Faculty of Sciences and Techniques of Marrakech

Graduation Project

In partial fulfillment of the requirements for the licentiate degree in Sciences and Techniques

---

Hybridization of Divide-and-Conquer Technique And Metaheuristic Algorithm For Better Contrast Enhancement In Medical Images

---

Authored by : Ali Ait Houssa
Supervised By : Pr. Nour Eddine ALAA

Academic Year 2023-2024

Contents

Chapter 1

Introduction:

1.1 Intro Into Image Processing

Unlike our continuous visual experience, medical imaging captures the body in digital snapshots. Image processing bridges the gap, translating these images for our brains.

Image processing, where computer science meets math, analyzes digital images to reveal hidden details. Unlike photos, medical scans often have subtle variations invisible to the naked eye. Here, image enhancement steps in to make these variations clear.

Image enhancement, refines digital images. It tackles limitations like poor lighting or noise, making them more interpretable for humans and suitable for further analysis tasks like object recognition. Through techniques like contrast enhancement and noise reduction, image enhancement unlocks crucial information for various fields, from medical imaging to entertainment.

Image enhancement is vital in medical imaging because it reveals hidden details crucial for accurate diagnoses. Like X-rays or MRIs, often capture faint variations in tissue density or function. Our eyes struggle to differentiate these subtle differences. Image enhancement techniques act like a magnifying glass for these faint details, amplifying them within the image.

By boosting these variations, doctors can see critical information more clearly, leading to several benefits:

- **Earlier and more accurate diagnoses:** Enhanced details can reveal abnormalities like tumors or fractures at earlier stages, allowing for prompt interventions that can significantly improve patient outcomes.
- **Improved treatment planning:** With a clearer picture of the affected area, doctors can plan treatments like surgery or radiation therapy with greater precision and effectiveness.
- **Reduced need for additional scans:** In some cases, enhanced images might provide sufficient information, potentially reducing the need for further scans, minimizing radiation exposure for patients.

This ever-evolving technology continues to revolutionize many aspects of our lives, pushing the boundaries of what we can extract and understand from visual data. By unveiling the previously invisible details within medical images, image processing empowers doctors to deliver better diagnoses and ultimately improve patient care.

1.2 Approaches And Techniques:

This report explores the frontiers of medical image processing, showcasing its vast potential across various healthcare applications. This multifaceted field acts as a bridge, connecting disciplines like mathematics, computer science, physics, and medicine. By leveraging sophisticated image processing techniques, medical professionals can unlock the hidden secrets within raw medical data. These techniques transform the data into clear visual representations, enabling deeper analysis and the development of targeted solutions for specific medical problems.

A variety of concepts and methodologies are employed to organize the realm of medical image processing, each emphasizing distinct facets of its fundamental domains. some notable approaches include:

- **Image Formation :** The process begins with the acquisition of raw image data, which contains information about the physical quantities measured during imaging. These quantities depend on the specific imaging modality used. For example, in computed tomography (CT) or digital radiography (DR), the raw data may consist of information about the energy of incident photons. In positron emission tomography (PET), it includes data about both the energy and detection time of photons. In magnetic resonance imaging (MRI), it involves parameters of the X-ray signal emitted by excited atoms. In ultrasonography, it comprises parameters of acoustic echoes.

  After acquiring the raw data, the next step is image reconstruction. This is a mathematical process where the acquired raw data is transformed into a visual image. In multidimensional imaging, such as (CT) or (MRI), this process often involves combining data from multiple angles or time steps to create a complete image. Image reconstruction deals with inverse problems, which are mathematical challenges where you need to deduce unknowns from observed data. Analytical and iterative algorithms are commonly used to solve these problems efficiently and accurately.

- **Image Computing :** Image Computing is a field that brings together computer science, engineering, physics, math, and medicine. It focuses on developing methods to solve problems related to medical images, used in both research and healthcare. The goal is to extract useful information from these images. Image computing methods can be divided into three main areas: enhancement, analysis, and visualization. Image enhancement improves the quality of an image to make it easier to interpret. This can be done by directly adjusting the image pixels (spatial domain technique) or by using frequency transforms and filters (frequency domain technique).

  Image analysis is the core process, involving tasks like identifying patterns, detecting

edges, removing noise, counting objects, and analyzing texture or image quality.

Visualization converts image pixels/voxels into 2D/3D graphics. This is crucial for understanding complex structures in medical or industrial applications. It helps to represent anatomical and physiological information accurately in a specific format and dimension.

- **Image Data Management:** The concluding phase in medical image processing centers on the meticulous management of acquired image data. With the progression of digital imaging technologies, the intricacies of handling medical images have escalated. This phase encompasses a spectrum of techniques tailored to effectively store, communicate, transmit, archive, and retrieve image data. (Example: https://nbia.cancerimagingarchive.net)

  For instance, the storage demands of a single grayscale radiograph in its original format often necessitate compression techniques to optimize space utilization. Notably, image data management plays a pivotal role in facilitating detailed, non-invasive visualization and exploration of internal anatomy. By enabling the creation and analysis of 3D models, it significantly contributes to enhanced patient treatment, innovation in drug delivery systems, medical device development, and more informed diagnostic procedures. Moreover, noise reduction and contrast enhancement stand out as essential techniques within this phase, crucial for enhancing the quality and utility of medical images.

On the other hand, there exists a variety of techniques used for image enhancement in the medical field. Common and significant techniques used to improve the quality of medical images include:

- **Noise Reduction:** Noise reduction in medical imaging is crucial for clearer images and accurate diagnoses. Noise, random intensity fluctuations, can obscure details. Techniques like filtering, wavelet transform, and deep learning-based de-noising reduce noise while preserving important features like edges. By improving the signal-to-noise ratio, these methods help clinicians identify abnormalities more accurately, improving patient care.

- **Reducing Artifacts:** in medical imaging is vital for precise diagnosis. These artifacts, stemming from equipment malfunction, patient movement, or tissue characteristics, can distort anatomical structures, potentially causing misdiagnosis or erroneous treatment choices.

- **Histogram Equalization:** is a contrast enhancement technique used in image processing. It adjusts the distribution of pixel intensities in an image to make it more uniform, thus improving visibility of features. It's particularly beneficial for images with uneven lighting or low contrast. However, it may also increase noise. Overall, histogram equalization is valuable for enhancing image quality and aiding in analysis.

- **The Retinex Algorithm:** enhances image contrast and color balance by separating an image into its illumination and reflectance components. By adjusting these components independently and recombining them, Retinex compensates for uneven lighting and improves overall image quality, particularly in low-contrast or challenging lighting conditions. It's a versatile tool for various image enhancement tasks.

- **Convolutional Neural Networks (CNNs):** are instrumental in medical image enhancement by learning complex mappings from low-quality to high-quality medical images. They effectively remove noise, enhance contrast, and improve overall image quality. Specifically designed architectures like U-Net are commonly used, enabling preservation of spatial information while enhancing features. (CNNs) are versatile for tasks like de-noising, de-blurring, and contrast enhancement, contributing to clearer and diagnostically valuable medical images for better patient care.

- **Divide And Conquer:** In medical image enhancement, combining Convolutional Neural Networks (CNNs) or Meta-heuristic methods (which we are going to tackle during this project) with the "divide and conquer" approach involves breaking the enhancement task into smaller subtasks. By dividing the image into patches, (CNNs) can focus on localized enhancement, improving efficiency and preserving details. This method enables more precise enhancement, especially for tasks like de-noising or contrast enhancement, while maintaining computational efficiency.

1.3 The Goal in this report:

In this report, we propose the utilization of the Divide And Conquer Algorithm in conjunction with several metaheuristic algorithms to enhance selected samples of medical images. Our objective is to improve the quality, clarity and exposing hidden details of medical images, thereby aiding medical professionals in accurate diagnosis and treatment planning.

The Divide And Conquer Algorithm serves as the foundation of our approach, allowing us to efficiently decompose the image enhancement problem into smaller or rather frequencies, more manageable sub-problems. By dividing the image into smaller regions, we can apply specialized enhancement techniques tailored to each region's characteristics. This facilitates targeted and localized improvements while minimizing computational complexity.

In addition to the Divide And Conquer Algorithm, we integrate several Meta-Heuristic algorithms into our framework to further optimize the image enhancement process. These Meta-Heuristic algorithms, inspired by natural phenomena (Particle Swarm Optimization) or

mathematical principles (Sine Cosine algorithm), offer innovative strategies for exploring and exploiting the solution space to achieve superior enhancement results.

The Divide And Conquer Algorithm provides a structured framework for decomposition and local enhancement, while Meta-Heuristic algorithms offer powerful optimization capabilities to fine-tune enhancement parameters and adaptively adjust to image characteristics depending on desired results.

Chapter 2

Divide And Conquer Method

2.1 Images And Convolution Product:

2.1.1 Image structure

Gray images in computer science are represented as a matrix of pixel values ranging from 0 to 255, with each holding a number related to the intensity of the shade of gray in that pixel, a black pixel is represented by 0, while a white one is represented with 255. The matrix holding those values is of dimensions $N \times M$ with N being the image width and M its height (see figure 2.1 below).

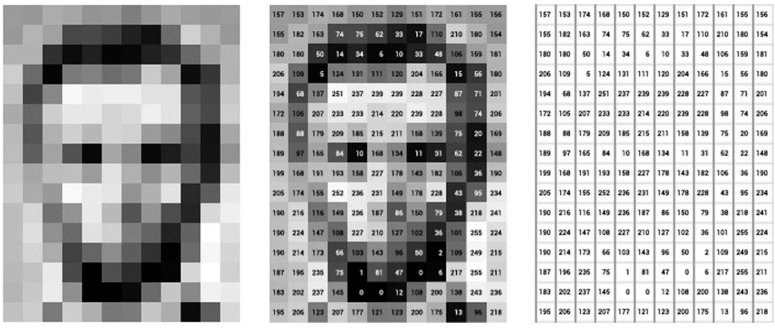

Figure 2.1: An image with its color values over it.

For colored images it's another story. It's basically an $N \times M \times K$ matrix with $K = 3$ generally being the depth or channels of the image (as shown in the figure 2.2). It is like a cube matrix such that every channel represents red, green, and blue (RGB) shades/values, each ranging from 0 to 255 as well. So by combining those 3 colors, we are capable of producing a full-color image that can be displayed on screens or printers.

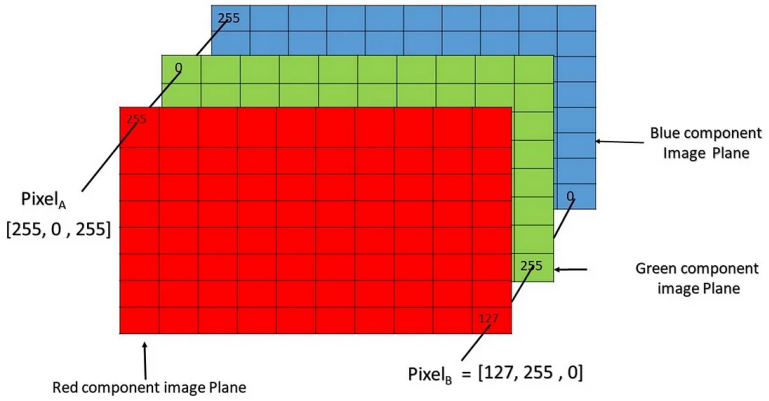

Figure 2.2: Example of RGB images representation.

2.1.2 Convolution Product And Kernels

Kernels: A kernel (also known as a convolution kernel, filter, or mask) is a small square matrix usually of size 3 x 3, 5 x 5 or 9 x 9 containing different patterns of numbers to produce different results under convolution product. They can scale more depending on requirements. these kernels are typically applied to an image or signal through convolution to perform operations such as blurring, sharpening, edge detection, and more, that we call filters as in filtering images. Kernels play a crucial role in various image processing tasks and are fundamental components of convolutional neural networks (CNNs).

Examples of kernels used in image convolution are *Gaussian kernel* or *Box Blur* that is used to compute the weighted average of the neighboring points (pixels) in an image which results in blurring the image. For Edge detection kernels they are used to highlight edges in the image. They emphasize rapid changes in intensity. One popular edge detection kernel is the Sobel kernel, which calculates the gradient of the image intensity at each pixel. Here are some examples of their numerical matrices:

Gaussian Blur Kernel

$$\begin{bmatrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{bmatrix}$$

Horizontal Edge Detection Kernel

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Identity Kernel

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Vertical Edge Detection Kernel

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Convolutional product, often referred to simply as convolution, is a mathematical operation that combines two functions to produce a third function that represents the amount of overlap between the two original functions as one is shifted over the other.

In the context of image processing, the convolution operation involves sliding a filter kernel (also known as a convolution kernel or mask) over the image matrix and computing the sum of the element-wise products between the filter and the overlapping portion of the image. Technically a convolution is done by multiplying a pixel's and its neighboring pixels color value by a matrix. This process is repeated for every pixel in the image, resulting in a new image that has been filtered or processed based on the characteristics of the convolution kernel.

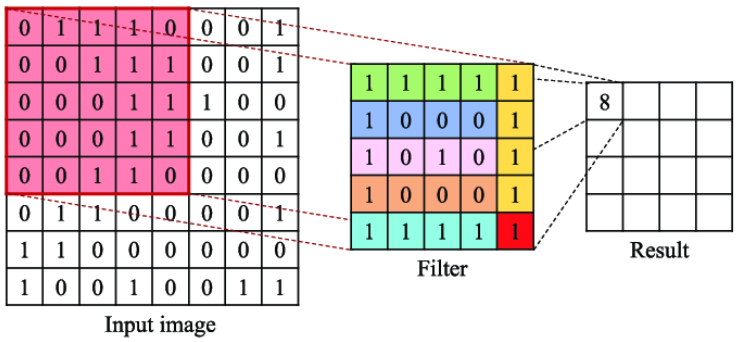

Figure 2.3: Illustration of images Convolutional Product.

If we take a look at the Gaussian blur matrix per example as shown in the figure below, after convoling it with the image, it looks as if its averaging the current pixel with its neighbouring pixels, and that's for every pixel in the image. Its more of like colors being bleeding out into their neighboring pixels.

And that is only the Gaussian kernel. There are many more with different effects. visual

examples are shown below.

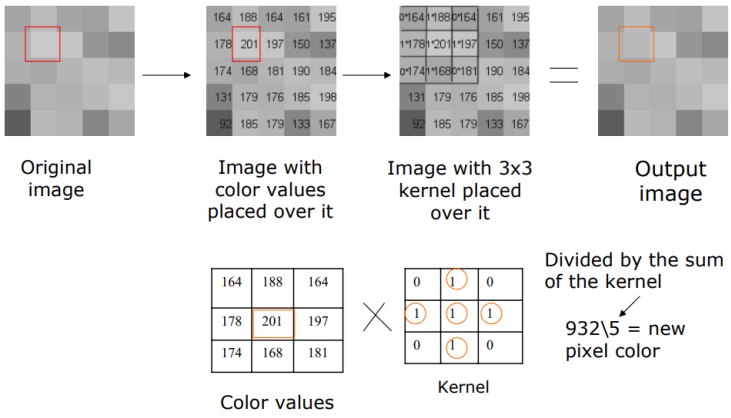

Figure 2.4: Illustration of images Convolutional Product.

2.1.3 Visual Examples

- **Sobel kernel:**

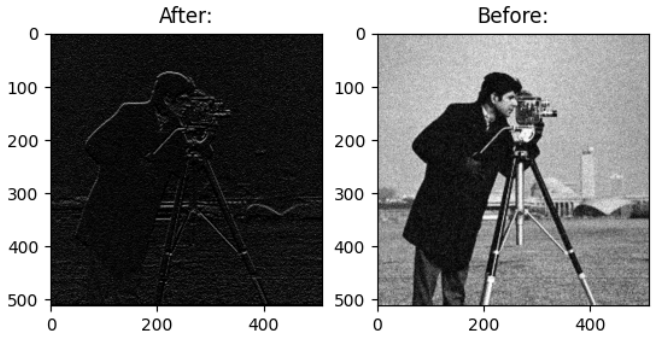

Figure 2.5: Before and after applying the Sobel kernel.

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- **Gaussian kernel (filter):**

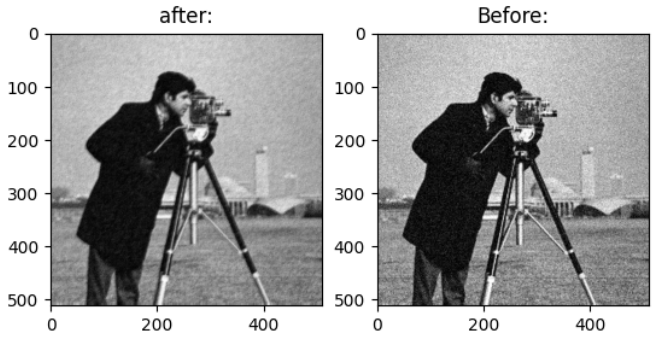

Figure 2.6: Before and after applying the Gaussian kernel (blur).

$$\begin{bmatrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{bmatrix}$$

- **Emboss kernel (filter):**

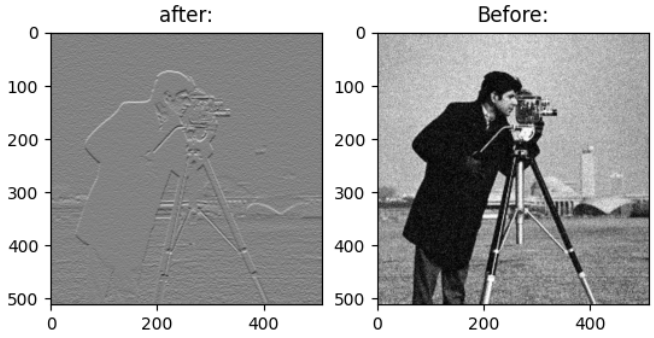

Figure 2.7: Before and after applying the Emboss kernel.

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

2.2 Introduction To Divide And Conquer Method

In computer science, the concept of divide and conquer serves as an algorithmic design paradigm. With this approach, a divide-and-conquer algorithm systematically breaks down a complex problem into smaller, more manageable sub-problems through recursive partitioning. This recursive process continues until the sub-problems are sufficiently simplified to be solved directly, often by employing base cases or straightforward methods.

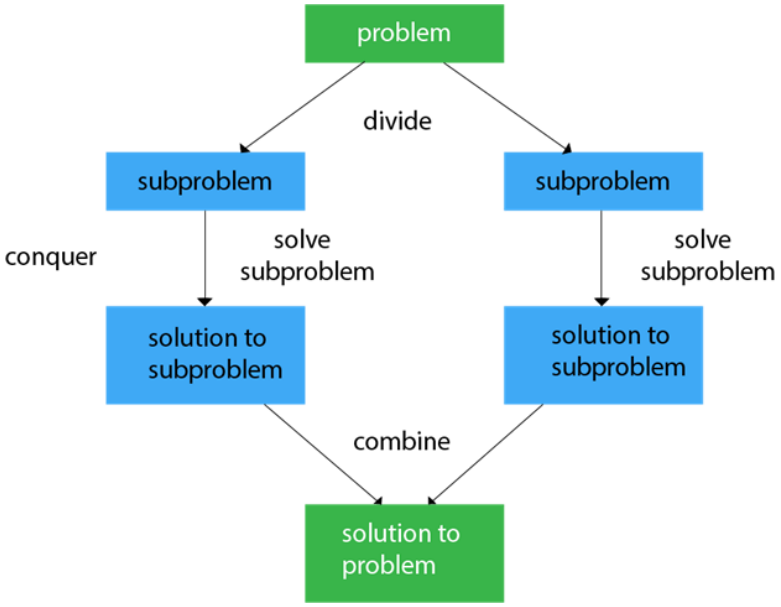

Figure 2.8: Divide And Conquer Diagram

Once each sub-problem is solved, the solutions are intelligently combined to derive a comprehensive solution to the original problem. This powerful strategy enables efficient problem-solving by breaking down intricate challenges into smaller, more digestible components, facilitating streamlined algorithmic solutions across various computational domains.

The divide-and-conquer method forms the foundation of efficient algorithms across a spectrum of problems. These include sorting tasks (like Quicksort and merge sort), large number multiplication (such as the Karatsuba algorithm), closest pair of points determination, syntactic analysis (for instance, top-down parsers), and discrete Fourier transform computation (FFT).

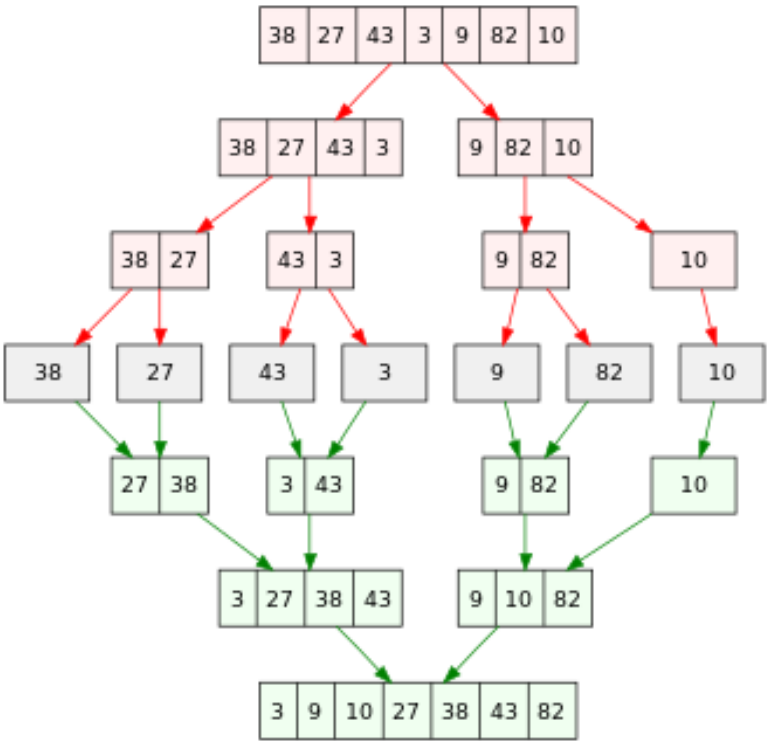

Figure 2.9: Illustration Of Merge Sort Algorithm that uses Divide And Conquer method

Divide-and-conquer algorithms are naturally implemented as recursive procedures. In that case, the partial sub-problems leading to the one currently being solved are automatically stored in the procedure call stack. As for a recursive function. It is a function that calls itself within its definition. (for simplicity see Quick sort algorithm as an example)

2.3 Divide And Conquer In Our Case

2.3.1 Decomposition Of The Image

Our approach utilizes the Divide and Conquer strategy for image processing. We begin with a grayscale image, represented as a 2D matrix. This matrix will be divided into four separate frequency components using specific kernels. Essentially, we're decomposing the image into its constituent frequency bands.

This process results in four separate images, each capturing a distinct frequency band of the original image. To achieve this separation, we will perform convolution with four specific kernels:

$$h_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{bmatrix} \quad h_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix} \quad h_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad h_4 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.1}$$

Then we perform two dimensional convolution product between the image and every kernel h_i

where h_3 and h_4 are two high-frequency kernels. And h_1 and h_2 are calculated using h_1 and $h_2$using the following equations:

$$h_1 = I - H_3 \quad and \quad h_2 = I - H_4 \tag{2.2}$$

Where H_3 and H_4 are respectively the Fourier transform of h3 and h4. I is a matrix where each of its elements is equal to one. Once the decomposition process of the image U is obtained (U_1, U_2, U_3, U_4), the sub-images U3 and U4 contain the high-frequency component of the entire image U whereas, U1 and U2 form regions with low-frequency. Then, we apply the FitzHugh-Nagumo model, that is initially developed to describe excitable systems such as nerve cells, which has been adapted for use in image processing. In this context, the model is typically employed for edge detection and image segmentation.

2.3.2 FitzHugh-Nagumo

The FitzHugh-Nagumo model is a simplification of the Hodgkin-Huxley model of neuronal action potentials. It consists of two differential equations describing the dynamics of an excitable medium.

In image processing, the FitzHugh-Nagumo equations can be adapted to enhance the detection of edges and features within an image. And are expressed in the following equations:

$$U(t + dt) = U(t) + \frac{dt}{\tau}(U(t)(U(t) - a)(1 - U(t)) - v(t)) \tag{2.3}$$

$$V(t + dt) = V(t) + dt(U(t) - bV(t)) \tag{2.4}$$

1. **Initialization:** The image is treated as an initial condition for the U variable, representing the pixel intensities.

2. **Dynamics:** The model evolves over time according to the FitzHugh-Nagumo equations.

3. **Edge Detection:** The evolution highlights areas of high intensity change, corresponding to edges. The U variable evolves in such a way that it enhances these edges.

4. **Segmentation:** By analyzing the dynamics of U and V different regions within the image can be distinguished, aiding in segmentation.

2.3.3 Enhancement using Effective Measure of Enhancement (EME) Function

In the realm of image processing, various techniques are employed to enhance and analyze digital images. One crucial aspect is the manipulation of sub-images or regions within a larger image to improve overall image quality or to extract specific features. The goal is often to balance different characteristics, such as contrast, sharpness, or noise levels, across the entire image. This can involve sophisticated algorithms that apply different weights to these sub-images, ensuring that the enhancements contribute positively to the final image.

Among the many functions used in these algorithms, the Effective Measure of Enhancement (EME) function stands out. EME is a vital tool for emphasizing the edges within an image, making it easier to identify transitions and boundaries between different regions. By adjusting the intensity of edges, EME helps in highlighting the important details that might otherwise be lost, particularly in images with low contrast or high noise levels. This function, along with others like it, plays a significant role in the pre-processing steps of image analysis, paving the way for more accurate and effective image segmentation, recognition, and interpretation.

Effective Measure of Enhancement (EME) is a function used in image processing to emphasize the edges within an image. Edges are critical in identifying transitions and boundaries between different regions of an image. Enhancing these edges helps in various tasks, such as object recognition, segmentation, and feature extraction, by making the important details more prominent.

The EME function is designed to increase the contrast around the edges, making them more distinct. The general formula for EME is:

$$EME(U, M, N) = \frac{1}{M \times N} \sum_{i=0}^{M} \sum_{j=0}^{N} \log(\frac{I_{maxij}}{I_{I_minij}}) \tag{2.5}$$

Where M and N are the dimensions of the image divided into non-overlapping blocks.I_{maxij} and I_{minij} are respectively the maximum minimum pixel intensity value in the $(i, j)^th$ block. and finally the log is the logarithm function, which helps in normalizing the contrast enhancement across different scales.

Steps in EME Calculation:

1. **Image Division:** Divide the image into non-overlapping blocks of size $M \times N$.

2. **Intensity Calculation** For each block, calculate the maximum and minimum pixel intensity values.

3. **Logarithmic Ratio:** Compute the logarithmic ratio of the maximum to the minimum intensity for each block.

4. **Summation:** Sum these logarithmic values for all blocks

5. **Normalization:** Normalize the sum by the total number of blocks to obtain the EME value.

2.3.4 EME and Metaheuristics:

Now back to our enhanced subspace image U which is reconstructed from a linear remapping approximation:

$$U_e = \sum_{i=0}^{4} w_i U_i \tag{2.6}$$

Where $w_{i_{i=1}}^{4}$ are the weights for balancing different sub-images are applied as part of the following algorithm, which generates the new image. And here where the Metaheuristics comes in.

The goal is to maximize the Edge Magnitude Enhancement (EME) function, which serves as our objective measure for edge contrast. To achieve this, we will employ metaheuristic algorithms, which are advanced optimization techniques suitable for solving complex problems with many local optima. Metaheuristics like Genetic Algorithms, Particle Swarm Optimization, or Sine Cosine algorithms are well-suited for this task because they can efficiently navigate the large search space of possible weight combinations to find the optimal set that maximizes the EME value. And finally we are going to compare the effectiveness of each of them heuristics.

Chapter 3

Metaheuristic Techniques

3.1 Introduction

3.1.1 Definition

Optimization problems are everywhere in our world, from scheduling deliveries to designing efficient circuits. Many optimization problems of practical as well as theoretical importance consist of the search for a "best" configuration of a set of variables to achieve some goals. This is where Metaheuristics comes in.

In computer science and mathematical optimization, a metaheuristic is a higher-level procedure or heuristic designed to find, generate, tune, or select a heuristic (partial search algorithm) that may provide a sufficiently good solution to an optimization problem.

They are a powerful class of algorithms designed to solve complex optimization problems that are often too difficult for traditional methods. They seem to divide naturally into two categories: those where solutions are encoded with real-value variables, and those where solutions are encoded with discrete variables. These problems, prevalent across various domains such as engineering, logistics, finance, and machine learning, frequently exhibit features such as non-linearity, high-dimensionality, and the presence of multiple local optima.

Traditional optimization techniques, which require specific mathematical properties like differentiability and convexity, often fall short in these scenarios due to their reliance on exact algorithms that can be computationally prohibitive for large-scale or highly complex problems.

So in short, Metaheuristics are methods used to find local or global minimums or maximums of a certain function with some certain parameters.

3.1.2 Exploration And Exploitation

The strength of metaheuristics lies in their ability to balance exploration and exploitation of the search space.

- **Exploration:** Exploration refers to the process of broadly searching the solution space to discover a wide variety of potential solutions. This aspect of the search strategy is designed to investigate different regions of the search space without bias towards any particular area.

  The goal of exploration is to ensure that the algorithm does not overlook any potentially optimal solutions by getting stuck in local optima—suboptimal solutions that appear to be the best within a limited region but are not the best overall. Effective exploration helps to avoid premature convergence on these local optima by introducing diversity into the search process.

  Various techniques, such as randomization, mutation, or the use of diverse initial solutions, are employed to achieve robust exploration. By thoroughly exploring the solution space, metaheuristics increase the likelihood of finding the global optimum or a high-quality near-optimal solution, making them well-suited for complex and large-scale optimization problems.

- **Exploitation:** Exploitation refers to the process of intensively searching around promising solutions that have been identified during the exploration phase.

  This aspect of the search strategy focuses on refining and improving these solutions to find the best possible outcome within their vicinity. The goal of exploitation is to capitalize on the knowledge gained from the exploration phase by honing in on areas of the search space that show potential for optimal or near-optimal solutions.

  Techniques such as local search, gradient descent, or crossover operations in genetic algorithms are often used to perform exploitation. By concentrating the search efforts on these promising regions, metaheuristics can effectively enhance the quality of solutions and ensure that they are not missing out on fine-tuning already good solutions.

  The balance between exploitation and exploration is crucial, as excessive exploitation can lead to premature convergence on suboptimal solutions, while insufficient exploitation can result in missing out on the optimal solutions. Thus, effective exploitation ensures that the algorithm makes the most of the discovered promising areas, contributing significantly to the overall efficiency and success of metaheuristic optimization.

3.1.3 Metaheuristic Classification:

Local Search vs. Global Search

There are a wide variety of metaheuristics and a number of properties with respect to which to classify them. some of the classifications are:

- **Local Search:** Local search strategies focus on improving simple algorithms, such as the hill climbing method, which seeks local optima but does not guarantee finding global optima. To enhance local search algorithms and discover better solutions, several meta-

  heuristic techniques have been developed. These include simulated annealing, which allows occasional moves to worse solutions to escape local optima; tabu search, which uses memory structures to avoid revisiting previously explored solutions; iterated local search, which repeatedly applies local search to modified versions of the current solution; variable neighborhood search, which systematically changes the neighborhood structure within the search; and GRASP (Greedy Randomized Adaptive Search Procedure), which combines greedy heuristics with randomized sampling to explore local regions effectively.

- **Global Search:** In contrast, global search metaheuristics typically involve population-based strategies that explore the solution space more broadly. Examples of these include

  ant colony optimization, which uses the foraging behavior of ants to explore the solution space; genetic algorithms, which employ principles of natural selection and genetics to evolve solutions over generations; evolution strategies, which use mechanisms similar to genetic algorithms but focus on the evolution of strategy parameters; particle swarm optimization, which simulates the social behavior of birds or fish to find optimal solutions; the rider optimization algorithm, which mimics the rider behavior of herd animals to explore the search space; and the bacterial foraging algorithm, which models the foraging behavior of bacteria to locate global optima.

Single Solution vs. Population Based Solution

Another classification dimension is the distinction between single solution and population-based searches. Single solution approaches focus on modifying and improving a single candidate so-

lution. Examples of single solution metaheuristics include simulated annealing, iterated local search, variable neighborhood search, and guided local search. In contrast, population-based

approaches maintain and improve multiple candidate solutions simultaneously, often using the characteristics of the population to guide the search. Examples of population-based metaheuris-

tics include evolutionary computation and particle swarm optimization. Additionally, there is a category of metaheuristics known as swarm intelligence, which involves the collective behavior of decentralized, self-organized agents in a population or swarm. Ant colony optimization, particle swarm optimization, social cognitive optimization, and the bacterial foraging algorithm are examples of swarm intelligence metaheuristics.

Hybridization and memetic algorithms

A hybrid metaheuristic blends a metaheuristic with alternative optimization methods, incorporating algorithms from mathematical programming, constraint programming, and machine learning. In a hybrid metaheuristic, both components can operate simultaneously and communicate information to steer the search process. In contrast, Memetic algorithms epitomize the

fusion of evolutionary or population-based approaches with distinct individual learning or local improvement procedures for problem exploration. An example of a memetic algorithm involves integrating a local search algorithm alongside or instead of a basic mutation operator within evolutionary algorithms.

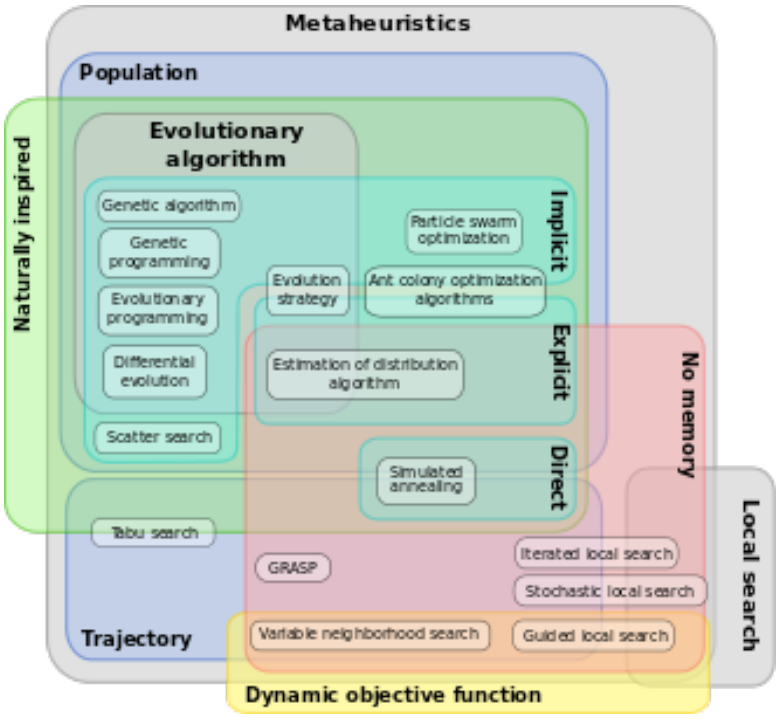

Figure 3.1: Euler diagram of the different classifications of metaheuristics.

3.2 Chosen Algorithms For Our Cause

For our cause we chose these four algorithms:

1. **Flower Pollination Algorithm (FPA)**
2. **Gravitational Search Algorithm (GSA)**
3. **Sine Cosine Algorithm**
4. **Particle Swarm Optimization (PSO)**

to find the minimum of the EME function. So for demonstration purposes, we will attempt to find the minimum of the following function:

$f(x) = x_1^2 + x_2^2$

Which trivially is the vector (0, 0). Using all four algorithms.

3.2.1 Flower Pollination Algorithm

Introduction

The Flower Pollination Algorithm (FPA) is a nature-inspired optimization technique developed to solve complex optimization problems by mimicking the natural pollination process of flowering plants. Introduced by Dr. Xin-She Yang in 2012, FPA leverages the biological principles of

flower pollination, which involves the transfer of pollen from one flower to another, facilitating plant reproduction. This algorithm capitalizes on the two primary modes of pollination: biotic,

where pollinators such as bees and birds transfer pollen, and abiotic, where wind or water aids the process. By integrating these modes, FPA effectively balances local and global search capabilities, enabling it to explore and exploit the solution space efficiently. The simplicity

and effectiveness of the Flower Pollination Algorithm have led to its widespread application in various fields, including engineering, finance, and artificial intelligence, making it a valuable tool for tackling a broad range of optimization challenges. For simplicity, the following four rules are used.

1. **Biotic cross-pollination (Global pollination):** It can be considered as a process of global pollination, and pollencarrying pollinators move in a way that obeys Lévy flights (Rule 1)

2. **Local pollination:** The algorithm employs abiotic pollination and self-pollination (Rule 2).

3. **Flower Constancy:** Pollinators such as insects can develop flower constancy, which is equivalent to a reproduction probability that is proportional to the similarity of two flowers involved (Rule 3).

4. **Switch Interaction:** The interaction or switching between local pollination and global pollination can be controlled by a switch probability $p \in [0, 1]$, which is slightly biased towards local pollination (Rule 4).

Global pollination

To formulate the updating formulas, the aforementioned rules must be translated into appropriate updating equations. For instance, during the global pollination step, flower pollen gametes are transported by pollinators such as insects, allowing pollen to travel over long distances as these insects can fly and cover extensive ranges. Consequently, Rule 1 and flower constancy (Rule 3) can be mathematically represented as follows:

$$x_i^{t+1} = x_i^t + \gamma L(\lambda)(g^* - x_i^t),$$

where x_i^t is the pollen i or solution vector x_i at iteration t, and g^* is the current best solution found among all solutions at the current generation/iteration. Here, γ is a scaling factor to control the step size.

Here $L(\lambda)$ is the parameter, more specifically the Lévy-flights-based step size, that corresponds to the strength of the pollination. Since insects may move over a long distance with various distance steps, a Lévy flight can be used to mimic this characteristic efficiently. That is, $L > 0$ is drawn from a Lévy distribution:

$$L \sim \lambda \frac{\Gamma(\lambda)\sin(\pi\lambda/2)}{\pi} \frac{1}{s^{1+\lambda}}, \quad (s > s_0 > 0).$$

Here, $\Gamma(\lambda)$ is the standard gamma function, which is an extension of the factorial function into the complex numbers. And this distribution is valid for large steps $s > 0$. In theory, it is required that $|s_0| \approx 0$, but in practice, s_0 can be as small as 0.1. However, generating pseudo-random step sizes that correctly obey this Lévy distribution is not trivial. There are a few methods for drawing such random numbers, Such as the Mantegna method.

Figure 3.2: A series of 50 consecutive steps of Lévy flights.

Local Pollination

For local pollination, both Rules 2 and 3 can be represented as:

$$x_i^{t+1} = x_i^t + \epsilon(x_j^t - x_k^t)$$

where x_j^t and x_k^t are pollen from different flowers of the same plant species. This essentially mimics flower constancy in a limited neighbourhood. Mathematically, if x_j^t and x_k^t come from the same species or are selected from the same population, this equivalently becomes a local random walk if ϵ is drawn from a uniform distribution in $[0, 1]$.

In principle, flower pollination can occur at various scales, both local and global. However, in reality, nearby flower patches or flowers in close proximity are more likely to be pollinated by local pollen than those farther away. To emulate this behavior, a switch probability (Rule 4) or proximity probability p can be effectively used to alternate between global pollination and intensive local pollination. Initially, a simple value of $p = 0.5$ can be used. Preliminary parametric studies have shown that $p = 0.8$ tends to perform better for most applications.

FPA Pseudo-Code

The following is the pseudo code for the implementation of the Flower Pollination Algorithm:

Flower Pollination Algorithm (or simply Flower Algorithm)

Objective min *or* max $f(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)$
Initialize a population of n flowers/pollen gametes with random solutions
Find the best solution $\boldsymbol{g}_*$ *in the initial population*
Define a switch probability $p \in [0, 1]$
while *(t <MaxGeneration)*
  for $i = 1 : n$ *(all n flowers in the population)*
    if *rand* $< p$,
      Draw a (d-dimensional) step vector L which obeys a Lévy distribution
      Global pollination via $\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \gamma L(\boldsymbol{g}_* - \boldsymbol{x}_i^t)$
    else
      Draw ϵ *from a uniform distribution in* $[0, 1]$
      Do local pollination via $\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \epsilon(\boldsymbol{x}_j^t - \boldsymbol{x}_k^t)$
    end if
    Evaluate new solutions
    If new solutions are better, update them in the population
  end for
    Find the current best solution $\boldsymbol{g}_*$
end while
Output the best solution found

Figure 3.3: Pseudo code of the proposed flower pollination algorithm (FPA).

3.2.2 Gravitational Search Algorithm

Introduction

The Gravitational Search Algorithm (GSA) is a heuristic optimization algorithm that has been attracting significant interest in the scientific community. Inspired by Newton's law of gravity

and the laws of motion, GSA falls under the category of population-based approaches and is considered to be particularly intuitive. The algorithm aims to enhance the exploration and exploitation capabilities of population-based methods by leveraging principles derived from gravitational laws. However, GSA has faced criticism for not strictly adhering to the true laws

of gravity, specifically for excluding the distance between masses in its calculations, which is a fundamental aspect of gravitational theory. Despite this critique, GSA continues to be actively researched and utilized within the scientific community.

This population-based heuristic algorithm is founded on the principles of gravity and mass interactions. It consists of a collection of searcher agents that interact with each other through gravitational forces. In this context, the agents are considered as objects, and their performance is measured by their masses. The gravitational force induces a global movement, causing all objects to move towards those with heavier masses. This slow movement of heavier masses ensures the exploitation step of the algorithm, leading to better solutions. The agents' movements adhere to the law of gravity, as illustrated in Equation (1), and the law of motion, as shown in Equation (2).

$$F_{ij} = G\frac{m_i m_j}{(R_{ij} + \epsilon)^a} \tag{3.1}$$

$$a_i = \frac{F_i}{m_i} \tag{3.2}$$

Based on Equation (1), F represents the magnitude of the gravitational force, G is the gravitational constant, M_1 and M_2 are the masses of the first and second objects, some variations use 3 masses: Active, Passive and Inertia just like in relativity. But for the sake of simplicity we are going to consider them the same. Respectively, R is the distance between the two objects,

and ϵ is a small number (around .01) to avoid 0 in denominator if two planets or agents are in the same position or rather n-norm distance is null. Equation (1) illustrates Newton's law of gravity, which states that the gravitational force between two objects is directly proportional to the product of their masses and inversely proportional to the square of the distance between them. In contrast, Equation (2) represents Newton's second law of motion, which states that when a force F is applied to an object, its acceleration a is determined by the force and its mass M.

GSA steps and parameters

In the Gravitational Search Algorithm (GSA), each agent is characterized by two parameters: position, gravitational mass(3 masses but for simplicity we're considering only one). The position of each mass represents a potential solution to the problem, while the gravitational mass is determined using a fitness/cost function. The algorithm navigates the search space by adjusting the gravitational mass . Agents are attracted to the heaviest mass, which represents the optimal solution in the search space. The steps of GSA are as follows:

1. **Agents initialization:** The positions of the N number of agents are initialized randomly.

$$X_i = (x_i^1 ..., x_i^d ..., x_i^n), \quad for \quad i = 1, 2, 3..., N. \tag{3.3}$$

   x_i^d represents the positions of the i^{th} agent in the d^{th} dimension, while n is the space dimension.

2. **Fitness evolution and best fitness computation:** For minimization or maximization problems, the fitness evaluation is performed by determining the best and worst fitness for all agents at each iteration: Minimization problems:

$$\text{best}(t) = \min_{j \in \{1,...,N\}} \text{fit}_j(t) \quad (4)$$

$$\text{worst}(t) = \max_{j \in \{1,...,N\}} \text{fit}_j(t) \quad (5)$$

Maximization problems:

$$\text{best}(t) = \max_{j\in\{1,\ldots,N\}} \text{fit}_j(t) \quad (6)$$

$$\text{worst}(t) = \min_{j\in\{1,\ldots,N\}} \text{fit}_j(t) \quad (7)$$

Here, $\text{fit}_j(t)$ represents the fitness value of the jth agent at iteration t, while best(t) and worst(t) represent the best and worst fitness values at iteration t, respectively.

3. **Gravitational constant (G) computation:** Gravitational constant G is computed at iteration t using the following formula:

$$G(t) = G_0 e^{(-\alpha \frac{t}{T})}$$

G_0 and α are initialized at the beginning and will be reduced with time to control the search accuracy. T is the total number of iterations.

4. **Calculation of agents masses:** Gravitational mass for each agent is calculated at iteration t is calculated using the formula:

$$m_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)} \quad (3.4)$$

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)} \quad (3.5)$$

5. **Calculation of agent's acceleration:** Acceleration of the i^{th} agents at iteration t is computed as it follows:

$$a_i^d(t) = \frac{F_i^d(t)}{M_i(t)} \quad (3.6)$$

$F_i^d(t)$ is the total force acting on i^{th} agent calculated as:

$$F_i^d(t) = \sum_{j=1, j\neq i}^{N} Rand_j F_{ij}^d(t) \quad (3.7)$$

Rand is a random number between 0 and 1.$F_{ij}^d(t)$ is computed as the following equation:

$$F_{ij}^d(t) = \frac{G(t).(M_i(t)M_j(t)).(x_j^d(t) - x_i^d(t))}{R_{ij}(t) + \epsilon} \quad (3.8)$$

$F_{ij}^d(t)$ is the force acting on agent i from agent j at d^{th} dimension and t^{th} iteration.$R_{ij}(t)$ is the Euclidian distance between two agents i and j at iteration t. G(t) is the computed gravitational constant at the same iteration while ϵ is a small constant.

6. **Velocity and positions of agents:** Velocity and the position of the agents at next iteration (t+1) are computed based on the following equations:

$$v_i^d(t+1) = Rand(v_i^d(t)) + a_i^d(t) \tag{3.9}$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1) \tag{3.10}$$

7. **Repeat steps 2 to 6:** Steps 2 to 6 are repeated until the maximum number of iterations is reached. The best fitness value at the final iteration is considered the global fitness, while the position of the corresponding agent in the specified dimensions is regarded as the global solution to the problem. The figure below illustrates the flowchart of the Gravitational Search Algorithm (GSA).

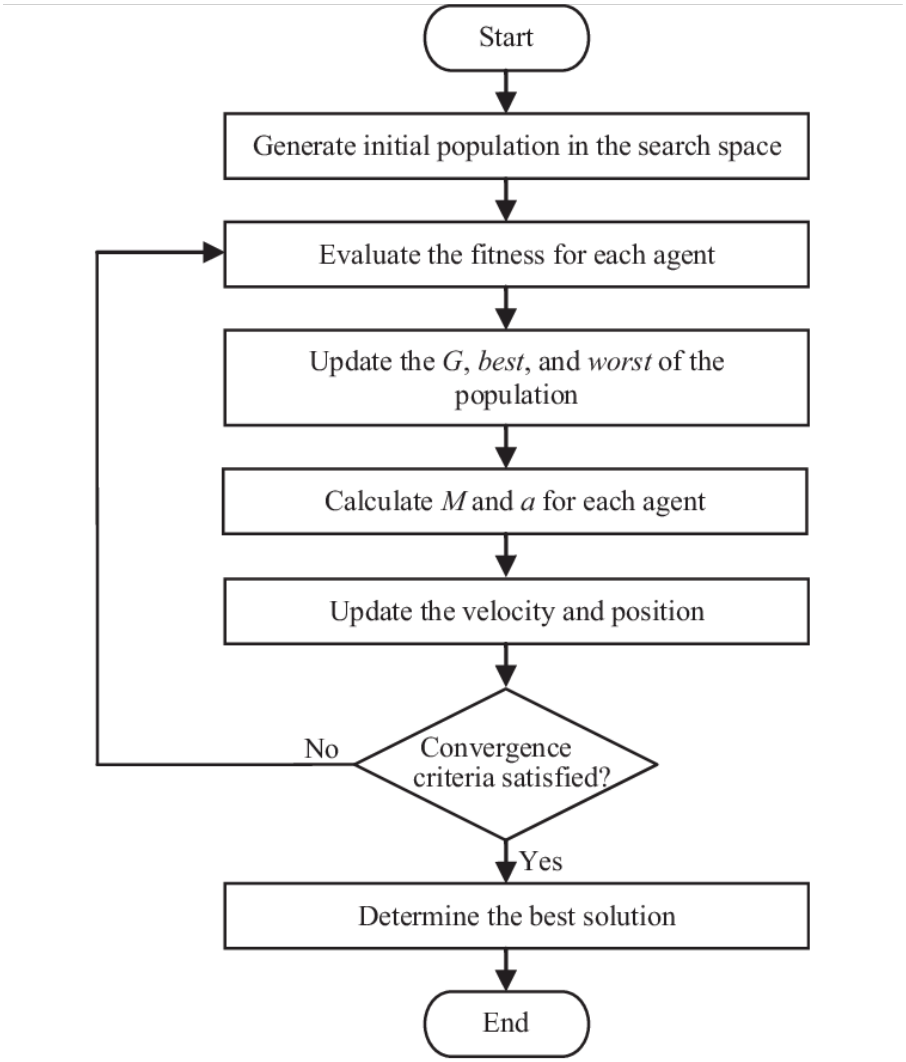

Figure 3.4: Flowchart of the GSA.

3.2.3 Sine cosine algorithm:

Introduction

The Sine Cosine Algorithm (SCA) is a modern metaheuristic optimization technique introduced to address complex optimization problems. It falls under the category of population-based probabilistic search methods and operates by updating the positions of search agents using the trigonometric sine and cosine functions. Inspired by the periodic nature of these functions, SCA leverages their inherent properties to effectively explore and exploit the search space. The

sine and cosine functions, which oscillate within the range of [1,1], enable the algorithm to balance the global exploration of the search space and the local exploitation of promising regions. This dual capability is crucial for navigating complex and multi-dimensional optimization landscapes. By systematically adjusting the positions of agents, SCA seeks to find optimal solutions through iterative improvements. SCA's simplicity, efficiency, and ability to avoid local optima

have made it a popular choice for a wide range of applications, including engineering design, machine learning, and operations research. Its performance in solving both unimodal and multimodal problems highlights its versatility and robustness in dealing with diverse optimization challenges.

Description of the Sine Cosine Algorithm

Like other population-based optimization algorithms, the sine cosine optimization process starts by randomly initializing a group of representative N solutions, known as search agents, within the search space. This group of N search agents is collectively referred to as the population.

1. **Initialization:** The population in the search space is randomly initialized within the search space bounds. The i^{th} search agent $X_i = (X_i^1, X_i^2 ... X_i^d, ..., X_i^n)$ is initialized using the following equation:

$$X_i^d = X_{i,lb}^d + rand().(X_{i,ub}^d - X_{i,lb}^d) \quad d = 1:n \quad , i = 1:N \tag{3.11}$$

   where X_i^d represents the d^{th} dimension of the i^{th} solution, $X_{i,lb}^d$ and $X_{i,ub}^d$ denote the lower bound and upper bound of the i^{th} solution in the d^{th} dimension of the search space, respectively. The function rand() generates uniformly distributed random numbers in the range [0, 1], and N denotes the number of the search agents in the population, i.e., the population size.

2. **Update of Positions:** After initializing the population within the search space, the next step is to update the position of each search agent to search for the optimal solution. This process involves evaluating the position of each agent using the objective function, which assigns a fitness or goodness value based on the optimization criteria. The search agent with the highest fitness is identified as the best search agent, and its position is

designated as the destination point. Using this destination point as a reference, other search agents then update their positions in the search space. The following equations detail the position update process:

$$X_i^d(t+1) = X_i^d(t) + r_1(t).sin(r_2).|r_3.P^d(t) - X_i^d(t)| \tag{3.12}$$

$$X_i^d(t+1) = X_i^d(t) + r_1(t).cos(r_2).|r_3.P^d(t) - X_i^d(t)| \tag{3.13}$$

where d = 1:n i = 1:N

$X_i^d(t) = (X_i^1(t), X_i^2(t), ...X_i^n(t))$ denotes the position of the i^{th} search agent in the t^{th} iteration. $P(t) = (P^1(t), P^2(t), ..., P^n(t))$ is the search agent having the best fitness (best solution so far) and considered as the destination point at t^th iteration. $|.|$ represents the absolute value, r_1 is a function of iteration t. Calculated using this equation:

$$r_1(t) = b - b.\frac{t}{T} \tag{3.14}$$

Here b is a constant parameter and T denotes the maximum number of iterations.

r_1 and r_2 are uniformly distributed random numbers calculated as the following:

$$r_2 = 2.\pi.rand() \tag{3.15}$$

$$r_2 = 2.rand() \tag{3.16}$$

Exploitation and exploration

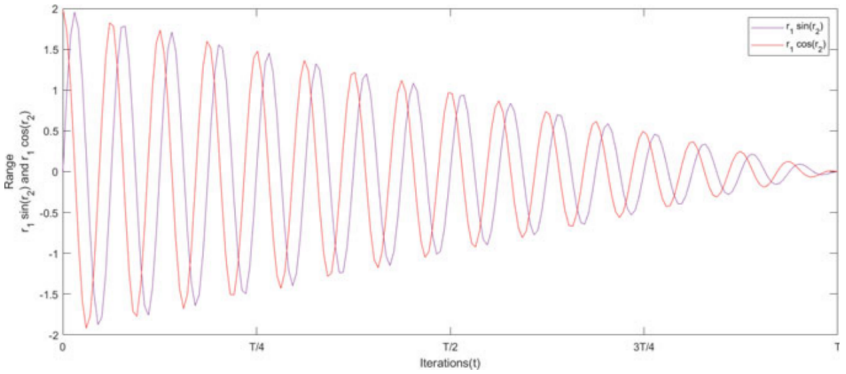

Figure 3.5: Trajectory of r1 sin(r2) and r1 cos(r2) considering b = 2.

Achieving a proper balance between exploration and exploitation is crucial in any population-based optimization algorithm. During the initial phase of the optimization process, or in the early iterations, the algorithm should emphasize exploration to thoroughly scan the design space. As the process progresses to later stages, the algorithm should shift its

focus to exploitation, honing in on promising local regions to locate the global optimum and ensure convergence. Therefore, as the number of iterations increases, the algorithm's exploration capability should decrease while its exploitation capability should increase. In the sine cosine algorithm, the control parameter r_1 is key to maintaining this balance between exploration and exploitation. This parameter ensures a smooth transition from

the exploration phase to the exploitation phase during the search process. The control parameter r_1 is a linearly decreasing function of the iteration counter t, which gradually reduces the value of the constant parameter$b$, the trigonometric functions sine and cosine are multiplied by the control parameter r_1, meaning that the range of these terms depends on the value of r_1. Since r_1 is influenced by the constant parameter b, which decreases linearly as the number of iterations increases, the SCA algorithm effectively controls the range of the terms $r_1.sin(r_2)$ and $r_1.cos(r_2)$ by adjusting the value of the constant parameter b. Moreover, it is clear that the control parameter r_1 works as a scaling factor for

the step size in the position update equations. In the early iterations, the SCA algorithm uses larger values of r_1 to enable search agents to make larger movements and thoroughly explore the search space. As the iterations progress, the value of r_1 decreases, causing the search agents to make smaller movements, which ensures focused exploitation in the promising local regions of the search space.

Random parameter r_4

To enhance the robustness of the sine cosine algorithm, two separate position update equations, or mechanisms, are employed. The choice of whether to use sine or the cosine equation to update the positions of the search agents is determined by a switch probability $p = 0.5$, based on a randomly generated number $r_4 \in [0, 1]$. If $r_4 \leq p$, cosine Equation is used; otherwise, sine Equation is applied. The following equation encapsulates this mechanism:

$$X_i^d(t+1) = \begin{cases} X_i^d(t) + r_1(t)\sin(r_2)\left|r_3 P^d(t) - X_i^d(t)\right| & \text{if } r_4 < p \\ X_i^d(t) + r_1(t)\cos(r_2)\left|r_3 P^d(t) - X_i^d(t)\right| & \text{if } r_4 \geq p \end{cases}$$

The equation above clearly indicates that each update equation has an equal 50% probability of being selected. Due to the absolute value term and the trigonometric sine and cosine functions in the position update equations, the search agents in the SCA algorithm follow a nonlinear search trajectory.

Pseudo code

The pseudo-code for the basic sine cosine algorithm is given in the algorithm below, and the flowchart is shown in Figure under it to provide a concise description of the underlying working procedure of the SCA algorithm.

Algorithm 1 Sine cosine algorithm (SCA)

0: Initialize the population $X_1, X_2, ..., X_N$ randomly in the search space.
0: Initialize the parameters associated with SCA.
0: Calculate the objective function value for each search agent in the population.
0: Identify the best solution obtained so far as the destination point P.
0: **initialize** $t = 0$, where t is iteration counter.
0: **while** Termination criteria is met **do**
0: Calculate r_1, using $r_1 = b - b \cdot \frac{t}{T}$ and generate the parameters r_2, r_3, r_4 randomly.
0: **for** each search agent **do**
0: **if** $r_4 < p$ **then**
0: Update the position of search agents using the cosine equation,
0: **else**
0: Update the position of search agents using the sine equation.
0: **end if**
0: **end for**
0: Update the current best solution (or destination point) P.
0: $t = t + 1$
0: **end while**
0: **return** the best solution P =0

SCA Flowchart

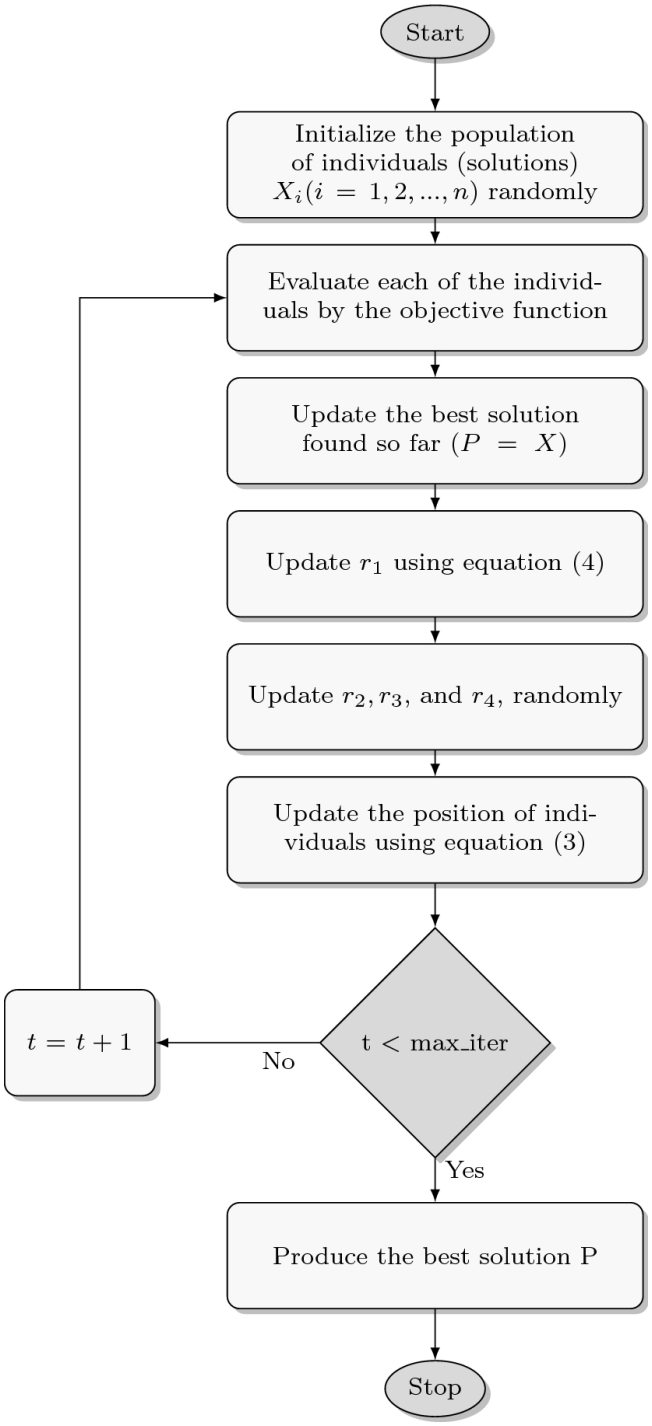

Figure 3.6: Sine Cosine Algorithm flowchart.

3.2.4 Particle Swarm Algorithm

Introduction

The Particle Swarm Optimization (PSO) algorithm is a popular and robust metaheuristic optimization technique inspired by the social behavior of birds flocking or fish schooling. Developed by James Kennedy and Russell Eberhart in 1995, PSO mimics the collective behavior of decentralized, self-organized systems. The algorithm is widely used for solving complex optimization problems across various domains due to its simplicity, flexibility, and ability to converge to optimal or near-optimal solutions.

In PSO, a population of candidate solutions, called particles, explores the search space. Each particle represents a potential solution to the optimization problem and has a position and velocity. The particles move through the search space by updating their velocities and positions based on their own best-known position and the best-known positions of their neighbors or the entire swarm. This iterative process continues until a termination criterion is met, such as a maximum number of iterations or a satisfactory fitness level.

A basic variant of the PSO algorithm operates by utilizing a population, known as a swarm, composed of candidate solutions, referred to as particles. These particles navigate through the search space based on a set of straightforward equations. The movement of each particle is influenced by its own best-known position in the search space as well as the best-known position of the entire swarm, and finally its direction. As particles discover improved positions, these positions subsequently guide the swarm's movements.

In PSO, each candidate solution to a problem is represented as a particle with a specific velocity, moving through the problem space akin to a flock of birds in flight. Each particle's next move is determined by combining its historical best position and current position with those of other swarm members, incorporating some random perturbations. As this process being iterated we expect that the swarm will collectively converge towards the optimum of the objective/cost function. Strengths of PSO algorithms include:

1. **Ease of Implementation:** PSO algorithms are relatively straightforward to implement compared to other optimization methods, making them accessible to a broad user base.
2. **Global Optimization Capability:** PSO algorithms are adept at finding global optimum solutions, avoiding entrapment in local optima.
3. **Versatility:** These algorithms can be applied to a wide variety of optimization problems, encompassing both single and multi-objective scenarios.
4. **High Accuracy:** PSO algorithms have demonstrated high accuracy in finding optimal solutions across numerous optimization problems.

5. **Parallel Implementation:** PSO algorithms can be efficiently implemented in parallel, making them ideal for large-scale optimization problems.

6. **Fast Convergence:** PSO algorithms generally converge more quickly than many other optimization methods, making them suitable for real-time applications.

7. **Robustness:** PSO algorithms are resilient to noisy and uncertain environments, making them practical for real-world applications.

Key Concept

The PSO algorithm is easy to implement, As it can be summerized in the following steps:

1. **Initalization:** A population (swarm) of candidate solutions (particles/flock) is generated randomly within the search space. Each particle has a position representing a potential solution and a velocity dictating its movement. Parameters such as the number of particles, inertia weight, cognitive coefficient, and social coefficient are set. These parameters influence the algorithm's behavior and performance.

2. **Fitness Evaluation:** Each particle's position is evaluated using the objective/cost function of the optimization problem. This evaluation determines the particle's fitness or quality of the solution it represents.

3. **Update Personal and Global Bests:** Each particle in the swarm has its own personal best which is the best personal position it has been to. So for each particle calculate its fitness and compare it the fitness of its personal best. If the current position is better, update its personal best position to the current position. For global best position which is the best position found so far by all the particles in the swarm. We identify the best fitness value among all particles. The position corresponding to this best fitness is designated as the global best.

4. **Velocity Update:** Each particle's velocity is updated based on three components:

    - **Inertia/direction:** The previous velocity of the particle, encouraging it to continue moving in the same direction.
    - **Cognitive Component:** The difference between the particle's current position and its personal best position, encouraging it to move toward its own best-known position.
    - **Social Component:** The difference between the particle's current position and the global position, encouraging it to move toward the best-known position in the swarm.

The formula to update a particle's velocity could be expressed as it follows:

$$v_i^d(t+1) = \omega v_i^d(t) + c_1.r_1.(pBest - x_i^d(t)) + c_2.r_2.(gBest - x_i^d(t)) \tag{3.17}$$

where v^d is the current velocity (current iteration) of the ith particle in the d^th dimension, ω is the inertia weight, c_1 and c_2 are cognitive and social coefficients, respectively, and r_1 and r_2 are randomly generated numbers between 0 and 1. $pBset$ and $gBest$ are respectively personal and global bests. And finally x^d is the position of the particle in the d^{th} dimension at the iteration t.

5. **Position Update:** Each particle's position is updated based on its new velocity:

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1) \tag{3.18}$$

where $x_i^d(t)$ is the last position (previous iteration) of the i^th particle in the d^th dimension. $v_i^d(t+1)$ is the updated velocity.

6. **Check Stopping Criteria:** The algorithm must checks if the stopping criteria are met. Common stopping criteria include reaching a maximum number of iterations, achieving a satisfactory fitness level, or observing negligible improvement over successive iterations. If its not met then repeat steps 3-5.

7. **Output The Best Agent:** Upon meeting the stopping criteria, the algorithm outputs the $gBest$ position and its corresponding fitness value as the optimal solution.

These steps illustrate the iterative process through which the PSO algorithm converges toward an optimal solution by simulating the social behavior of particles in a swarm.

Flowchart

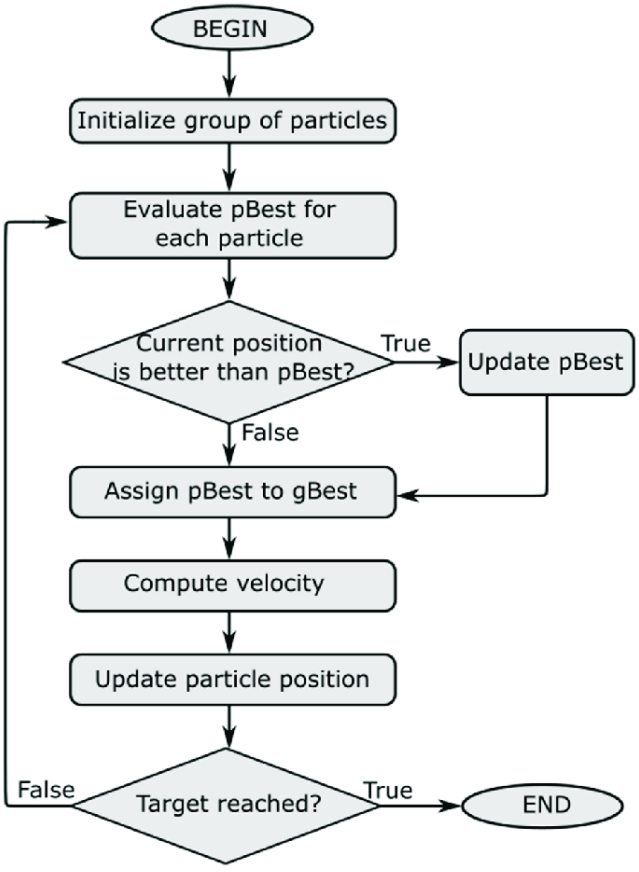

Figure 3.7: Particle Swarm Algorithm flowchart.

Algorithm 2 Particle Swarm Algorithm (PSO)

0: Initialize the population $\{x_1, x_2, \ldots, x_N\}$ randomly in the search space
0: Initialize the parameters associated with PSO
0: Calculate the objective/fitness function value for each search particle in the population
0: Identify the best solution obtained so far as the destination point P
0: **initialize** $t = 0$, where t is iteration counter
0: **while** Termination criteria is met **do**
0: **for** each search particle **do**
0: Calculate the particle's velocity using the formula: $v_i^d(t+1) = \omega v_i^d(t) + c_1.r_1.(pBest - x_i^d(t)) + c_2.r_2.(gBest - x_i^d(t))$
0: Calculate the particle's position by: $x_i^d(t+1) = x_i^d(t) + v_i^d(t+1)$
0: **if** Particle's position is better than its Personal Best **then**
0: Update its personal best
0: **if** Particle's position is better than the swarm's Global Best **then**
0: Update the swarm's Global Best.
0: **end if**
0: **end if**
0: **end for**
0: $t = t + 1$
0: **end while**
0: **return** the best solution P =0

Chapter 4

Hybridation Of Divide And Conquer And Metaheuristics

In this project, several tools and libraries were employed to enhance medical images using Divide And Conquer method and Metaheuristics. The use of these tools facilitated various aspects of the image processing workflow, from reading and manipulating medical images to visualizing and analyzing the results. The main tools used in this project are:

- **Python**: A versatile and powerful programming language that is widely used in scientific computing and data analysis.
- **Matplotlib**: A plotting library for Python, used for creating static, animated, and interactive visualizations.
- **OpenCV**: An open-source computer vision and machine learning software library, used for image processing and computer vision tasks.
- **NumPy**: A library for the Python programming language, used for support of large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
- **pydicom**: A Python package for parsing DICOM (Digital Imaging and Communications in Medicine) files, used for handling medical imaging data.

The complete source code for this project, along with detailed instructions and documentation, will be available on my GitHub repository. This will enable others to replicate the results, explore the methods used, and contribute to further enhancements. You can access the repository at the following URL:

4.1 Hybridation Of Divide And Conquer And Metaheuristics

In this project, we employ a hybrid approach combining the divide and conquer method with metaheuristics to optimize the Effecetive Measure of Enhancement function (EME), which serves as an effective measure of image enhancement.

The divide and conquer method breaks down the image processing task into manageable sub-problems by applying various filters to the image and analyzing the results in smaller blocks. This granular approach ensures that local details and features are effectively captured.

To find the optimal combination of these filtered images, we utilize Particle Swarm Optimization (PSO), Flower Pollination Algorithm (FPA), Gravitational Search Algorithm (GSA) and Sine Cosine Algorithm (SCA), some powerful metaheuristic algorithms. By exploring and exploiting the search space, these algorithms identify the optimal weights for the filtered images, thereby minimizing the EME. This hybridization leverages the strengths of both methods, achieving a robust and efficient solution for enhancing medical images.

4.2 Results

The results of this project demonstrate the effectiveness of the hybrid approach combining divide and conquer methods with metaheuristics in optimizing medical image enhancement. By applying various filters and using those Metaheuristics to find the optimal combination, we were able to significantly increase the (EME) of the images. This reduction in EME indicates a substantial improvement in image clarity and detail, making the enhanced images more suitable for medical analysis and diagnosis. The comparative analysis of the original and optimized images, presented alongside their respective EME values, clearly illustrates the enhancements achieved through our method:

4.2.1 Results: Particle Swarm Optimization

The results obtained using PSO with the specified parameters are as follows:

- **Population size:** 100
- **Number of dimensions:** 4
- **Bounds:** $[0, 1] \times [0, 1] \times [0, 1] \times [0, 1]$
- **Number of iterations:** 100

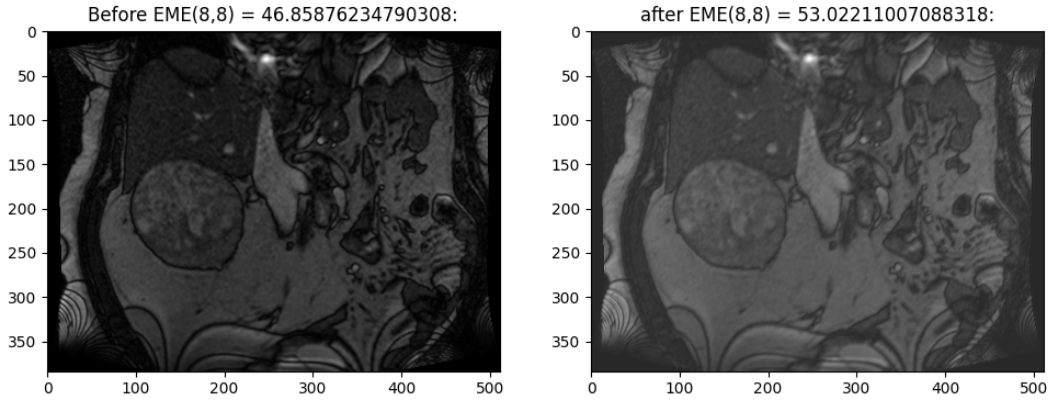

Figure 4.1: An image with its color values over it.

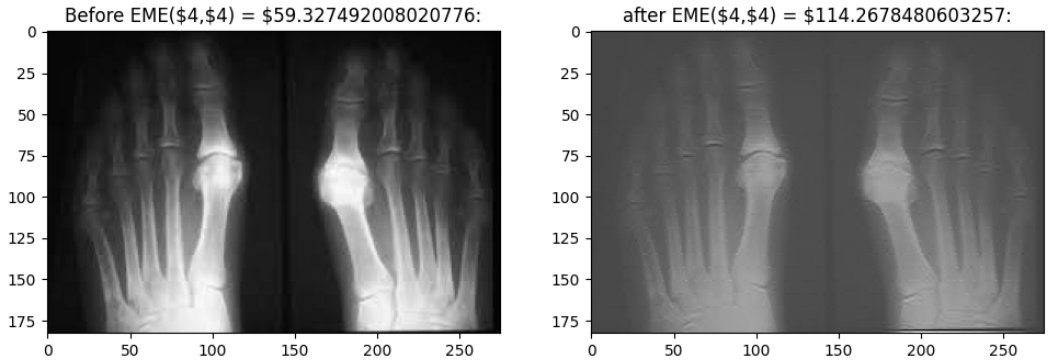

Figure 4.2: An image with its color values over it.

4.2.2 Results: Sine Cosine Algorithm

The results obtained using SCA with the specified parameters are as follows:

- **Population size:** 100
- **Number of dimensions:** 4
- **Bounds:** $[0, 1] \times [0, 1] \times [0, 1] \times [0, 1]$

- **Number of iterations:** 100
- **Constant parameter b:** 2

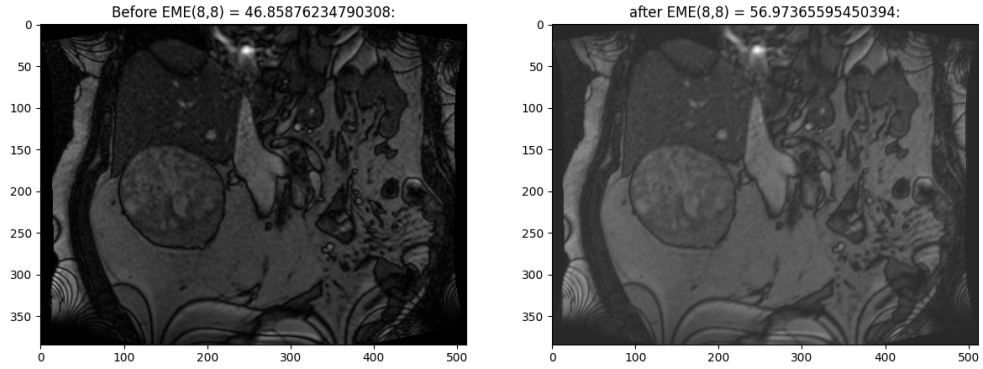

Figure 4.3: An image with its color values over it.

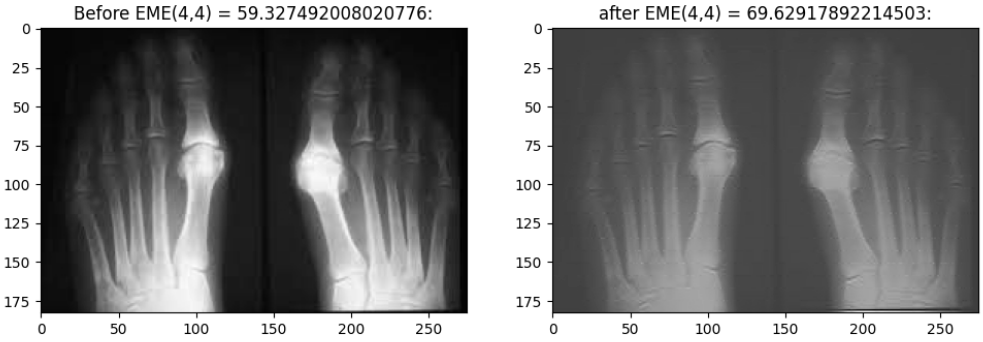

Figure 4.4: An image with its color values over it.

4.2.3 Results: Flower Pollination Algorithm

The results obtained using FPA with the specified parameters are as follows:

- **Population size:** 100
- **Number of dimensions:** 4
- **Bounds:** $[0, 1] \times [0, 1] \times [0, 1] \times [0, 1]$
- **Number of iterations:** 100
- **Gamma:** .5
- **lambda:** .5
- **Probability switch between local and global pollinations:** 0.8
- **s:** 0.2

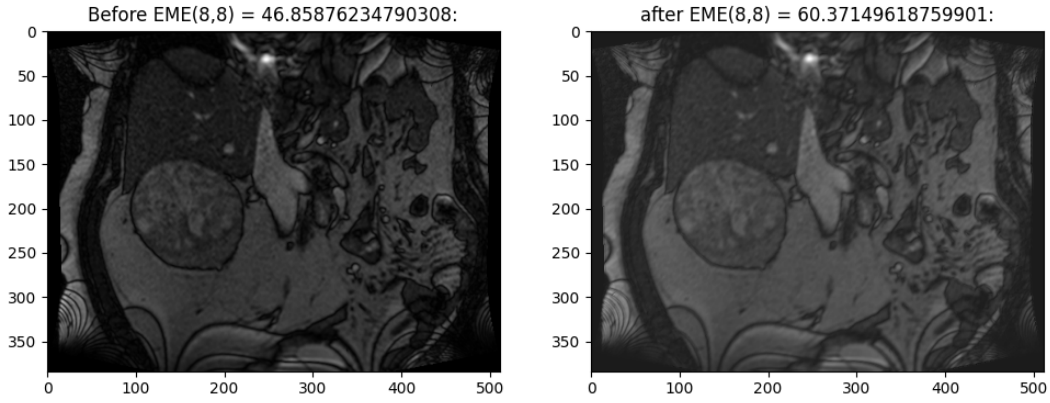

Figure 4.5: An image with its color values over it.

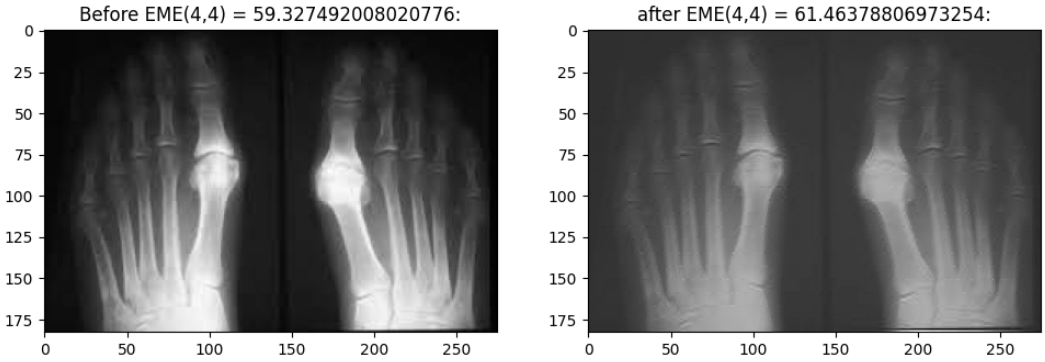

Figure 4.6: An image with its color values over it.

4.2.4 Results: Gravitational Search Algorithm

The results obtained using GSA with the specified parameters are as follows:

- **Population size:** 100
- **Number of dimensions:** 4
- **Bounds:** $[0, 1] \times [0, 1] \times [0, 1] \times [0, 1]$
- **Number of iterations:** 100
- **Alpha** b**:** .5
- **G0** b**:** .5
- **P** b**:** 2

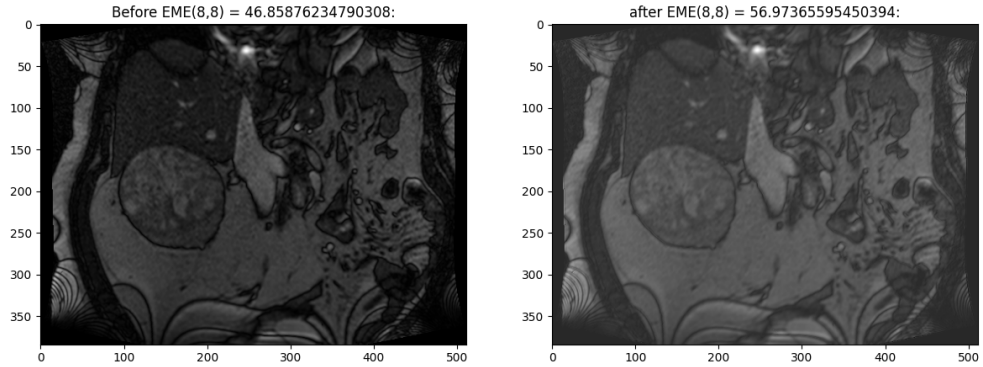

Figure 4.7: An image with its color values over it.

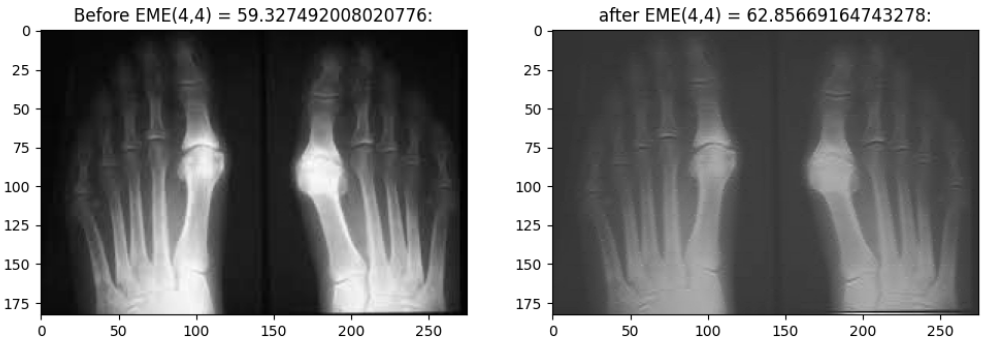

Figure 4.8: An image with its color values over it.

4.3 Conclusions And Perspectives

The implementation of various metaheuristic algorithms, including the Particle Swarm Optimization (PSO), Sine Cosine Algorithm (SCA), Flower Pollination Algorithm (FPA), and Gravitational Search Algorithm (GSA), for the enhancement of medical images has demonstrated improvements in image quality. The optimization of filter weights through these algorithms has effectively enhanced the Effective Measure of Enhancement (EME), resulting in more diagnostically useful images.

The PSO algorithm, with its ability to converge quickly to an optimal solution, provided a strong baseline for performance. The Sine Cosine Algorithm introduced additional flexibility and diversity in search patterns, leading to comparable, if not superior, results in certain cases. The Flower Pollination Algorithm, inspired by the natural pollination process, excelled in balancing exploration and exploitation, enhancing image details effectively. Lastly, the Gravitational Search Algorithm, leveraging the principles of gravity and mass interactions, showcased robust performance in optimizing image enhancement parameters.

Overall, the hybridization of the divide and conquer method with these metaheuristics has proven to be an effective strategy for optimizing image enhancement. Each algorithm contributed uniquely to the enhancement process, demonstrating the value of exploring different Metaheuristic approaches in medical image processing tasks. The improved EME values before and after optimization confirm the efficacy of these methods in enhancing image clarity and detail, thereby potentially aiding in better medical diagnoses.

Bibliography

[1] Hybridization of Divide-and-Conquer technique and Neural Network algorithm for better contrast enhancement in medical images - Aqel F.1 , Alaa K.2 , Alaa N. E.3 , Atounti M.2

[2] GSA: A Gravitational Search Algorithm - Esmat Rashedi, Hossein Nezamabadi-pour *, Saeid Saryazdi

[3] https://en.wikipedia.org/wiki/Divide-and-conquer_algorithm

[4] Flower Pollination Algorithm: An Introduction - Xin-She Yang

[5] A Comprehensive Survey on Gravitational Search Algorithm - Esmat Rashedi1 , Elaheh Rashedi2 , Hossein Nezamabadi-pour3

[6] Image Convolution - Jamie Ludwig Satellite Digital Image Analysis, 581 Portland State University

[7] A New Measure of Image Enhancement - Sos S. Agaian, Karen P. Lentz, and Artyom M. Grigoryan

[8] Sine Cosine Algorithm for Optimization - Jagdish Chand Bansal · Prathu Bajpai · Anjali Rawat · Atulya K. Nagar

[9] Mammographic Image Enhancement Using Indirect Contrast Enhancement Techniques – A Comparative Study - K.Akilaa, L.S.Jayashreeb, A.Vasukic

[10] Image enhancement using divide-and-conquer strategy - Peixian Zhuang, Xueyang Fu, Yue Huang, Xinghao Ding

[11] https://medium.com/image-processing-with-python/image-enhancement-with-python-d3040a39e394

[12] https://dataanalyticspost.com/Lexique/particle-swarm-optimization/

[13] https://fr.wikipedia.org/wiki/Optimisation_par_essaims_particulaires

[14] https://en.wikipedia.org/wiki/Convolution

[15] What is Convolution - 3Blue1Brown - https://www.youtube.com/watch?v=KuXjwB4LzSAt=79s

[16] Gravitational Search Optimization Algorithm: A Review - Sujata L Patekar1, Supriya L Patekar2

[17] https://nbia.cancerimagingarchive.net

[18] https://numpy.org/doc/

[19] https://docs.opencv.org/4.x/

[20] https://link.springer.com/chapter/10.1007/978-981-19-9722-8_2: :text=Sine%20cosine%20algorithm%20(SCA)%20%5B,trigonometric%20functions%20sine%20ar

[21] https://github.com/7ossam81/EvoloPy/tree/master

[22] https://github.com/prashamsatalla/Underwater-image-color-enhancement-with-PSO-python-implementation

[23] https://seyedalimirjalili.com/pso