

**DEMOCRATIC AND POPULAR REPUBLIC OF ALGERIA
UNIVERSITY OF CHIKH LARBI TEBESSI –TEBESSA
FACULTY OF EXACT SCIENCES AND SCIENCES OF
NATURE AND LIFE**



**MASTER THESIS -LMD-
Branch: Systems and multimedia**

Fractal image compression with bat inspired algorithm

Presented by:

Idriss GABA
Khalil TITI

Advisor

MR. Rafik MENASSEL

Jury members:

**President: DR. Chawki DJEDDI
Examiner: MR. Lakhdar LAIMECHE**

May 2017

Abstract

Abstract

The redundancy found in the uncompressed images can be reduced by image compression so that we can store or transmit images in an economic way. There are many techniques being used for this purpose but the digital media is growing fast, therefore it requires more extensive research.

Compression is used to reduce data size which may allow better storage and transfer. Actual trends of compression techniques use fractal theory algorithms, which appear to be powerful tools to improve image quality.

On the other hand, heuristics algorithms represent a set of approaches that are used to solve hard optimization tasks with rational resources consumption. They are characterized with their fast convergence and reduction of research complexity.

In this paper, we try to combine for the first time a bio-inspired heuristic called “Bat Inspired Algorithm” with fractal image compression.

A comparison is made between our proposed approach and different existing methods i.e. FIC with quadtree decomposition, FIC with WPA...etc. Results show improvements in our algorithm from different aspects (encoding time, CR, PSNR, MSE).

Keywords: Image Compression, Metaheuristics, Fractal, Bat inspired algorithm.

Résumé

La redondance trouvée dans les images non compressées peut être réduite par compression d'image afin que nous puissions stocker ou transmettre des images d'une manière économique. Il existe de nombreuses techniques à cet effet, mais les médias numériques augmentent rapidement, donc il faut plus de recherches dans ce domaine.

La compression est utilisée pour réduire la taille des données qui peut permettre un meilleur stockage et transfert. Les tendances des méthodes de compression actuelles sont celles des algorithmes de la théorie fractale, qui apparaissent comme un outil puissant pour améliorer la qualité d'image.

D'autre part, les heuristiques représentent un ensemble d'approches utilisées pour résoudre des tâches d'optimisation difficiles avec une consommation rationnelle des ressources. Ils se caractérisent par leur convergence rapide et leur réduction de la complexité de la recherche.

Dans cette étude, nous essayons de combiner pour la première fois une heuristique bio-inspirée appelée "Bat Inspired Algorithm" avec la compression fractale d'image.

Une comparaison est faite entre notre approche proposée et les différentes méthodes existantes, telles que : décomposition Quad-tree, WPA etc. Les résultats montrent des améliorations dans notre algorithme dans différents aspects (temps de codage, CR, PSNR, MSE).

Mots clés : Compression d'image, Métaheuristiques, Fractale, Bat inspired algorithm.

الملخص

يمكن تقليل التكرار الموجود في الصور غير المضغوطة بواسطة ضغط الصورة بحيث يمكننا تخزين الصور أو نقلها بطريقة اقتصادية. هناك العديد من التقنيات المستخدمة لهذا الغرض ولكن وسائل الإعلام الرقمية تنمو بسرعة، وبالتالي فإنها تتطلب بحثًا ذات نطاق أوسع.

يستخدم الضغط لتقليل حجم البيانات التي قد تسمح بتخزين ونقل أفضل تقنيات الضغط تتجه حاليًا نحو استخدام خوارزميات النظرية الكسورية، والتي يبدو أنها أداة فعالة لتحسين جودة الصورة.

من ناحية أخرى، خوارزميات الاستدلال تمثل مجموعة من النهج التي تستخدم في حل مهام التحسين الصعبة مع استهلاك معقول للموارد. وهذه الخوارزميات تتميز بتقاربها السريع وتقليلها لصعوبة البحوث.

في هذا المشروع، سنحاول الجمع، ولأول مرة بين استدلال مستوحى من البيولوجيا يسمى " bat inspired algorithm" مع ضغط الكسوري للصورة.

وقد أجريت مقارنة بين نهجنا المقترح وأساليب مختلفة، مثل " FIC with quadtree decomposition, FIC with WPA" ... الخ. تظهر النتائج تحسينات في خوارزميتنا من جوانب مختلفة (وقت الترميز، CR، MSE، PSNR).

الكلمات المفتاحية: ضغط الصور، الاستدلال، كسورية، bat inspired algorithm

Table of contents

Table of contents

Abstract	I
Résumé	II
الملخص	III
Acknowledgement	VII
Dedication	VIII
List of Figures	X
List of Equations	XI
List of Tables	XII
List of Abbreviations	XIII
General Introduction	1
Chapter1: State of the art	3
Introduction:	3
Part I: Compression.....	3
1. Data compression:	3
1.1. Definition:	3
1.2. Brief history:	3
1.3. Types:	4
1.3.1. Lossless data compression:	4
1.3.2. Lossy data compression:	4
1.4. Compression formats:	5
2. Image Compression:.....	5
2.1. Definition:.....	5
2.2. Image compression Methods:.....	6
2.2.1. Methods for Lossy compression:.....	6
2.2.2. Methods for Lossless compression:.....	7
2.3. Different image formats:.....	8
Part II: Optimization metaheuristics.....	8
1. Optimization problems:.....	8
1.1. Combinatorial optimization:.....	9
1.1.1 Exact algorithms:.....	9
1.1.2. Approximate algorithms:.....	9
2. Metaheuristics:.....	9
2.1. Definition:.....	10
2.2. Examples of metaheuristic algorithms:.....	10
2.2.1. Genetic algorithms:	10
2.2.2. The tabu-search method:	11
2.2.3. The ant colony optimization:	12
2.2.4. The Nested Partition Method:	12

2.2.5. Simulated Annealing:.....	12
Conclusion of the second part:.....	13
Conclusion:	13
Chapter 2: Proposed approach	14
Introduction:.....	14
Part I: Fractal image compression and Bat Algorithm.....	14
1. Fractal image compression:.....	14
1.1. Definition:.....	14
1.2. History:	15
1.3. Advantages and disadvantages of fractal image compression:.....	16
1.4. Iterated function system (IFS):.....	17
1.5. SELF-SIMILARITY PROPERTY:.....	18
1.6. Working approach:.....	18
1.7. Fractal Image Compression Techniques:.....	20
1.7.1. quad-tree decomposition:.....	20
1.7.2. Genetic Algorithm (GA):.....	21
1.7.3. Particle Swarm Optimization (PSO):.....	21
1.7.4. Artificial Bee Colony optimization (ABC):.....	21
1.7.5. Embedded Zero tree Wavelet (EZW) coding:.....	21
1.7.6. Wolf Pack Algorithm:.....	22
1.8. Comparison between Techniques:.....	22
2. Bat-Inspired Algorithm:.....	23
2.1. Behavior of bats:.....	23
2.2. Acoustics of Echolocation:.....	23
2.3. Bat Algorithm:.....	24
Part II: Bat Algorithm for Fractal Image Compression.....	26
1. Huffman Coding:.....	26
1.1. Huffman Encoding:.....	26
1.2. Huffman Decoding:.....	26
2. The proposed algorithm:.....	27
Conclusion:.....	29
Chapter 3: Results and Discussion	29
Introduction:.....	30
Part I: Implementation.....	30
1. Work environment:	30
1.1. Hardware:.....	30
1.2. Software:.....	31
1.2.1. Definition:.....	31
1.2.2. Work environment:.....	31
2. Implementation:.....	32
2.1. Initialization:.....	32
2.2. Encoding:.....	33
2.3. Decoding:.....	41

2.4. Results:.....	41
Part II: Tests and results.....	42
1. Settings of the Bat Algorithm:.....	42
1.1. Number of bats:.....	42
1.2. Loudness:.....	43
1.3. Frequency:.....	44
1.4. The best result:.....	45
2. Bat with different methods of FIC:.....	45
2.1. Standard FIC:.....	46
2.2. Particle Swarm Optimization:.....	46
2.3. Wolf pack algorithm:.....	47
2.4. Genetic Algorithm:.....	47
2.5. Quad-tree Decomposition:.....	48
Conclusion:.....	48
General Conclusion	49
References	50

Acknowledgements

Firstly, we thank god the almighty.

We wish to express our most sincere gratitude for all those, close or far, who have contributed to the completing of this paper, and especially to:

Our advisor, [Mr. Rafik MENASSEL](#), for his constant aid and support, as well as his beneficial advices, without which we wouldn't have been able to finish this work,

We also thank the jury members [DR. Chawki DJEDDI](#) and [MR. Lakhdar LAIMECHE](#) for having accepted to review and evaluate our project.

We also would like to address our honest acknowledgements to all the professors whom have accompanied us throughout our curriculum and have allowed us this happy event.

Dedication

To the soul of my dear late father "Allah yarhmou", who so hoped to see this day, who sacrificed himself for my success, helped me, and taught me patience and perseverance. I hope that he is proud of me as he always been. Thank you father, for making me the man I am today.

To the woman who gave me life, the source of tenderness that gave me love, and courage. My beloved mother, thank you for everything.

To my dear brother and friend: Ali.

To my dearest sisters: Khawla, Tahani, Asma, her husband Sofiane, their son Mohamed and their newborn baby Awab.

To my uncle Faycel, who took the place of my father, his sons Mohamed, Dhia, and all his family.

To my uncle Mohamed, who was my biggest supporter, thank you for everything.

To my aunt Yasmina, who filled me with affection, her son Iskander and all of her family.

To the rest of my family with all my feelings of respect.

For, Khalil, Mahdi, Oussama, and Amine, who have accompanied me on my academic journey.

To all of my friends.

To all of my colleagues in the 2017 promotion, and the rest of the students.

Idriss GABA

Dedication

I dedicate this work to my dear parents, whom have always picked me up when I was down and encouraged me to go on every adventure, especially this one. I have become who I am today thanks to your support and continuous care.

To my brothers: "Ramzi", who helped me, and guided me to achieve success during all of my studies, and to his beautiful wife "Radhia". To my best friend "Mohammed".

To my sister "Imen", who I believe embodies all things beautiful and pure.

To my dear sister "Ahlem", the kindest person in the world, and her beautiful and cute daughter "Jaida", and to the whole family of "Titi", I wish good health and success in life.

To my partner "Idriss", I would like our relationship to continue for my entire life, and may God protect it.

To my Friend (s).

To all those that I love and whom love me.

Khalil TITI

List of figures

List of Figures

Figure 1.1 Model for compression system (2)	6
Figure 1.2: Optimization Types and solutions (10)	10
Figure 2.1: example of fractal image fern (15)	15
Figure2.2: Iterated function simple (16)	17
Figure 2.3: self-similarity in Lenna image (17)	18
Figure 2.4: Partition of Range and Domain blocks (18)	19
Figure 2.5. The proposed Fractal Compression Technique.....	26
Figure 3.1: MATLAB work environment.....	30
Figure 3.2: Command window.....	31
Figure 3.3: example of sparse variable.....	39
Figure 3.4: Compress Lenna with BIA.....	40
Figure 3.5: Compress Cameraman with BIA.....	41
Figure 3.6: Lenna with PSO and BIA.....	45
Figure 3.7: Peppers with WPA and BIA.....	46

List of equations

List of Equations

$f_i: \mathbb{R}^n \rightarrow \mathbb{R}^n \mid i < N$ (2.1).....	14
$w1(X) = w1 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}$ (2.2).....	17
$T(\alpha, t_0, (i, j))_{\text{best}} \iff R_{k,l}$ (2.3).....	19
$\lambda = \frac{v}{f}$ (2.4).....	23
$MSE = \frac{1}{m n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i, j) - K(i, j)]^2$ (3.1).....	30
$PSNR = 10 \log_{10} \left[\frac{255^2}{MSE} \right]$ (3.2).....	30
$CR = \frac{\text{Uncompressed size}}{\text{Compressed Size}}$ (3.3).....	30

List of Tables

List of Tables

Table 1.1: Summary of the format (7)	8
Table 2.1: Different improvements on FIC (15)	22
Table 3.1: results for different Number of Bats.....	42
Table 3.2: results for different Loudness.....	43
Table 3.3: results for different Frequency.....	44
Table 3.4: comparison between standard FIC and BIA.....	45
Table 3.5: comparison between PSO and BIA.....	45
Table 3.6: comparison between WPA and BIA.....	46
Table 3.7: comparison between GA and BIA.....	46
Table 3.8: comparison between Quad-tree Decomposition and BIA.....	47

List of abbreviations

List of Abbreviations

GIF:	Graphics Interchange File
JPEG:	Joint Photographic Experts Group
BMP:	BitMaP
PNG:	Portable Network Graphics
TIFF:	Tagged Image File Format
MPEG:	Motion Pictures Expert Group
MP4:	Motion Picture 4
AVI:	Audio Video Interleave
DCT:	Discrete Cosine Transform
RLE:	Run length encoding
LZW:	Lempel-Ziv-Welch
FIC:	Fractal Image Compression
PIFS:	partitioned iterated function system
Bpp:	bits per pixel
GA:	Genetic Algorithm
PSO:	Particle Swarm Optimization
ABC:	Artificial Bee Colony
EZW:	Embedded Zero tree Wavelet
WPA:	Wolf Pack Algorithm
BIA:	Bat Inspired Algorithm
dB:	Decibel
MSE:	Mean Squared Error
PSNR:	Peak Signal Noise Ratio
CR:	Compression Ratio

General introduction

General Introduction:

Due to exponentially increasing size of every type of data file, despite the evolving nature of storage units, it is still never enough to save all the needed files. Therefore, compression exists to remedy this problem and allow for higher number of files to be stored.

Some types of files are more demanding in terms of storage space than others, primarily multimedia files. One of the most important types of media files that are used in almost every aspect is images.

The volume of images poses significant problems in terms of storage and transmission over networks with limited speeds. The compression of these data becomes unavoidable. Existing compression formats have shown their limitations on images with details such as text images and background areas.

In order to avoid this problem, researchers continue to develop new methods to compress images hoping that someday they will achieve a perfect compression technique that significantly saves storage space, and does not degrade the quality.

One of the most known compression techniques is the fractal image compression (FIC), this technique was introduced in 1897 by Hutson and Barnsley (reference). FIC is based on fractals (self-similarity parts) in order to compress the images. However, this technique's main issue is that it takes too much time on the encoding process.

In recent years, researchers found new techniques based on a combination between the fractal image compression and some optimization metaheuristics in order to reduce the encoding time and get better size and quality for images.

In this study, we are interested in the optimization metaheuristics on fractal image compression, by proposing a new approach that combines the bat inspired algorithm with the fractal image compression so as to make it faster to encode and get better results in both size and quality.

This document is organized as follow:

The first chapter is divided into two essential parts. The first part presents the prerequisites for understanding the context of image compression. It presents the methods of compression of the images of the two families: Lossless (without loss) and Lossy (with loss). The second part focuses on the optimization problems and their solutions and defines metaheuristics with different examples for them.

In the second chapter, we will discuss the fractal image compression in details, then we will indicate the different metaheuristics applied on FIC, and we will site their improvements. Next, we will present the Bat inspired algorithm and explain the behavior of bats and the technique they use to detect obstacles and preys. Finally, we will discuss our proposed approach, which is a combination between this optimization metaheuristic and the fractal image compression.

In the third chapter, we will explain our proposed approach for the objective of improving the fractal image compression in many aspects. To do this, we will first indicate our work environment, and then we will explain the implementation of our algorithm in details. After that, we will choose the best parameters to use in our algorithm by comparing different results using different values.

Finally, a comparison will be made between our algorithm and the different methods.

We will conclude this manuscript by presenting the potential perspectives of our approach.

Chapter: 1

State of the art

Introduction:

Because of the fast growth of the information age, the necessity for mass storage and fast communication links grows. Storing images in less memory leads to a direct reduction in storage cost and faster data transmissions. These facts explain the hard work of researchers on new image compression algorithms.

Images are stored on computers as collections of bits, pixels or points form the picture elements. Most of these data contain redundancy that can be removed. However, compression can be done in a way that the human eye does not detect a degradation on the image.

Part I: Compression

1. Data compression:

1.1. Definition:

Data Compression is a crucial process that allows the reducing data file sizes, to preserve storage space in all electronic devices. It involves encoding data in a smaller bit representation than the original. The result of compression can be either lossy, where there is a loss of information, or lossless where all data is preserved.

1.2. Brief history:

The idea of data compression began about 200 years ago. Represented as Morse code, invented in 1838 for use in telegraphy, which is based on using shorter code words for letters such as "e" and "t" that are more common in English. Modern work on data compression began in the late 1940s with the development of information theory. In 1949, **Claude Shannon** and **Robert Fano** devised a systematic way to assign code words based on probabilities of blocks. An optimal method for doing this was then found by **David Huffman** in 1951. Early implementations were typically done in hardware, with specific choices of code words being made as compromises between compression and error correction. In the mid-1970s, the idea emerged of dynamically updating code words for Huffman encoding, based on the actual data encountered. And in the late 1970s, with online storage of text files becoming common, software compression programs began to be developed, almost all based on adaptive Huffman coding. In 1977, **Abraham**



Lempel and **Jacob Ziv** suggested the basic idea of pointer-based encoding. In the mid-1980s, following work by **Terry Welch**, the so-called LZW algorithm rapidly became the method of choice for most general-purpose compression systems. It was used in programs such as PKZIP, as well as in hardware devices such as modems. In the late 1980s, digital images became more common, and standards for compressing them emerged. In the early 1990s, lossy compression methods also began to be widely used. Current image compression standards include GIF (LZW); JPEG (lossy discrete cosine transform, then Huffman or arithmetic coding); BMP (run-length encoding, etc.); TIFF (FAX, JPEG, GIF, etc.). Typical compression ratios currently achieved for text are around 3:1, for line diagrams and text images around 3:1, and for photographic images around 2:1 lossless, and 20:1 lossy [1].

1.3. Types:

1.3.1. Lossless data compression:

With lossless compression, every single bit of data that was originally in the file remains after the file is uncompressed. All of the information are completely restored. This is generally the technique of choice for text or spreadsheet files, where losing words or financial data could pose a problem. The Graphics Interchange File (GIF) is an image format used on the Web that provides lossless compression. Lossless compression algorithms usually exploit statistical redundancy in such a way as to represent the sender's data more concisely without error. Lossless compression is possible because most real-world data has statistical redundancy [2] .

1.3.2. Lossy data compression:

Another type of compression, called lossy data compression is possible if some loss of reliability is acceptable. Lossy compression reduces a file by permanently eliminating certain information, especially redundant information. When the file is uncompressed, only a part of the original information is still there (although the user may not notice it). Lossy compression is generally used for video and sound, where most users will not detect a certain amount of information loss. Generally, a lossy data compression will be guided by research on how people recognize the data in question. For example, the human eye is more sensitive to subtle variations in luminance than it is to difference in color. JPEG image compression works in



part by "rounding off" some of this less-important information. Lossy data compression provides a way to obtain the best reliability for a given amount of compression. Lossy image compression is used in digital cameras, to increase storage capacities with minimum ruin of picture quality [2].

1.4. Compression formats:

There are as many types of data compression as there are types of files.

In audio files, there is MP3, M4A, and AMR... etc.

In video file compression there is MPEG, MP4, 3GP, AVI... etc.

In addition, image compression types include JPEG, GIF, PNG, and TIFF... etc. Which will be the focus of our research.

2. Image Compression:

2.1. Definition:

We can define image compressing as an application of data compression that encodes the original image with fewer bits, in other words image compression reduces the amount of information necessary for a visual representation close to the original image.

The ideal goal of this application is to reduce a graphics file size without degrading the quality to a mediocre level. Not only does this process allow a higher number of images to be stored in a given memory space, it also minimizes the amount of time needed to transfer images throughout the WEB [3].



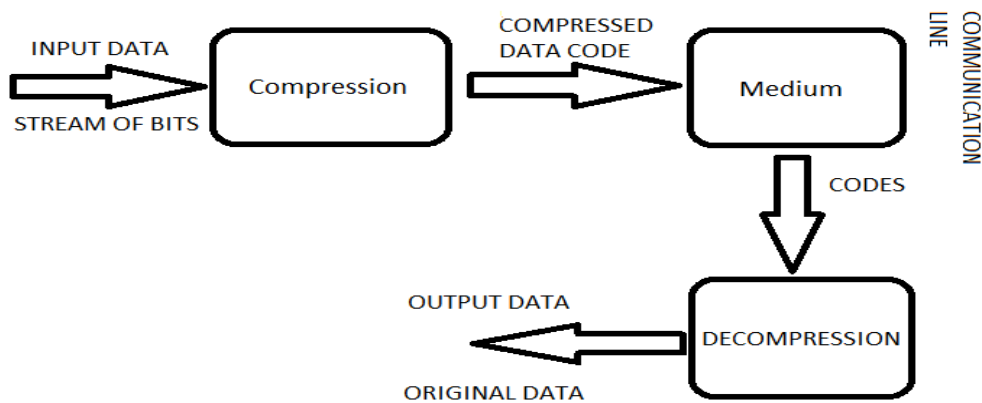


Figure 1.1 Model for compression system [2].

2.2. Image compression Methods:

2.2.1. Methods for Lossy compression:

1. Color space: It reduces the color space to the most common colors in the image. The selected colors are specified in the color palette in the header of the compressed image. Each pixel just references the index of a color in the color palette [4].

2. Chroma subsampling: This takes advantage of the fact that the human eye perceives spatial changes of brightness more sharply than those of color, by averaging or dropping some of the chrominance information in the image [4].

3. Transform coding: This is the most commonly used method. In particular, a Fourier-related transform such as the Discrete Cosine Transform (DCT) is widely used. The more recently developed wavelet transform is also used extensively, followed by quantization and entropy coding [5].

4. Fractal Compression: Fractal Image Compression technique identify possible self-similarity within the image and used to reduce the amount of data required to reproduce the image. Traditionally these methods have been time consuming, but some latest methods promise to speed up the process [5].



2.2.2. Methods for Lossless compression:

1. Run length encoding: (RLE) is a very simple form of data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run. This is most useful on data that contains many such runs: for example, simple graphic images such as icons, line drawings, and animations. It is not useful with files that do not have many runs as it could greatly increase the file size [5].

2. Huffman Encoding: an algorithm for the lossless compression of files based on the frequency of occurrence of a symbol in the file that is being compressed. The Huffman algorithm is based on statistical coding, which means that the probability of a symbol has a direct bearing on the length of its representation. The more probable the occurrence of a symbol is the shorter will be its bit-size representation. In any file, certain characters are used more than others. Using binary representation, the number of bits required to represent each character depends upon the number of characters that have to be represented. Huffman compression is a variable-length coding system that assigns smaller codes for more frequently used characters and larger codes for less frequently used characters in order to reduce the size of files being compressed and transferred [5].

3. LZW Compression: is named after its developers, A. Lempel and J. Ziv, with later modifications by Terry A. Welch. It is the foremost technique for general-purpose data compression due to its simplicity and versatility. Typically, you can expect LZW to compress text, executable code, and similar data files to about one-half their original size. LZW also performs well when presented with extremely redundant data files, such as tabulated numbers, computer source code, and acquired signals. Compression ratios of 5:1 are common for these cases. LZW is the basis of several personal computer utilities that claim to "double the capacity of your hard drive" [4].

4. SCZ CODING: is a simple set of compression routines for compressing and decompressing arbitrary data. The initial set of routines implement new lossless compression algorithms with perfect decompression. The library is called SCZ, for simple compression format. SCZ is intended as subroutines for calling within your own applications without legal



or technical encumbrances. It was developed because the standard compression routines, such as JPEG, GIF, etc., are fairly large, complex, and difficult to integrate-with, maintain, understand, have external dependencies [6].

2.3. Different image formats:

FORMAT	NAME	CHARACTERISTICS
BMP	Windows bitmap	Uncompressed format
TIFF	Tagged Image File Format	Lossless: Document scanning and imaging format. Flexible: LZW, CCITT, RLE,
PNG	Portable Network Graphics	Lossless: improve and replace GIF. Based on the DEFLATE algorithm.
JPEG	Joint Photographic Experts Group	Lossy: big compression ratio, good for photographic images
JPEG 2000	Joint Photographic Experts Group 2000	Lossy: eventual replacement for JPEG

Table 1.1: Summary of the format [7].

As we can conclude from the chart above each type of image format has its strengths and weaknesses, which makes them individually advantageous or disadvantageous depending on the required purpose.

Part II: Optimization metaheuristics

1. Optimization problems:

Optimization problems are common in many disciplines and various domains. In optimization problems, we have to find solutions, which are optimal or near optimal with respect to some goals. Usually, we are not able to solve problems in one-step, but we follow some process, which guides us through problem solving. Often, the solution process is separated into different steps, which are executed one after the other. Commonly used steps are recognizing and defining problems, constructing and solving models, and evaluating and implementing solutions [8].

An optimization problem can be defined as a finite set of variables, where the correct values for the variables specify the optimal solution. If the variables range over real numbers,



the problem is called continuous, and if they can only take a finite set of distinct values, the problem is called combinatorial. In our case, we are dealing with combinatorial optimization problems because the number of communities of interests is finite.

1.1. Combinatorial optimization:

The combinatorial optimization involves problems in which their set of feasible solutions is discrete or can be reduced to a discrete one, and the goal is to find the best possible solution. In areas such as routing, task allocation, scheduling, and so forth, most of the problems are modelled in the form of combinatorial optimization problems [9].

Two classes of algorithms are available for the solution of combinatorial optimization problems:

1.1.1 Exact algorithms:

Are guaranteed to find the optimal solution and to prove its optimality for every finite size instance of a combinatorial optimization problem within an instance dependent run time [10].

1.1.2. Approximate algorithms:

Often also called heuristic methods or simply heuristics, seek to obtain good, that is, near-optimal solutions at relatively low computational cost without being able to guarantee the optimality of solutions [10].

2. Metaheuristics:

Metaheuristics have been established as one of the most practical approach to simulation optimization. However, these methods are generally designed for combinatorial optimization, and their implementations do not always adequately account for the presence of simulation noise. Research in simulation optimization, on the other hand, has focused on convergent algorithms, giving rise to the impression of a gap between research and practice. This chapter surveys the use of metaheuristics for simulation optimization, focusing on work bridging the current gap between the practical use of such methods and research, and points out some promising directions for research in this area. The main emphasis is on two issues: accounting



for simulation noise in the implementation of metaheuristics, and convergence analysis of metaheuristics that is both rigorous and of practical value [11].

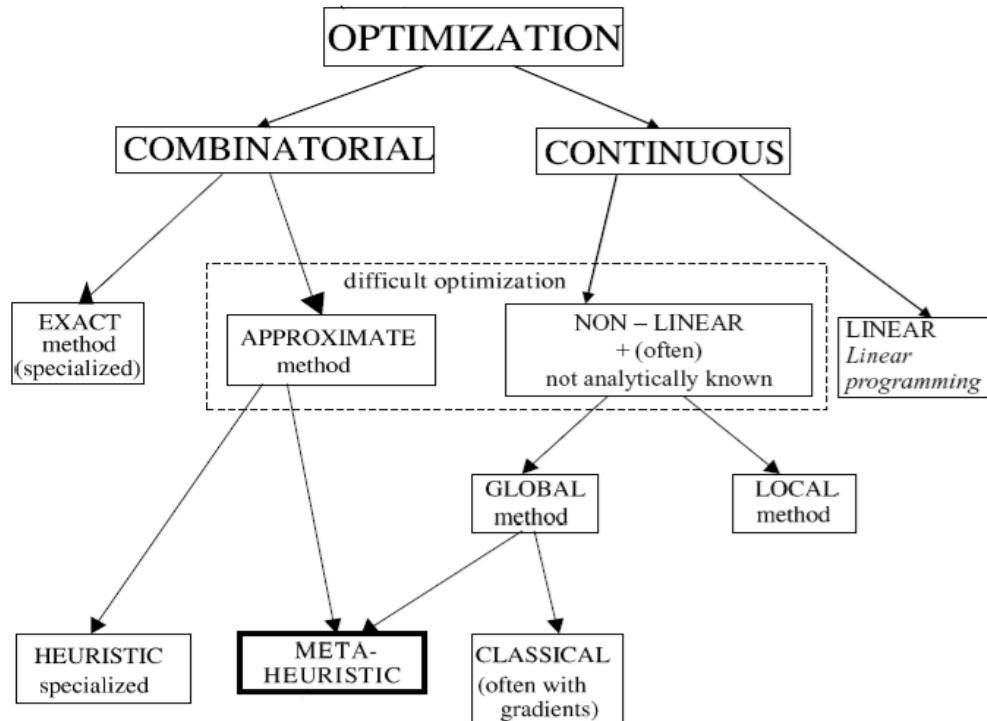


Figure 1.2: Optimization Types and solutions [10].

2.1. Definition:

A metaheuristic is a set of algorithmic concepts that can be used to define heuristic methods applicable to a wide set of different problems. It can be seen as a general-purpose heuristic method toward promising regions of the search space containing high-quality solutions [10].

A metaheuristic is a general algorithmic framework, which can be applied to different optimization problems with relatively few modifications to make them adapted to a specific problem [10].

2.2. Examples of metaheuristic algorithms:

2.2.1. Genetic algorithms:

This section looks closer at one of the popular metaheuristics for simulation optimization, namely genetic algorithm. As an approach to global optimization, genetic



algorithms (GA) have been found to be applicable to optimization problems that are intractable for exact solutions by conventional methods (Holland 1975, Goldberg 1989). It is a set-based search algorithm, where at each iteration it simultaneously generates a number of solutions. In each iteration, a subset of the current set of solutions is selected based on their performance and these solutions are combined into new solutions. The operators used to create the new solutions are survival, where a solution is carried to the next iteration without change, crossover, where the properties of two solutions are combined into one, and mutation, where a solution is modified slightly. The same process is then repeated with the new set of solutions. The crossover and mutation operators depend on the representation of the solution, but not on the evaluation of its performance. They are thus the same even though the performance is estimated using simulation. The selection of solutions, however, does depend on the performance. The general principle is that high performing solutions (which in genetic algorithms are referred to as fit individuals) should have a better chance of both surviving and being allowed to create new solutions through crossover [11].

2.2.2. The tabu-search method:

Tabu search was introduced by Glover (1989, 1990) to solve combinatorial optimization problems and it has been used effectively for simulation optimization, most notably by the OptQuest simulation optimization software (April et al. 2003). It is a solution-to-solution method and the main idea is to make certain moves or solutions tabu, that is they cannot be visited as long as they are on what is called the tabu list. The tabu list L_k is dynamic and after each move, the latest solution θ_k , or the move that resulted in this solution, is added to the list and the oldest solution or move is removed from the list. Another defining characteristic of tabu search is that the search always selects the best non-tabu solution from the neighborhood, even if it is worse than the current solution. This allows the search to escape local optima, and the tabu list ensures that the search does not revert back. Tabu search numerous other elements, such as long-term memory that restarts the search, with a new tabu list, at previously found high quality solutions, and a comprehensive treatment of this methodology can be found in Glover and Laguna (1997) [11].



2.2.3. The ant colony optimization:

Ant colony optimization is a technique for optimization that was introduced in the early 1990's. The inspiring source of ant colony optimization is the foraging behavior of real ant colonies. This behavior is exploited in artificial ant colonies for the search of approximate solutions to discrete optimization problems, to continuous optimization problems, and to important problems in telecommunications, such as routing and load balancing. First, we deal with the biological inspiration of ant colony optimization algorithms. We show how this biological inspiration can be transferred into an algorithm for discrete optimization. Then, we outline ant colony optimization in more general terms in the context of discrete optimization, and present some of the nowadays best performing ant colony optimization variants. After summarizing some important theoretical results, we demonstrate how ant colony optimization can be applied to continuous optimization problems [12].

2.2.4. The Nested Partition Method:

Introduced by Shi and Ólafsson (2000a), the nested partition method (NP) is another metaheuristic for combinatorial optimization that is readily adapted to simulation optimization problems (Shi and Ólafsson 2000b). The key idea behind this method lies in systematically partitioning the feasible region into subregions, evaluating the potential of each region, and then focusing the computational effort to the most promising region. This process is carried out iteratively with each partition nested within the last. The computational effectiveness of the NP method relies heavily on the partitioning, which, if carried out in a manner such that good solutions are clustered together, can reach a near optimal solution very quickly [11].

2.2.5. Simulated Annealing:

SA is so named because of its analogy to the process of physical annealing with solids, in which a crystalline solid is heated and then cooled slowly until it reaches its most regular crystal lattice configuration (i.e., its minimum state energy). When the cooling schedule is sufficiently slow, the resulting structure is free of crystal defects.

SA establishes a connection between this thermodynamic process and the search for heuristic solutions for optimization problems. The algorithm starts from a heuristic solution and at each iteration tries to improve its value. Improving solutions are always accepted,



while non-improving solutions are accepted only under given conditions. The probability of accepting non-improving moves is indeed proportional to a parameter, defined temperature in SA literature, which typically decreases during the execution of the approach. The key feature is that SA provides a means to escape from poor local optima, by allowing hill-climbing moves. As the temperature decreases, tending toward zero, the worsening moves are accepted with less frequency, and the solution tends to a (local or possibly global) optimum [13].

Conclusion of the second part:

In the second part, we presented the optimization problems and cited its solutions, then we went into details in metaheuristics and their methods.

Conclusion:

As of yet there are still no ideal image compression techniques that are immune to the disadvantages of the previously mentioned formats, also there are many metaheuristics that can be used to improve the compression technics. In the second chapter, we try to combine metaheuristics with a standard image compression algorithm in order to perform it and get better results.



Chapter: 2

Proposed approach

Introduction:

FIC is a successful technique of lossy image compression, however this method has its flaws, mainly the encoding time, and therefore researchers try to optimize it by combining it with optimization metaheuristics.

In this chapter, we will discuss fractal image compression in general and we will indicate the different optimization methods that were created to improve it, and finally we will describe our proposed approach in details.

Part I: Fractal image compression and Bat Algorithm

1. Fractal image compression:

1.1. Definition:

The fractal image compression (FIC, in short) is a modern technique for lossy image compression; Hutson and Barnsley originally introduced it in 1987. FIC is a technique, which is used to encode the image in such a way that it reduces the storage space by exploring the self-similarities between different isolated image regions and store only the parameters of contract transform instead of the image pixels. This principle allows for the construction of an approximation of the original image by detecting the recurrence of the patterns on various scales, and tends to eliminate the information redundancy in the source image in order for the result to be accurate enough to be acceptable. The FIC is based on an Iterated function system f , a finite set of contractions defined on a metric space R^n by the relation [14]:

$$f_i: R^n \rightarrow R^n \mid i < N \quad [2.1]$$

FIC is based on fractals (self-similar parts), which are used in order to compress image. Fractal algorithms convert these parts (referred as fractals) or geometric shapes into mathematical information, and they called as ‘fractal codes’ which are later used to reconstruct an image.

The following figure demonstrates an example of fractal images.



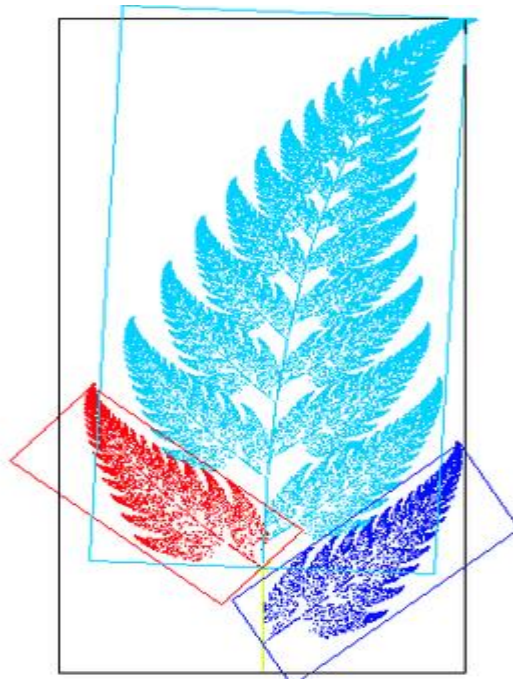


Figure 2.1: example of fractal image **fern** [15]

However, an asymmetric process characterizes this technique, and it spends much more time in the encoding process in comparison with the decompression process. FIC consists of searching for the best match block, mostly on large sized images. That is why this technique is better suited for textures and low-resolution patterns.

1.2. History:

The conservation of resolution as well as the high ratio of compression have made FIC one of most favorable techniques in image encoding. Its history dates back to the 90s where Jacquin proposed the first image compression method; its idea is to divide the image onto squared domain blocks. The principle of compression proposed is to look for the most matched domain block corresponding to each range block, determine the appropriate contract transform and store their parameters [14].

The idea was interesting but it remains limited to domestic applications due to high time consuming restrictions. Since that, researchers introduced new ideas in order to reduce the huge encoding time; the work of Thomas and Deravi combines range blocks and by utilizing region-growing method makes them more adaptive with image content [14].



Cardinal proposed a similar idea; it is based on a geometrical partition of the greyscale image block feature space. The experimental comparisons with previously published methods show a significant improvement in speed with no quality loss. Cardinal thought of employing the one-norm of normalized block to circumvent the disproportionate search in block matching. By another way, Chong and Pi presented a new adaptive search approach to reduce the computational complexity of fractal encoding; in order to exclude many unreserved domain blocks to accelerate the compression of fractal images [14].

Various other researches were introduced new concepts to improve the search quality such the encoding via the Fourier transform, special image features, DCT inner product. The most approaches were based on matching error threshold to restrict the searching space.

Recently, Lin and Wu proposed a search strategy based on image block edge property, which demonstrates an acceptable performance. Furthermore, numerous research papers have been published during last decay; they have enhanced the quality of image without improvement in resources of coding process [14].

1.3. Advantages and disadvantages of fractal image compression:

Fractal image compression has its merits and demerits compared to the other methods; we summarize them as follows [15]:

Advantages:

- Good mathematical encoding frame.
- Resolution-free decoding.
- High compression ratio.

Disadvantages:

- Slow encoding.



1.4. Iterated function system (IFS):

Michael Barnsley explained that we can represent an image as a set of mathematical equations, in which the basis of FIC is formed as an IFS code. However, because of its complexity, this idea became unwieldy [15].

Arnaud Jacquin created an enhancement to IFS by using partitioned iterated function system (PIFS). This function comprises metric space X, a set of sub domain Di, (I=1..n) and a group of contractive mappings Wi: Di → X, I=1.....n.

Images with IFS are named affine transformations; they can be a mixture of transformations (translation, rotation and scaling). Wi is the affine transformation on I → I²

$$w1(X) = w1 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix} \tag{2.2}$$

Where a, b, c, d, e and f are coefficient, which determines **the rotation, skew and scaling.**

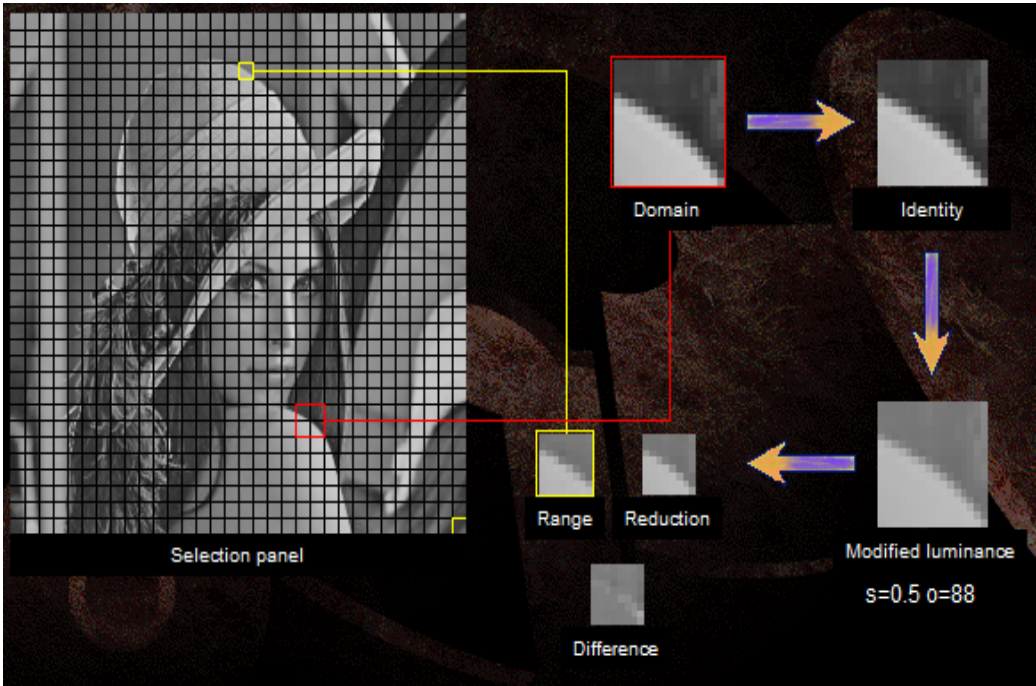


Figure2.2: Iterated function simple [16].



1.5. Self-similarity property:

The self-similarity found in fractals does not exist in a typical image. Because normal images contain a different kind of self-similarity. The Lenna figure shows blocks that are similar at different scales. For example, a part of her shoulder overlaps a smaller block and they look almost identical [17].



Figure 2.3: self-similarity in Lenna image [17].

As we can observe from the figure above, parts of the image are self-similar with well transformed parts, but the entire image is not self-similar. That is why the main purpose of FIC is to eliminate the redundancy of these self-similar parts.

1.6. Working Approach:

Suppose that we have a 128×128 image where each pixel is represented between 1 to 256 levels of grey. The image will be partitioned to non-overlapping blocks r_i of size $s \times s$ called range blocks, then the image is reduced to 64×64 by averaging (down sampling and low-pass-filtering) [18].

The resulting image will also be partitioned to non-overlapping blocks d_i with the same size of r_i which are called domain blocks. After that each changed domain block $T(D_{i,j})$ will be compared to each range block $R_{k,l}$, in order to find the most similar domain block to range block.



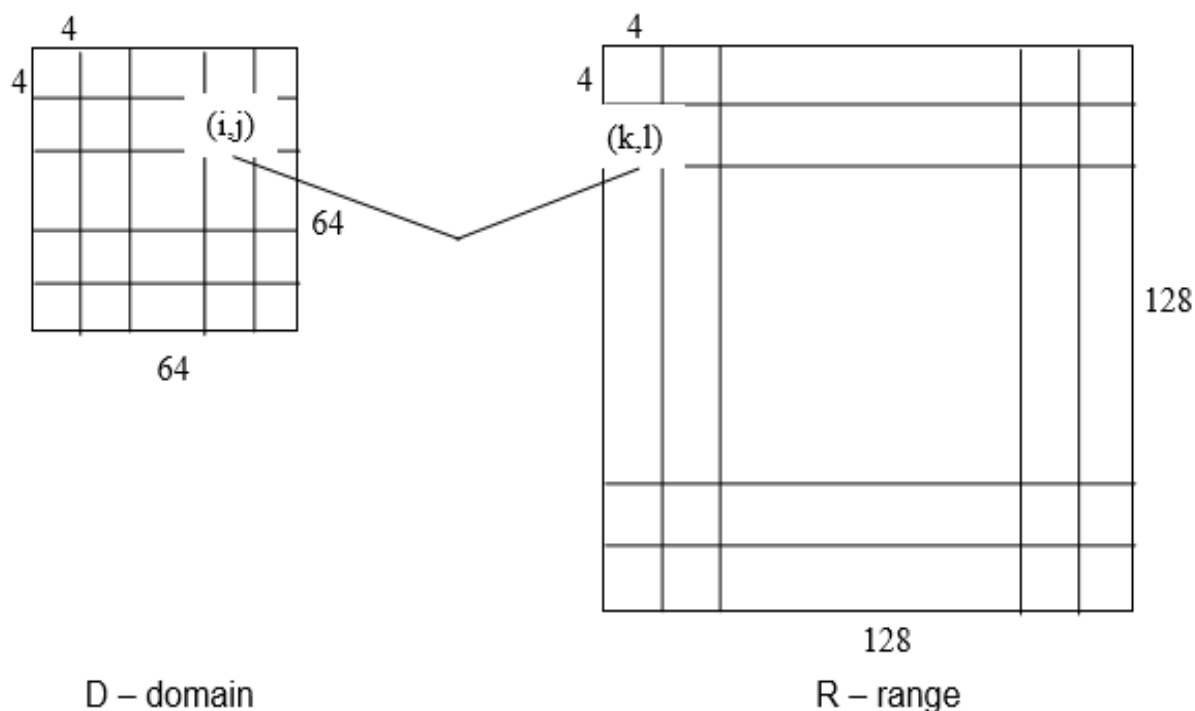


Figure 2.4: Partition of Range and Domain blocks [18]

Following that, the transformed domain block that is found to be the most similar to the range block, is allocated to the same range block, i.e. the position of the domain block and the coefficient of the transformation that was applied to the domain block, which are saved into a file labeling all the transformation. This file called the Fractal Code Book [18].

$$T(\alpha, t_0, (i,j))_{\text{best}} \implies R_{k,l} \quad [2.3]$$

Finally, Fractal decoding methods consist of the reconstruction of the range blocks from the most similar domain block by using the transformations defined in the Fractal Code Book.



The following algorithm explains the fractal image compression:

- Step 1: Read the binary image
- Step 2: Convert it into gray level image
- Step 3: Divide the image into small blocks without overlapping
Called As range blocks with $S*S$ size.
- Step 4: Introduce large square blocks, with overlapping called as
Domain blocks with $2S * 2S$ size.
- Step 5: for each range block find the matching domain block
Which closely resembles range block with respect to
Some metric.
- Step 6: Write out compressed data in form of local IFS code
- Step 7: Apply data compressed algorithm to obtain a compressed
IFS code.

1.7. Fractal Image Compression Techniques:

1.7.1. quad-tree decomposition:

It is one of the partition-based methods. It divides an image into variable size range block. In this type of partition, a square image is split into square blocks of equal sizes, and then tests each block to check whether each block meets some criteria of homogeneity. If a block meets the criteria it is not divided any further, if the block does not meet the criteria, then the block is splitted into further four blocks and again test is applied to those blocks [19]. This process is repeated iteratively until each block meets the criteria resulting in many different sizes of blocks. It is represented in a tree like structure, where each node will have four sub nodes. Adjustments of Quad-tree size is done by using two parameters, minimum level and maximum level. By this method we can increase the compression ratio and reduce the bits used to represent an image i.e. bits per pixel (bpp).



1.7.2. Genetic Algorithm (GA):

GA is an Algorithm simulating process of natural evaluation, which is applied for constraint functions and controlled parameters for optimization. GA is very effective in solving non-linear and multiple extreme problems. GA was proposed to get the matching domain block for each range block in FIC, which uses the PIFS [20]. Without needing or using an extensive search mechanism, GA tries to locate close optimum solutions.

1.7.3. Particle Swarm Optimization (PSO):

Eberhart and Kennedy [21] have created the PSO technique that is used for computation. PSO is a general-purpose optimization algorithm, which is also used for the concept of fitness. PSO based on the analogy of the group of birds. It gives mechanism that individuals in the group communicate and exchange information, which is similar to insect and human being behavior. PSO is a low-cost algorithm and can be employed in a small number of lines of code because it only needs basic mathematical operations whereas a full FIC search can find the exact best domain block for each corresponding range block. It is very time consuming, however.

1.7.4. Artificial Bee Colony optimization (ABC):

An iteration-based technique that was broadly defined by Dervis Karaboga in 2005, ABC is an algorithm that bases itself on the behavior of honeybees. It is an optimization tool, which provides a population based search procedure where each individual called food positions are altered by artificial bees with time aiming to find out the food source with large nectar amount. ABC consists of three types of bees (a) employed bee, (b) onlooker bee and (c) scout bee. The onlooker bees that are waiting in the hive receive information from the employed bees regarding the nectar sources that have been discovered before. Onlooker bees choose an exploitable food source based on the information received from the employed bees. Scout bees quest for a food source randomly within the environment in order to find nourishment [22].

1.7.5. Embedded Zero tree Wavelet (EZW) coding:

Shapiro introduced EZW. It is a wavelet-based technique used for compression [23]. EZW mainly operates on 2-D images. It provides a high compression ratio and better quality of a reconstructed image but yields lower PSNR. Here, the entire coefficient corresponding to the



same spatial location is organized in a tree-like structure. These trees have parent-child relationships among the co-efficient of sub-bands having spatial orientation.

1.7.6. Wolf Pack Algorithm:

The Wolf Pack Algorithm (WPA, in short) [14], is one of this family (bio-inspired) of algorithms that employed in order to approximate solutions for various optimization problems. WPA is a population-based metaheuristic stirred by the social hunting comportment of wolves. It consists essentially in making wolves hunt, find the trace of prey and capture it under the command of a chief wolf.

1.8. Comparison between Techniques:

Methods	Improvements on fractal image compression
quad-tree decomposition	<ul style="list-style-type: none">- Can increase the compression ratio.- Reduce the bits used to represent an image (bits per pixel).
Genetic Algorithm (GA)	<ul style="list-style-type: none">- Huge reduction in searching space and time.- Achieves high PSNR.
Particle Swarm Optimization (PSO)	<ul style="list-style-type: none">- Reduce the encoding time.
Artificial Bee Colony optimization (ABC)	<ul style="list-style-type: none">- Reduce the Compression time.
Embedded Zero tree Wavelet (EZW) coding	<ul style="list-style-type: none">- Improve the visual quality.
Wolf Pack Algorithm(WPA)	<ul style="list-style-type: none">- Achieves high Compression Ratio.- Improve the Compression Time.

Table 2.1: Different improvements on FIC [15].

As we can conclude from this table, there is still no perfect method, which improves time, quality and compression ratio together. Each of these method implemented has its strengths and weaknesses depending on their use.



2. Bat-Inspired Algorithm:

Bat Algorithm (BA) is a new metaheuristic technique proposed by Xin-She Yang in 2010, based on the echolocation performance of bats. The capability of echolocation of bats is fascinating as they can find their prey and distinguish different types of insects even in complete darkness [24].

2.1. Behavior of bats:

Being the only mammals that can truly fly, and having an advanced ability of echolocation makes bats intriguing animals. Scientists estimate there is roughly 996 different species of bats, and that accounts for up to 20% of all mammals on the planet [24].

Echolocation is a type of sonar used by bats not only to detect their preys' location and how fast they are moving, but also to circumvent obstacles and pinpoint their resting crevices in the dark. Bats unleash loud sound pulses and await the echo that reflects back from their surroundings. The pulses they emit can have varying properties and can be linked closely to the hunting strategies of bats.

2.2. Acoustics of Echolocation:

Although pulses remain often between 25 kHz and 150 kHz in a constant frequency, individual pulses only remain up to 8 to 10 ms. Bats produce between 10 and 20 ultrasonic sound bursts every second, each of which remain between 5 and 20 ms. However, when bats are hunting for their prey, and they are close by, they can speed up their pulse emission rate to a threshold of 200 /secs. Such a short sound burst is a testament to the fantastic ability of the signal processing of bats [24].

Given that the speed of sound in the air is characteristically $v = 340$ m/s, the wavelength λ of the ultrasonic sound bursts with a continual frequency f :

$$\lambda = \frac{v}{f} \quad [2.4]$$

For a typical frequency between 25 kHz to 150 kHz, wavelengths range between 2 to 14mm and are equal in order as the bats' prey sizes.



The pulses that bats produce can reach an impressive loudness of 110 dB, but auspiciously enough, these pulses remain in the ultrasonic domain. Pulse loudness can take various levels such as very loud when bats are hunting and low to a quiet sound when they are aiming for their prey. Such short pulses usually have a roaming range of few meters that depend on the frequency [24].

Research indicates that bats construct a three-dimensional layout of their surroundings by using the time delay between their ears, the variations of echo loudness and the interval between echoes' emission and detection. Bats have the ability to not only measure the distance and itinerary of their targets, but also their traveling speed and what kind they are [24].

In reality, Bats use all their senses as a combination to maximize the efficient detection of prey and smooth navigation. However, only echolocation and its accompanying behaviors are treated here.

Bats' echolocation can be formulated in an objective optimized function to create new algorithms of optimization.

2.3. Bat Algorithm:

Bats fly randomly in a search space R_i using velocity V_i at position (solution) X_i . They emit pulses at a fixed wavelength λ with varying frequency f and loudness A (varies from a large positive A_0 to a minimum constant value A_{\min}) to search for the prey [24].

As the bats select the best solutions, they generate a local solution around the selected best solutions.



The following algorithm explains the behavior and movement of bats:

```
Objective function  $f(\mathbf{x})$ ,  $\mathbf{x}=(\mathbf{x}_1, \dots, \mathbf{x}_d)^T$   
Initialize the bat population  $\mathbf{x}_i$  ( $i=1,2,\dots,n$ ) and  $\mathbf{v}_i$   
Define pulse frequency  $\mathbf{f}_i$  at  $\mathbf{x}_i$   
Initialize pulse rates  $\mathbf{r}_i$  and the loudness  $\mathbf{A}_i$   
while ( $\mathbf{t} < \text{Max number of iterations}$ )  
    Generate new solutions by adjusting frequency  
    and updating velocities and locations/solutions  
    if ( $\text{rand} > \mathbf{r}_i$ )  
        Select a solution among the best solutions  
        Generate a local solution around the selected best solution  
    end if  
    Generate a new solution by flying randomly  
    if ( $\text{rand} < \mathbf{A}_i$  &  $f(\mathbf{x}_i) < f(\mathbf{x}^*)$ )  
        Accept the new solutions  
        Increase  $\mathbf{r}_i$  and reduce  $\mathbf{A}_i$   
    end if  
    Rank the bats and find the current best  $\mathbf{x}^*$   
end while  
Postprocess results and visualization
```



Part II: Bat Algorithm for Fractal Image Compression

Fractal image compression is a modern technique used for lossy image compression. However, this technique’s main problem is that it takes a large amount of time. Moreover, the approaches that were suggested to reduce time negatively affect the quality.

To remedy that, researchers have discovered a new way to improve the fractal compression encoding by combining fractal algorithm with other coding methods (ex: FIC with quad-tree decomposition, Wolf Pack Algorithm for FIC).

Therefore, our proposed work is based on this new technique, in which a combination between the Bat-inspired algorithm and fractal image compression is made in order to improve the quality and compression time.

The following schema demonstrates our proposed algorithm.

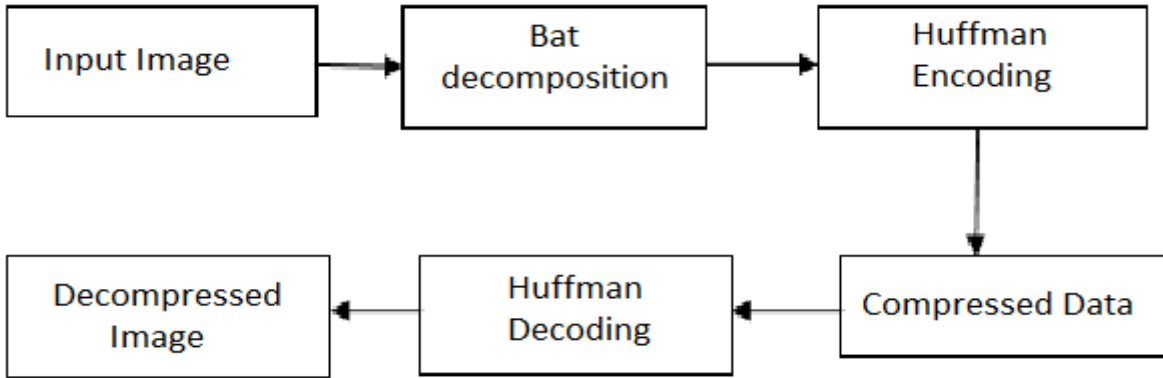


Figure 2.5. The proposed Fractal Compression Technique.

1. Huffman Coding:

1.1. Huffman Encoding:

The Huffman encoding algorithm begins with creating a list of all the symbols in a descending order of their occurrences; the next step is to construct a binary tree with a symbol at every leaf (from the bottom to the top). Each step of this procedure, two symbols with the smallest occurrence are selected, added to the top of the partial tree, deleted from the list and replaced with another symbol signifying the two original symbols [42]. After reducing the list to only one auxiliary symbol that represents the entire symbols, the Huffman tree is complete.



1.2. Huffman Decoding:

The codes have to be determined by the encoder before starting the compression, the determination is based on the probabilities of the occurrence of symbols. The probabilities have to be stored on the output as side information, in order to make any Huffman decoder capable of decompressing data [41].

The algorithm for decoding is simple.

- Start at the root and read the first bit off the input (the compressed file).
- If it is zero, follow the bottom edge of the tree;
- if it is one, follow the top edge. Read the next bit and move another edge toward the leaves of the tree.
- When the decoder arrives at a leaf, it finds there the original, uncompressed symbol, and the decoder emits that code.
- The process starts again at the root with the next bit.

2. THE PROPOSED ALGORITHM:

Our algorithm steps are as follows:

Step 1: Bats fly randomly on the image using loudness L and frequency F .

Step 2: Bats compare each block to its neighbors to see if it meets some criterion of homogeneity depending on loudness and frequency. If they meet a criterion ($\text{color_level_block} - \text{color level neighbor} \leq \text{frequency}$), they create a domain block with size $L*L$ that has only one value (average of the domain block).

Step 3: The iterations stop when bats search the entire image.

Step 4: After decomposing the image into domain blocks, the position of bats X_i and the block size blksz. will be stored in a sparse S .

Step 5: In this step, we try to find the best solution by eliminating the solution with the smaller block.



Step 6: Huffman encoding is used to store the data (positions, block sizes and values) in order to calculate the compression ratio.

Step 7: Then, Huffman decoding is utilized to restore the image data of the compressed image, after which we reconstruct it.

The following algorithm explains the workings of our approach:

```
Algorithm Fractal-with-bats
Begin
Initialization:
Generate bats (Number_bats = 1..N)
Loudness L;
Frequency F;
While not (stopping criteria)
  For each bat
    If similarity = 1
      Create domain block;
      Store position in vectors I,J;
      Store block sizes in vector blksz;
    Else
      Store position in vectors I,J;
      Store block sizes in vector blksz;
    End-if
  End-while
End-while
Search for best solutions;
Store the positions and block sizes in a sparse S;
End.
```



Conclusion:

In this chapter, we presented fractal image compression and its algorithm, then we sited different optimization methods created to improve it, after which we described the optimization metaheuristic (the bat inspired algorithm), and finally we introduced our proposed method, its properties and optimizations. In the following chapter, we will exhibit our implementation and compare the results with other methods.



Chapter: 3

Results and discussion

Introduction:

In order to carry out this project, it is necessary to choose technologies to simplify its implementation. For this, after completing the conceptual study in the previous chapter, we will discuss the implementation part in the following. We begin by presenting the hardware and software environment, then, realization of our application, and finally a comparison of our method with other FIC methods is done.

The criteria we used in the comparison are as follows:

- Encoding time.
- Decoding time.
- MSE (Mean Squared Error) presented with this formula:

$$MSE = \frac{1}{m n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i, j) - K(i, j)]^2 \quad [3.1]$$

- PSNR (Peak Signal Noise Ratio) which is calculated by this formula:

$$PSNR = 10 \log_{10} \left[\frac{255^2}{MSE} \right] \quad [3.2]$$

- Compression ratio :

$$CR = \frac{Uncompressed\ size}{Compressed\ Size} \quad [3.3]$$

Part I: Implementation

1. Work environment:

1.1. Hardware:

- Desktop PC with the following specifications:
 - AMD FX™-6100 Six-Core Processor 3.30 GHz.
 - 32 GB RAM.
 - Windows 10 64 bit.
- Dell laptop with the following specifications:
 - Intel Core i3-3217U 1.80 GHz
 - 4 GB RAM.
 - Windows 10 64 bit.



1.2. Software:

The code was implemented with MATLAB 2013a version 8.1.

1.2.1. Definition:

MATLAB (MATrix LABoratory) is a built up around vectors and matrices. MATLAB is fourth-generation programming language and it is one of the easiest programming language for writing mathematical programs. In addition, MATLAB possesses a number of toolboxes that are used for processing signals, processing images, optimization...Etc.

1.2.2. Work environment:

The following figure shows the MATLAB work environment, which contains five important windows.

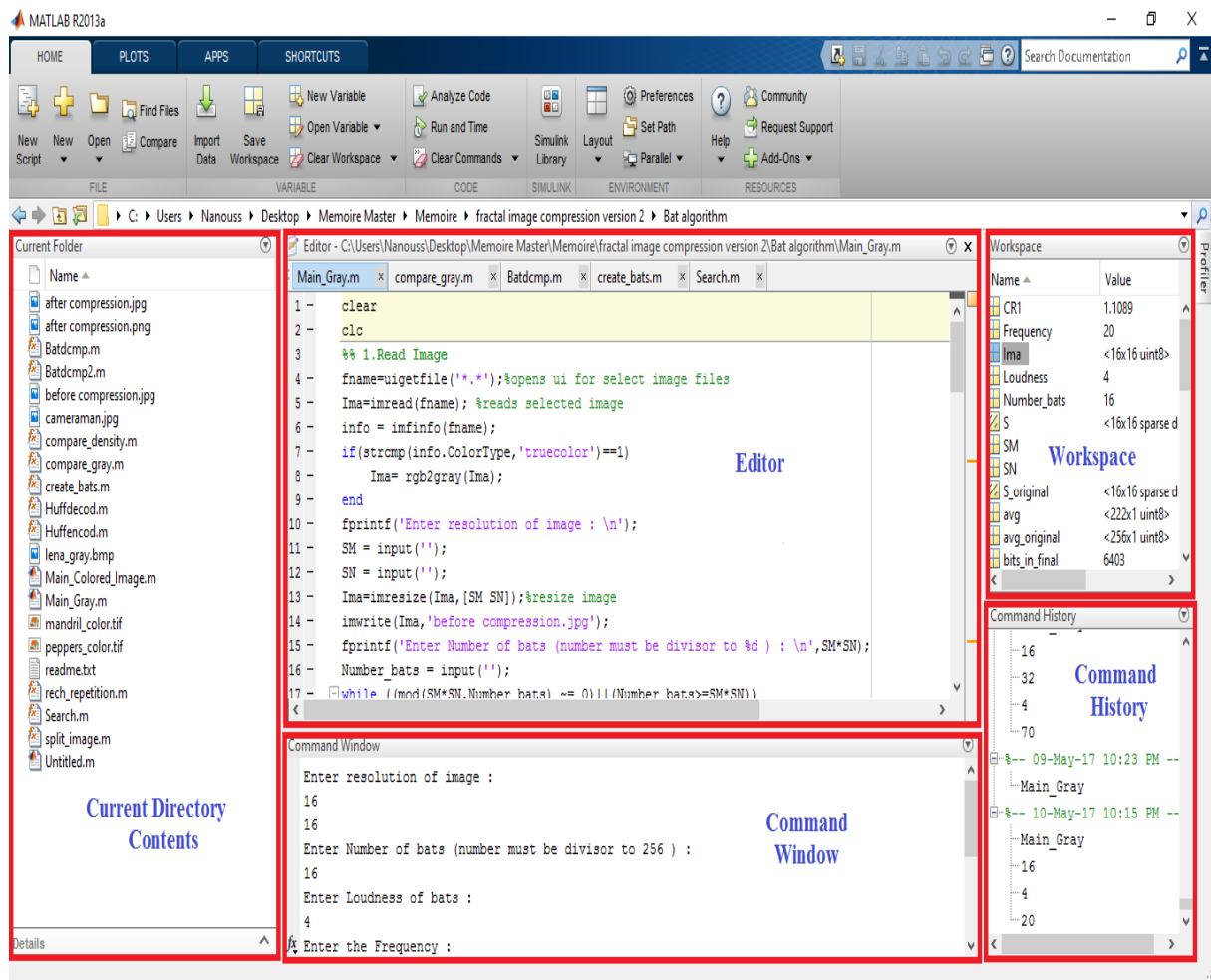


Figure 3.1: MATLAB work environment.



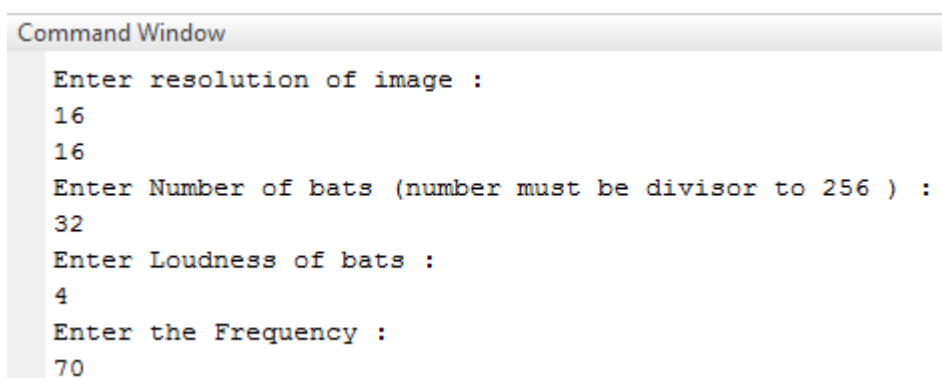
- Current Directory Contents: shows the contents of the current working folder.
- Editor: allows editing MATLAB programs and functions.
- Command Window: uses to type the command and shows the result of programs writing in the Editor.
- Workspace: displays the variables that are defined, and what type of variable each is.
- Command History: shows the commands that are already used.

2. Implementation:

2.1. Initialization:

The first step in the creation of our algorithm is to instruct the user to input the variables needed. These variables are listed below:

- Resolution of the image: the user inputs the height and the width of the image in order to resize it.
- Number of bats: this variable indicates how many bats will be spread onto the image in every iteration (the number of bats must be a divisor to Size=width x height).
- Loudness: this user-input variable is used to determine the size of the block that will be scouted by bats.
- Frequency: it regulates the level of homogeneity between pixels in the scouted block.



```
Command Window
Enter resolution of image :
16
16
Enter Number of bats (number must be divisor to 256 ) :
32
Enter Loudness of bats :
4
Enter the Frequency :
70
```

Figure 3.2: Command window.



2.2. Encoding:

The encoding process goes through multiple steps beginning with the bat decomposition and ending with the Huffman encoding.

Step 1: Bat decomposition:

```
S = Batdcmp(Ima,Loudness,Number_bats,Frequency); %% Bat decomposition
```

This function decomposes the image into small blocks depending on the similarity criteria, then it affects the results into a sparse **S**, This sparse contains the position of blocks and their sizes.

The first step of the Bat decomposition is to initialize bats and spread them on the image.

- Creating bats:

```
Bats = create_bats(SM,SN,Number_bats); %% Create Bats and set their  
Random positions
```

The input variables in this function are **width**, **height** and **number of bats**.

```
function final = create_bats(m,n,number_bats)  
resultat = cell(m,n);  
final = cell(m*n/number_bats,number_bats);  
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_bats  
    for j = 1:number_bats  
        final{i,j} = Cell{i+((j-1)*(m*n/number_bats))};  
    end  
end
```



In the beginning, an empty cell is created, the size of which is defined as: the number of rows = width * height / number of bats while the number of columns = number of bats.

After the cell is filled with the ordered positions of the entire image's pixels, the shuffling process begins. The cell is transformed into a vector, which is then shuffled using the **randperm** function. The resulting shuffled vector is transformed back into a cell.

- Start the decomposition :

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

In this step, we start a nested loop, the first one is the iteration loop and the second one is a loop to browse the bats.

After that, bats' positions (**m,n**) will be restored from the cell that was previously created.

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

Afterwards, a **Search** function is used in order to test if this position is already browsed or not.

```
if (Search(m,n,Coordinates)==0) %% check if the pixel already
used
```

The input variables in this function are **m**, **n** and **Coordinates**, which contains the positions of pixels that are already used.



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

Next, if this position does not exist in **Coordinates**, the bat starts to test the similarity by using the **compare_gray** function.

```
comp = compare_gray(m,n,Ima,Loudness,Frequency); %% check
the similarity
```

The input variables in this function are **m**, **n**, **Ima** (The image), **Loudness**, and **Frequency**.



```
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
```

The first step is to start a loop depending on the loudness, starting with the minimum value of loudness 2, then we create a vector **V1** containing the pixels that we want to compare with the input pixel, after that we will test if each pixel of 2x2 has any similarity with the input pixel. If the test is false, the function will stop and the **compare_gray** value will be 1. If the similarity test is true, **compare_gray** variable will be 2 and then it will test again on 3x3 size and so on so forth until either the **compare_gray** value is equal to loudness or similarity is not found and the **compare_gray** value equals the last size in which a similarity is found.



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

Following, if the result variable (**comp**) from the previous function equals 1 then only the position of the current pixel will be stored in **Coordinates** and the block size will also be 1, otherwise we store the position of the pixels that create a block, which starts from the current position with size **comp*comp** and store the block size that is equal to **comp**.

After this step, the bats will have scouted the entire image and all the positions and the block sizes will have been stored.

- Choose the best solution:

The **Coordinates** matrix contains redundant positions, thus, in order to eliminate the worst solution we use the **rech_repetition** function.

```
Coordinates= rech_repetition(Coordinates,blkksz);
```

The input variables in this function is **Coordinates** which contains the redundant positions and **blkksz** which contains the block sizes.



- Deleting the unnecessary positions

```
I = [I;Coordinates(:,1)];
J = [J;Coordinates(:,2)];
for z = 1:length(I)
    if(I(z)==0)
        blkosz(z)=0;
    end
    if(blkosz(z)==-2)
        blkosz(z)=0;
        I(z) = 0;
        J(z) = 0;
    end
end
I = nonzeros(I);
J = nonzeros(J);
blkosz = nonzeros(blkosz);
```

This process begins with separating x , y from **Coordinates** matrix and affecting them to **I**, **J**, Then we search for the positions with a value of 0 in order to make the value of the block sizes of these positions 0 as well.

Next, we locate the block sizes with the value -2 and give them a value of 0 instead.

Finally, we use the predefined function **nonzeros** to delete these values from the vectors (**I**, **J**, **blkosz**).

- Affect the data on a Sparse

```
S = sparse(I,J,blkosz);
```

We created a sparse where we saved the **I** and **J** positions as well as the block sizes.



(3,1)	3
(6,1)	2
(8,1)	2
(10,1)	1
(11,1)	1
(12,1)	1
(13,1)	2
(15,1)	1
(16,1)	1
(1,2)	3
(4,2)	2
(10,2)	1
(11,2)	1
(12,2)	1
(15,2)	1
(16,2)	1

Figure 3.3: example of sparse variable.

Step 2: Calculate the mean value:

This phase consists of calculating the mean value of the block

```
[i,j,blksz] = find(S); %record x and y coordinates and blocksize
blkcount=length(i); %Number of total blocks
avg=zeros(blkcount,1);%record mean values
for k=1:blkcount
    avg(k)=mean2(Ima(i(k):i(k)+blksz(k)-1,j(k):j(k)+blksz(k)-1));
    %find mean value of each block
end
avg=uint8(avg);
```

We extract the positions and the block sizes from the previously created sparse, and then we perform a loop on the image and calculate the mean value of each domain block that was created.

Step 3: Huffman Encoding:

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



Huffman encoding is a predefined function that creates a tree where the bottom contains the most redundant values, and then becoming less redundant the more we approach the top of the tree.

2.3. Decoding:

Step 1: Huffman decoding:

Through Huffman encoding, we retrieve data and the dictionary in order to recover the positions **I**, **J**, the block sizes and the mean values.

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

Step 2: Reconstructing the Image:

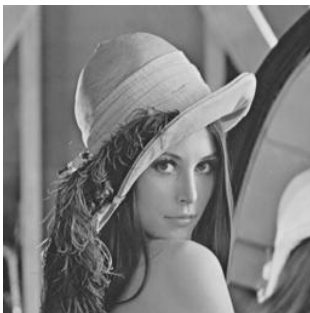
Using the positions (**I**, **J**), the block size and the mean values, we created a loop to reconstruct the image.

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

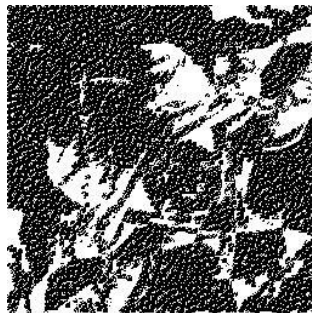
2.4. Results:

In order to display the performance of our program, we compress 2 standard test images: Lenna and Cameramen, with 256 gray levels.

Original image



Bat Decomposition



Decompressed image

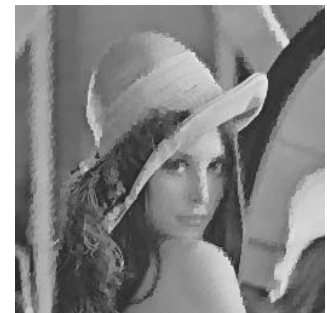


Figure 3.4: Compress Lenna with BIA.



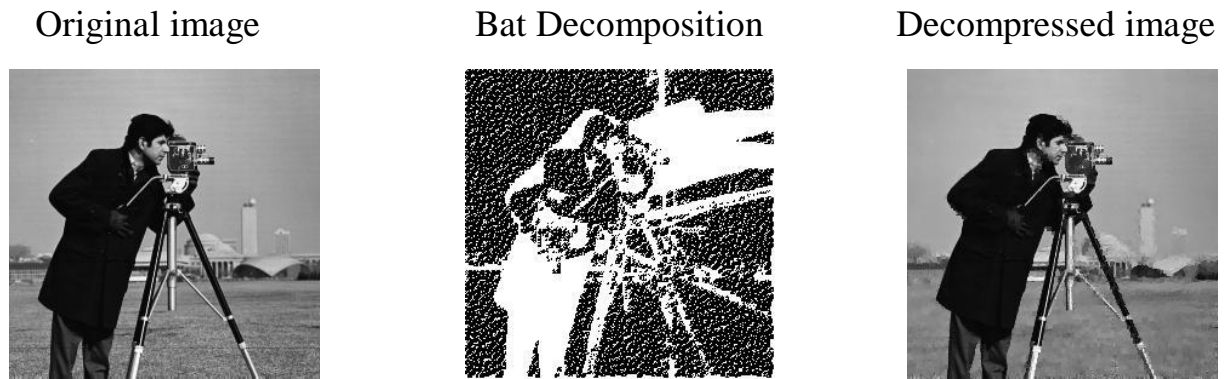


Figure 3.5: Compress Cameraman with BIA.

The processing parameter are follows: the resolution of the image: 256*256, the number of bats: 256, Loudness: 4, Frequency: 70.

Part II: Tests and results

1. Settings of the Bat Algorithm:

In this part, we will adjust our algorithm with different values of variables (number bats, Loudness, Frequency) and we will use the standard test images Cameraman and Lenna of size 32 X 32.

1.1. Number of bats:

The Images has a resolution of 32x32, loudness value 3 and frequency 40.

Image	Number of bats	Time compression	Time decompression	Compression ratio	PSNR	MSE
Camera man	2	0.488	0.705	1.385	31.608	10.088
	4	0.459	0.707	1.366	30.934	8.563
	8	0.452	0.729	1.372	31.216	9.417
	16	0.451	0.749	1.349	30.934	8.879
	32	0.509	0.843	1.355	29.827	10.045
	64	0.472	0.919	1.365	30.412	10.405
	128	0.478	0.749	1.348	31.895	9.663
	256	0.475	0.883	1.335	30.989	8.899



	512	0.457	0.750	1.392	29.997	9.870
Lenna	2	0.518	0.739	1.303	30.629	14.440
	4	0.548	0.727	1.315	30.083	15.612
	8	0.505	0.716	1.311	30.071	14.929
	16	0.514	0.713	1.306	29.984	15.348
	32	0.563	0.784	1.299	30.452	15.037
	64	0.562	0.779	1.298	31.228	13.887
	128	0.512	0.715	1.306	30.669	15.603
	256	0.518	0.709	1.320	30.389	16.118
	512	0.520	0.702	1.321	29.856	14.710

Table 3.1: results for different Number of Bats.

As we can conclude from the table above, in the **Cameraman** picture the best two values for number of bats are 4 and 8. However, 8 is better than 4 in the encoding time, compression ratio and PSNR.

In the **Lenna** picture, the best values are 8 and 512. In this case the value 8 is better than 512 in the compression time.

Considering the results of this test, the Best value is 8.

1.2. Loudness:

We took the same images with a resolution of 32x32, 8 as a number of bats, and 40 as frequency.

Image	Loudness	Time compression	Time decompression	Compression ratio	PSNR	MSE
Cameraman	2	0.453	0.720	1.267	33.788	6.531
	3	0.452	0.729	1.372	31.216	9.417
	4	0.458	0.721	1.376	33.022	7.807
	5	0.469	0.702	1.355	30.220	9.375
	6	0.438	0.727	1.359	30.119	9.027
	7	0.460	0.723	1.348	28.971	9.130



	8	0.442	0.727	1.279	34.284	5.600
	9	0.455	0.736	1.220	30.813	6.962
	10	0.468	0.757	1.205	33.859	5.012
	11	0.472	0.755	1.187	33.548	4.362
Lenna	2	0.530	0.735	1.248	31.781	14.848
	3	0.505	0.716	1.311	30.071	14.929
	4	0.537	0.726	1.324	29.230	16.853
	5	0.586	0.751	1.299	30.536	14.382
	6	0.519	0.713	1.244	30.812	13.766
	7	0.516	0.715	1.223	31.290	12.190
	8	0.503	0.756	1.228	31.032	11.917
	9	0.495	0.740	1.204	32.213	9.056
	10	0.492	0.761	1.194	31.873	8.372
	11	0.488	0.737	1.192	32.050	9.695

Table 3.2: results for different Loudness.

From this table, in the **Cameraman** picture the best two values for Loudness are 2 and 8. However, 8 is better than 2 in compression time, compression ratio, PSNR and MSE.

In the **Lenna** picture the best values are 8 and 7, but in this case 8 is better than 7 in both the compression time and MSE.

Based on this test's results, the best value is 8.

1.3. Frequency:

The Images are 32x32, Number bats is 8 and Loudness is 8.

Image	Frequency	Decomposition time	Time compression	Time decompression	Compression ratio	PSNR	MSE
Cameraman	20	0.119	0.490	0.714	1.118	40.341	1.526
	30	0.113	0.458	0.698	1.209	33.576	4.4806
	40	0.109	0.442	0.727	1.279	34.284	5.600
	50	0.140	0.460	0.738	1.357	29.463	11.990



	60	0.125	0.435	0.706	1.422	26.887	15.954
	70	0.183	0.453	0.716	1.586	23.701	22.989
	80	0.187	0.467	0.702	1.722	23.769	33.415
Lenna	20	0.091	0.495	0.755	1.066	42.199	1.904
	30	0.106	0.512	0.735	1.099	35.522	4.947
	40	0.098	0.503	0.756	1.228	31.032	11.917
	50	0.117	0.514	0.718	1.359	27.649	23.370
	60	0.160	0.549	0.745	1.590	23.577	39.059
	70	0.161	0.521	0.717	1.850	22.204	50.050
	80	0.295	0.646	0.720	2.064	20.869	58.349

Table 3.3: results for different Frequency.

As we can observe from the table, in the **Cameraman** image, the best frequency values are 30 and 40. However, the value 30 decreases the decompression time and has less MSE.

In **Lenna**, the two best values are 20 and 30, in this case the value 30 have better compression ratio than the value 20.

Depending on the results of the two images, the best frequency is 30.

1.4. The best result:

Finally, the results from these tests, the best parameters of the BIA are as follows:

- Number of bats: 8.
- Loudness: 8.
- Frequency: 30.

2. Bat with different methods of FIC:

Now we will compare our Algorithm using the best values previously mentioned with the different methods that were created to improve fractal image compression.



2.1. Standard FIC:

Image	Encoding time (sec)			Compression ratio			PSNR (db)		
	FIC		BIA	FIC		BIA	FIC		BIA
	Method 1	Method 2	/	Method 1	Method 2	/	Method 1	Method 2	/
Lenna	600	55	29.56	1	2.66	1.388	27	21	32.540

Table 3.4: comparison between standard FIC and BIA.

The tested image is Lenna with resolution 128x128. From this comparison, we notice the huge optimization in our algorithm in the case of the encoding time, compression ratio and the PSNR compared to the standard fractal image compression. In different words our algorithm has improved the standard FIC from all aspects.

2.2. Particle Swarm Optimization:



Figure 3.6: Lenna with PSO and BIA.

Image	Compression ratio		PSNR	
	PSO	BIA	PSO	BIA
Lenna	1.89	1.481	34.39	33.281
Barbra	1.89	1.450	32.98	33.599

Table 3.5: comparison between PSO and BIA.



In this test, we used the test images Lenna and Barbra with size 256x256. From the table above the PSO has better Compression ratio. In the case of PSNR, the two methods has close results.

2.3. Wolf pack algorithm:

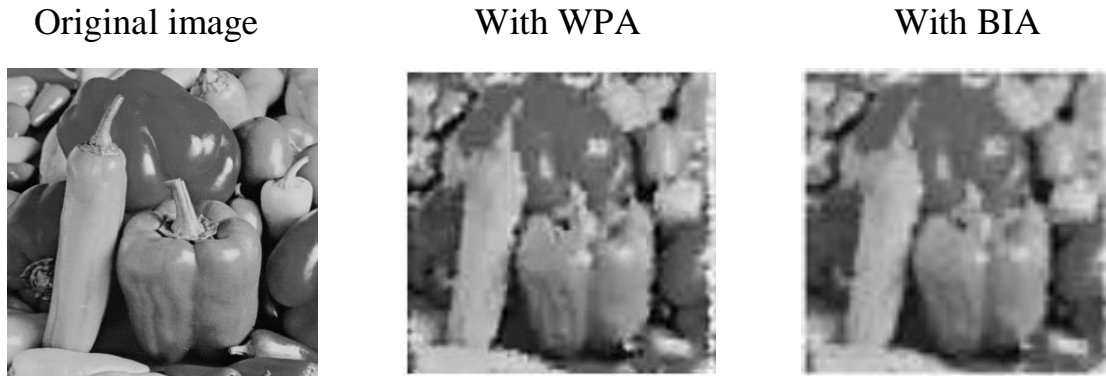


Figure 3.7: Peppers with WPA and BIA.

Image	Compression ratio		Encoding time	
	WPA	BIA	WPA	BIA
Boat	1.109	1.552	2.83	3.15
Building	1.110	1.431	1.98	2.65
Pepper	1.111	1.195	2.04	1.896

Table 3.6: comparison between WPA and BIA.

In this test, three 64x64 test images (Boat, Building, Pepper) were used. We can observe an obvious superiority on compression ratio; however, the encoding time in WPA is a little better.

2.4. Genetic Algorithm:

Image	Compression ratio			PSNR		
	GA		BIA	GA		BIA
	Single level	Two Level	/	Single level	Two Level	/
Lenna	1.277	1.117	1.350	26.16	30.22	35.580

Table 3.7: comparison between GA and BIA.



The image used in this test is 256x256 Lenna. The results shows that BIA is way better than GA in both compression ratio and PSNR.

2.5. Quad-tree Decomposition:

Image	Compression ratio		PSNR		Compression time		MSE	
	Quad-tree	BIA	Quad-tree	BIA	Quad-tree	BIA	Quad-tree	BIA
Cameraman 32x32	1.532	1.279	22.715	34.284	0.423	0.458	37.625	4.4806
Lenna 32x32	1.095	1.099	22.213	35.522	0.480	0.512	61.021	4.947
Cameraman 64x64	2.212	1.458	23.472	29.291	0.983	2.868	32.261	7.529
Lenna 64x64	1.442	1.293	23.294	33.416	0.753	2.314	47.460	9.115

Table 3.8: comparison between Quad-tree Decomposition and BIA.

As we can notice from this table, the Quad-tree has the superiority in the compression time and the compression ratio, but when it comes to the quality, the BIA surpasses the Quad-tree.

Conclusion:

In this chapter, we explained the implementation of our algorithm following it with example pictures of our results, then we made tests in order to find the best criteria for our algorithm, finally we compared the results with different optimization methods.



General conclusion

General conclusion:

The amount of information increases faster than the storage capacity. Therefore, we need to compress data during the transfer too. This field has a long life ahead of itself. New important algorithms are born every year.

Any compression attempts to eliminate redundancy, either by a different, but reversible, structuring that allows to restore the original (lossless compression), or by removing some of the information considered useless or irrelevant (lossy, irreversible methods). Irreversible methods offer a much higher compression ratio than lossless methods; Of course, sometimes it is out of the question to lose the information. One of the most known lossy methods is the fractal image compression, a method that uses self-similarity to eliminate redundancy on the image.

We have proposed a new approach for improving the fractal image compression by combining it with a metaheuristic known as bat inspired algorithm (BIA).

Our proposed approach is divided into many steps. First, the encoding process starts with decomposing the image into domain blocks. The next step is to use the Huffman encoding to store the data pixels. In the decompression process the image's data will be restored using Huffman decoding, and then the image will be reconstructed using these data. The last step is to calculate the standard quality measurements (PSNR and MSE) as well as the compression ratio.

Compared with other optimization metaheuristics, our algorithm offers better results in many aspects, mainly encoding and decoding time, size and quality.

Perspectives:

Below are some perspectives that can be drawn out of our contribution:

- Attempting to test our approach with a large number of documents.
- Increase the compression ratio; reduce the encoding and decoding time of our approach.
- Combining our proposed approach with other optimization metaheuristics.
- Apply our method on colored images.



References

References

1. Wolfram, Stephen. A New Kind of Science. Wolfram Media, 2002. p. 1069.
2. Mahmud, Salauddin. An Improved Data Compression Method for General Data. March 2012, International Journal of Scientific & Engineering Research, Vol. 3, p. 1.
3. Wei-Yi. An Introduction to Image Compression. Taipei : s.n., p. 1.
4. A, Subramanya. Image Compression Technique. Feb-March 2001, Potentials IEEE, Vol. 20, pp. 19-23.
5. Hannah, David Jeff Jackson & Sidney Joel. Comparative Analysis of image Compression Techniques. 7 –9 March 1993. pp. 513-517.
6. SCZ - Simple Compression Utilities and Library. scz-compress.sourceforge.net. [Online] November 26, 2008.
7. Aguilera, Paula. Comparison of different image compression formats. p. 3.
8. Rothlauf, F. Design of Modern Heuristics. Springer-Verlag Berlin Heidelberg : s.n., 2011.
9. Moslem Shahsavar, Amir Abbas Najafi, Seyed Taghi Akhavan Niaki. Mathematical Problems in Engineering. Vol. 2011.
10. R.J. Moraga, G.W. DePuy. Metaheuristics: A Solution Methodology for Optimization Problems. A.B. Badiru. G.E. Whitehouse Handbook of Industrial and Optimization Problems, 2006.
11. Ólafsson, S. Nelson and Henderson, Metaheuristics: Handbook on Simulation, Handbooks in Operations Research and Management Science VII. 2006. pp. 633-654.
12. Blum, Christian. Antcolony optimization: Introduction and recent trends. Barcelona, Spain, October 11, 2005, Universitat Politècnica de Catalunya, p. 1.
13. D. Henderson, S.H. Jacobson, and A.W. Johnson. The theory and practice of simulated annealing. F. Glover and G.A. Kochenberger. Boston : Kluwer Academic, 2003. pp. 287–320.
14. R. Menassel, B. Nini, T. Mekhaznia, Wolf Pack algorithm for a fractal image compression, MedPRAI-2016, Tebessa, Algeria, November 22nd-23rd 2016.
15. Veena.K, Bhuvaneshwari.P, Various Techniques of Fractal Image Compression - A Review, International Journal Of Engineering And Computer Science ISSN: 2319-7242. Volume 4 Issue 3 March 2015, Page No. 10984-10987
16. Jacquin AE. Image Coding Based on a Fractal Theory of Iterated Contractive Image Transformations. IEEE Trans Image Process. 1992. P :18–30.

17. Gaganpreet Kaur, Manjinder Kaur, Fractal Image Compression using Soft Computing, International Journal of Computer Trends and Technology (IJCTT) - volume4Issue4 –April 2013.
18. Miroslav Galabov, Fractal Image Compression, International Conference on Computer Systems and Technologies - CompSysTech'2003.
19. Veenadevi .S. V and A.G. Ananth. Fractal image compression using Quad-tree decomposition and Huffman coding, signal and image Processing: an international journal (SIPIJ) vol.3, no. 2, April 2012.
20. Suman K. Mitra, C. A. Murthy, and Malay K. Kundu. Technique for Fractal Image Compression Using Genetic Algorithm. IEEE Transactions On Image Processing, VOL. 7, NO.4, APRIL 1998.
21. Y. Chakrapani and K. Soundararajan. Implementation of fractal Image compression employing particle swarm optimization. World journal of modeling simulation, vol.6, 2010, no.1.pp 40-46
22. D. Karaboga, B. Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, Journal of Global Optimization, Vol. 39, 2007, pp. 459- 471
23. R. Sudhakar, M.R. Image compression using coding of wavelet Coefficients- a survey, ICGST_GCCIP Journal, vol.5, no.6 pp 25- 38, June 2005.
24. Xin-She Yang. A New Metaheuristic Bat-Inspired Algorithm (2010).