99] Li J, Kuo CCJ. Image compression with a hybrid wavelet-fractal coder. IEEE Transactions on Image Processing, Volume 8, No 6, pp. 868–874. 1999.

[Han 08] Han JHJ. Fast Fractal Image Compression Using Fuzzy Classification. FSKD '08. Fifth International Conference on Fuzzy Systems and Knowledge Discovery, 2008, pp. 272–276.

[Mit 98] Mitra SK, Murthy C a, Kundu MK. Technique for fractal image compression using genetic algorithm. IEEE Trans Image Process. Volume 7, No 4, pp. 586–593. 1998.

[Li 08] Li J, Yuan D, Xie Q, Zhang C. Fractal Image Compression by Ant Colony Algorithm. In: 2008 The 9th International Conference for Young Computer Scientists. pp. 1890–1894. 2008.

[Tse 08] Tseng C-C, Hsieh J-G, Jeng J-H. Fractal image compression using visual-based particle swarm optimization. Image Visual Computation volume 26, No 8, pp. 1154 – 1162. 2008.

[Ola et al 08] Olamaei J, Niknam T, Gharehpetian G. Application of particle swarm optimization for distribution feeder reconfiguration considering distributed generators. Applied Mathematics and Computation. Volume 201, No 1–2, pp. 575-586, 2008.

[Gan et al 11] Gandomi AH, Alavi AH. Multi-stage genetic programming: A new strategy to nonlinear system modeling. Information Sciences, volume 181, No 23, pp 5227–5239. 2011.

[Blu et al 08] Blum C, Li X. Swarm Intelligence in Optimization. Swarm Intelligence Introduction, pp. 43–85. 2008.

[Fel 07] Felix TSC, Manoj KT. Swarm Intelligence, Focus on Ant and Particle Swarm Optimization. In: Felix TSC, Manoj KT, editors. Numerical Analysis and Scientific Computing. I-Tech Education and Publishing; 2007. ISBN 978-3-902613-09-7.

[Lu 93] Lu G. Fractal image compression. Signal Processing: Image Communication, volume 5, No 4, pp. 327–343. 1993.

[Hut 81] Hutchinson J. Fractals and self-similarity. Indiana University Mathematics Journal, volume 30, No 5, pp. 713–747. 1983.

[Bar 95] Barnsley MF, Demko S. Iterated Function Systems and the Global Construction of Fractals. Proc of the Royal Society Mathematical, Physical and Engineering Sciences volume 399, pp. 243–275. 1985.

[Pei et al 04] Peitgen H-O, Jürgens H, Saupe D. Chaos and Fractals [Internet]. Mathematica. 2004. ISBN 978-1-4684-9396-2.

[Gei et al 94] Geir Egil Øien. Skjalg Lepsøy. Fractal-based image coding with fast decoder convergence. Signal Processing. Volume 40, No 1, pp. 105-117. 1994.

[Moo et al 99] Ho Moon Y, Soon Kim H, Shin Kim Y, Ho Kim J. A novel fast fractal-decoding algorithm. Signal Processing: Image Communication, volume 14, No 4, pp. 325–333. 1999.

[Moo et al 97] Moon YH, Baek KR, Kim YS, Kim JH. Fast fractal decoding algorithm with convergence criteria. Society of Photo-Optical Instrumentation Engineers, volume 36, No 7, pp 1992–1999. 1997.

[Jac 92] Jacquin AE. Image Coding Based on a Fractal Theory of Iterated Contractive Image Transformations. IEEE Transactions on Image Processing. Volume1, No 1, pp. 18-30. 1992.

[Jac 93] Jacquin AE. Fractal image coding: a review. Proceedings of the IEEE. Volume: 81, No 10, pp. 1451–1465. 1993.

[Tho et al 95] Thomas L, Deravi F. Region-based fractal image compression using heuristic search. IEEE Transactions on Image Processing. Volume 4, No 6, pp. 832–838. 1995.

[Car 01] Cardinal J. Fast fractal compression of greyscale images. IEEE TRANSACTIONS ON IMAGE PROCESSING, Volume 10, No. 1, pp. 159–164. 2001.

[He et al 06] He C, Xu X, Yang J. Fast fractal image encoding using one-norm of normalised block. Chaos, Chaos, Solitons & Fractals. Volume 27, No 5, pp 1178-1186. 2006.

[Ton et al 01] Tong CS, Pi M. Fast fractal image encoding based on adaptive search. Transactions on Image Processing. Volume 10, No9, pp. 1269–1277. 2001.

[Har et al 00] Hartenstein H, Saupe D. Lossless acceleration of fractal image encoding via the fast Fourier transform. Signal Processing: Image Communication. Volume 16, No4, pp. 383–394. 2000.

[Zha et al 07] Zhang C, Zhou Y, Zhang Z. Fast Fractal Image Encoding Based on Special Image Features. Tsinghua Science and Technology. Volume 12, No 1, pp. 58–62. 2007.

[Tru et al 00] Truong TK, Jeng JH, Reed IS, Lee PC, Li AQ. A fast encoding algorithm for fractal image compression using the DCT inner product. IEEE Transactions on Image Processing. Volume 9, No 4, pp. 529–535. 2000.

[Lin et al 11] Lin YL, Wu MS. An edge property-based neighborhood region search strategy for fractal image compression. Computers & Mathematics with Applications. Volume 62, No 1, pp. 310–318. 2011.

[Wu et al 14] Wu HS, Zhang FM. Wolf pack algorithm for unconstrained global optimization. Hindawi Publishing Corporation. Mathematical Problems in Engineering. Volume 2014, Article ID 465082, pp. 1-17 2014.

Chapitre 5

[Mit 98] Mitra SK, Murthy C a, Kundu MK. Technique for fractal image compression using genetic algorithm. IEEE Trans Image Process. Volume 7, No 4, pp. 586–593. 1998.

[Vis 12] Vishvas V. Kalunge and Varunakshi Bhojane. Time Optimization Of Fractal Image Compression By Using Genetic Algorithm. International Journal of Engineering Research & Technology (IJERT). Volume 1 No 10, pp. 1-5. 2012.

[Wan & Zha 14] Wang & Zhang, 2014. Discrete wavelet transform-based simple range classification strategies for fractal image coding. Nonlinear Dynamics, volume 75, No 3, pp. 439–448. 2014.

[Wan et al 15] Wang Xing-Yuan, Zhang Dou-Dou, Wei Na. Fractal image coding algorithm using particle swarm optimization and hybrid Quadtree partition scheme. IET Image Processing, Volume 9, No 2, pp. 153-161. 2015.