CHAPTER IV

ENCRYPTION IMAGE USING GENETIC ALGORITHM

4.1. Introduction to Soft Computing Algorithms

The soft computing algorithms are a set of techniques covering many fields that belong to various categories in computational intelligence. Soft computing algorithms consist of several computing paradigms particularly:

(i) Genetic Algorithms, (ii) Neural Networks and (iii) Fuzzy System. The idea of soft computing algorithms started in 1981 by Lotfi A. Zadeh, when his first paper “what is soft computing” was published on soft data analysis. [123] Table 4.1 shows the brief history of soft computing algorithm development.

<table>
<thead>
<tr>
<th>Evolutionary Computing (EC)</th>
<th>Rechenberg 1960</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Networks (NN)</td>
<td>McCulloch 1943</td>
</tr>
<tr>
<td>Fuzzy Logic (FL)</td>
<td>Zadeh 1965</td>
</tr>
<tr>
<td>Genetic Programming (GP)</td>
<td>Koza 1992</td>
</tr>
<tr>
<td>Evolution Strategies (ES)</td>
<td>Rechenberg 1995</td>
</tr>
<tr>
<td>Evolutionary Programming (EP)</td>
<td>Fogel 1962</td>
</tr>
<tr>
<td>Genetic algorithms (GA)</td>
<td>Holland 1970</td>
</tr>
</tbody>
</table>

Genetic algorithms (GAs) are one of the soft computing algorithms branches. (GAs) is an important method of artificial intelligence, which has been applied to generate encryption ‘key’, which plays a vital role in any type of encryption. This chapter includes a proposition of three new approaches of algorithms for image encryption using (GAs). The first approach is based on the concept of (GAs), which is used to output new encryption method by exploiting the feature of the crossover and mutation operation of genetic algorithms. The second approach is based on bit-level using
(GAs) with pseudo random sequence to encrypt image stream. The third approach is based on real coded genetic algorithm and toral automorphism to identify the best fit pixels of encryption algorithm with the help of fitness function. High security and high feasibility characterize these approaches, so these features ease the integration with digital image encryption.

4.2. Basic Concepts of Genetic Algorithms

Genetic Algorithms have been originated from the studies of cellular automata conducted by John Holland and his colleagues at the University of Michigan. [124-126] A genetic algorithm is a searching technique used in computer science to find approximate solutions to optimization problems. GAs are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, natural selection and recombination (or crossover). Once we have the genetic representation and the fitness function defined, GA proceeds to initialize a population of solutions randomly, and then improves it through repetitive application of mutation, crossover and selection operators. Researchers have adopted GA as a solution to optimization in various fields in recent years. GA is a solution to optimization problem started gaining popularity towards the end of the last century as used to solve optimization problems in construction. Its intrinsic parallelism facilitates the uses of distributed processing machines such as Distribution Network Planning.

The GAs start with guesses and attempt to improve the guesses by evolution. GAs typically have five parts:

1. A representation of a guess called a chromosome.
2. An initial pool of chromosomes.
3. A fitness function.
5. A crossover operator and a mutation operator. [127,128]
A chromosome can be a binary string or more elaborate data structure. The initial pool of chromosomes can be randomly produced or manually created. The fitness function measures the suitability of a chromosome to meet a specified objective. For covering based ATG, a chromosome is fitter if it corresponds to greater coverage. The selection function decides which chromosomes will participate in the evolution stage of the genetic algorithms made up by the crossover and mutation operators. The crossover operator exchanges genes from two chromosomes and creates two new chromosomes. The mutation operator changes a gene in a chromosome and creates one new chromosome. GA has well-defined steps: A basic algorithm for a GA is as follows:

*The pseudo code for GA is:*

```plaintext
Initialize (population)
Evaluate (population)
While (stopping condition not satisfied)
do
|
| Selection (population)
| Crossover (population)
| Mutate (population)
| Evaluate (population)
|
```

The algorithm will iterate until the population has evolved to form a solution to the problem, or until a maximum number of iterations takes place. [129]

Problems which can be solved using genetic algorithms include timetabling, scheduling problems and many scheduling software packages based on GA. [130]

The Genetic Algorithms (GAs) are exploring algorithms based on the theory of natural selection with an inventive fitness of nature. The central idea of research on GAs has been robustness [131]. The implications of robustness are the removal of costly resigns and higher level of variation. The depiction of a natural population is done using chromosomes, which are generally a set of binary numbers. Each number represents a cell and can be perceived as an affirmative or negative answer.
The basic idea behind GAs begins with a set of candidate solutions (chromosomes) called population. A new population is created from solutions of an old population in hope of getting a better population. Solutions which are chosen to form new solutions (offspring) are selected according to their fitness. The more suitable the solutions are the bigger chances they have to reproduce. This process is repeated until some condition is satisfied [132].

4.2.1. Basic elements of GAs
Most of GAs methods are based on the following elements:
(i) Populations of chromosomes
(ii) Selection according to fitness
(iii) Crossover to produce new offspring
(iv) Random mutation of new offspring. [133]

4.2.1.1. Chromosomes
The chromosomes in GAs represent the space of candidate solutions. Possible chromosomes encodings are binary, permutation, value, and tree encodings. For the Knapsack problem, we use binary encoding, where every chromosome is a string of bits, 0 or 1.

4.2.1.2. Fitness Function
GAs require a fitness function which allocates a score to each chromosome in the current population. Thus, it can calculate how well the solutions are coded and how well they solve the problem. [133]

4.2.1.3. Selection
It is a quantitative criterion based on fitness value to choose which chromosomes from population will go to reproduce. Intuitively, the chromosome with more fitness value will be considered better and in order to implement proportionate random choice, Roulette wheel selection is used for selection. [134, 136] GAs are different from the other search processes owing to the fact that they work on coding of the
parameter set and not on the parameters. [137] It is also general belief that GAs use payoff and not auxiliary knowledge. Moreover, determinism is not needed in GAs. The initial population for GAs is generated by applying the following procedure. [131, 134,135]

Initial population is stored in a 2D array, let it be called init_pop[].

```
for i = 0 to n
begin
for j = 0 to m
begin
    Generate a random number x modulo 100
    If(x <= 50) then init_pop[i][j] = 0
    else init_pop[i][j] = 1
end
end
```

4.2.1. Crossover
Crossover operator has the significance as that of crossover in natural genetic process. In this operation, two chromosomes are taken and a new one is generated by taking some attributes of the first chromosome and the rest from the second chromosome. In GAs, a crossover can be classified into the following types: [87-89]

4.2.1.1. Single Point Crossover: In this type, a random number is selected from 1 to n as the crossover point, where n is the number of chromosome. Any two chromosomes are taken and an operator is applied.

```
<table>
<thead>
<tr>
<th>Before</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>After</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```
4.2.1.4.2. Two Point Crossover:
In this type of crossover, two crossover points are selected and the crossover operator is applied.

Before:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

After:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

4.2.1.4.3. Uniform Crossover:
In this type, bits are copied from both chromosomes uniformly.

Before:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

After:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 4.1 displays the crossover operation. The crossover is the process in which the strings are able to mix and match their desirable qualities in a random fashion. The crossover operation proceeds in three simple steps:

Two new random strings are selected in figure 4.1(a). A random location in both strings is selected in figure 4.1(b). The portions of the strings to the right of the randomly selected location in the two strings are exchanged in Figure 4.1(c). In this way, information is exchanged between strings, and portions of two strings are exchanged and combined.

Figure 4.1 Illustration of the Crossover Operation
4.2.1.5. Mutation

Mutation is a genetic operator used to maintain genetic diversity from one generation of population to the next. It is similar to biological mutation. Mutation allows the algorithm to avoid local minima by preventing the population chromosomes from becoming too similar to each other. [135] GAs involve string-based modifications to the elements of a candidate solution. These include bit-reversal in bit-string GAs. [131]

\[
\begin{array}{cccccccc}
1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 1 & 0
\end{array}
\]

4.3. Mechanism of GAs

The fundamental mechanism consists of the following stages which are also described in figure 4.2.

1. Generate randomly the initial population.
2. Select the chromosomes with the best fitness values.
3. Recombine selected chromosomes using crossover and mutation operators.
4. Insert offspring into the population.
5. If a stop criterion is satisfied, return the chromosome(s) with the best fitness. Otherwise, go to Step 2.

In GA, the population is defined to be the collection of individuals. A population is a generation that undergoes under changes to produce new generation. Like nature, GAs have also collection of several members to make population healthy. A chromosome that is a collection of genes is correspondence to individual of population. Each individual chromosome represents a possible solution to the optimization problem. The dimension of the GA refers to the dimension of the search space which equals the number of genes in each chromosome.
Figure 4.2 Fundamental Mechanisms of Simple Genetic Algorithms

4.4. Applications of Genetic Algorithms

There are many applications of genetic algorithms in different fields. These applications are concerned with problems which are hard to be solved but have easily verifiable solutions. Another common trait, which belongs to these applications, is the equation style of fitness function. Cryptography and cryptanalysis could be considered to meet these criteria. However, cryptology is not closely related to the typical GA application areas and, subsequently, fitness equations are difficult to generate. This makes the use of a genetic algorithms approach to cryptology rather unusual. Genetic algorithms find application in bioinformatics, phylogenetics, computational science, engineering, economics, chemistry, manufacturing, mathematics, physics, pharmacometrics, and other fields such as image enhancement, image processing, and image cryptography. [138]
4.4.1. Image Enhancement

The function of (GAs) can be applied to work on the image for enhancement. The main steps for solving a problem using GAs are:

1. Initializing the population of possible solutions.
2. Calculating the evaluation i.e. fitness function that plays the role of the environment and rating the solution in terms of their ’fitness’.
3. Defining the genetic operators (selection, crossover, mutation) that alter the composition of children during reproduction.
4. Establishing values for the parameters (population size, probabilities of applying genetic operators) that the genetic algorithms use. [139, 140]

4.4.2. Image Processing

The first task of machine vision is to enhance image quality in order to obtain a required image perception. It is done by removing noise, amplifying image contrast and amplifying the level of details [141].

4.4.3. Information Security

The application of genetic algorithms (GAs) to the field of cryptology is rather unique. Few works exist on this topic. Genetic algorithms are evolutionary algorithms based on the notion of natural selection [142]. The genetic algorithms have been proven to be reliable and powerful optimization technique in a wide variety of applications. It can be applied to both texts and images. Genetic algorithms are secure since they do not directly utilize the natural numbers. The results obtained for generating keys using genetic algorithms should be good in terms of coefficient of autocorrelation. Generally, genetic algorithms have two basic functions, namely crossover and mutation [143].
4.4.4. Airlines Revenue Management

With the increasing interest in decision support systems and the continuous development of computer science, revenue management is a discipline which has received a great deal of interest in recent decades. Revenue management is the control of inventory and pricing of a perishable product in order to improve the efficiency of its marketing. Although revenue management has seen many new applications throughout the years, the main focus of research continues to be the airline industry. In the proposed system, an optimized flight booking and transportation terminal open/close decision system is presented using Genetic Algorithms in order to maximize the revenue of airline. In this system, the particular booking terminal’s historical booking data is observed. Consequently, its frequency is generated with linguistic variable and the deviation of booking is interpreted. Using the observed data and genetic algorithms, the terminal open/close decision system is optimized. Finally, the experimentation is performed with the synthetic data to prove the significance of the system [144]

4.4.5. Robotics

The genetic algorithms are the most common algorithms used in the field of robotics for navigating a robot in its environment. In the case of a mobile robot, an optimization algorithm presents a navigation system which will reach its given objective without getting lost or colliding with another object in its environment.

[145, 146]
4.5. Properties of Genetic Algorithms

Some properties of GA are as follows:

- The GA can solve optimization problem which can be described with chromosome encoding.
- It solves problems with multiple solutions.
- Its structural gives the possibility to solve the solution structure and solution parameter problems at the same time by means of GA.
- GA is a very easy method which can be understood and practically it does not demand the knowledge of mathematics.
- GAs are easily transferred to existing simulations and models.[147]

4.6. Proposed Algorithms Based on Genetic Algorithms

4.6.1. Using Crossover and Mutation Operators of GAs

In this approach, a new algorithm is proposed for image security using the concept of genetic algorithms based on crossover and mutation features. This algorithm provides more secure image encryption, minimum lossless in data, maximum speed and maximum distortion in encryption, with the addition of extra degree of protection. The encryption processes of this approach are summarised in the next steps:

1. Loading an image.
2. Determining the size of image (height and width).
   2.1. Checking the result of \( (H \mod 8) \) and \( (W \mod 8) \). If they are equal to zero then go to 3rd step otherwise do

\[
H = H + (8 - (H \mod 8)) \quad \text{and} \quad W = W + (8 - (W \mod 8)).
\]  

(4.1)

3. Dividing the image into a set of blocks, each block size is \( (8 \times 8) \).

Considering a block \( B (w \times h) \) where \( w \) and \( h \) are width and height of \( B \).

4. Extracting the RGB components and store them as \( R \), \( G \) and \( B \) matrix
\[ R = R(r \times c), \quad G = G(r \times c), \quad B = R(r \times c). \]  

\hspace{1cm} (4.2)

5. Doing crossover operator.

Crossover proceeds in three simple steps:

5.1. Select randomly two strings from the block, one vertically and one horizontally.

5.2. Select a random location from strings.

5.3. Swap together the portions of strings on right side. A secret key is used for crossover. In this research, the secret key has two attributes termed \( a, b \) belonging to 1. The crossover is done by swapping \( a \) to \( b \) in each vector.

6. Doing mutation operator

For each vector \( V_i \) mutation is done by another secret key of single variable of \( k \), and by the following equation: 
\[ V_i[block] = 255 - V_i[block]. \] (4.3)

6.1. Constructing an encrypted block from the set of \( N \) vector produced from the mutation.

7. Getting the encrypted block

8. Repeating the 4th, 5th and 6th steps for all blocks to get the encrypted image

### 4.6.2. Bit Level by Using Crossover and Mutation of GAs

1. Loading an image.

2. Determining the height and width of image (H and W).

2.1. Checking the result of \( H \ mod \ 8 \) and \( W \ mod \ 8 \). If they are equal to 0 then go to 4th step otherwise do \( H = H + [8 - (H \ mod \ 8)] \) and \( W = W + [8 - (W \ mod \ 8)] \).

3. Dividing the image into a set of blocks, each block size is \( (8 \times 8) \).

Considering a block \( B \ (w \times h) \) where \( w \) and \( h \) are width and height of \( B \).
4. Separating each block into three greyscale images of RED, GREEN and BLUE, arrange the pixels from row to column, in this case three sequences are getting $P_R$, $P_G$, and $P_B$, as follows:

$$
\begin{align*}
P_R &= \{r_1, r_2, \ldots, r_{wh}\} \\
P_G &= \{g_1, g_2, \ldots, g_{wh}\} \\
P_B &= \{b_1, b_2, \ldots, b_{wh}\}
\end{align*}
$$

(4.4)

where $r_i$, $g_i$ and $b_i$ are the $i$th pixel values of RED, GREEN and BLUE components between 0 and 255.

5. Slicing each pixel in each component of RGB into eight binary pixels, ranging from (0) to (7) for the least significant bit (LSB) and most significant bit (MSB) respectively.

6. Doing crossover operation.

The crossover proceeds in three simple steps:

a) Selecting randomly two strings from the block, one vertically and one horizontally.

b) Selecting a random location from strings selected.

c) Swapping together the portions of strings on right side. The secret key is used for crossover.

7. Doing mutation operation

For each vector $V_i$, do mutation by another secret key of single variable of $k$, and by doing the following equation:

$$
V_i[\text{block}] = 255 - V_i[\text{block}].
$$

8. Reconstructing the RGB components for all blocks

9. Repeating the 4th, 5th, 6th, 7th and 8th steps for all blocks to get encrypted image
4.6.3. Image Encryption Using Real Coded Genetic Algorithm and Toral Automorphism

This section introduces a new type of genetic algorithm which named the real coded genetic algorithm for function optimization. This algorithm combines the advantages of traditional real genetic algorithm with toral automorphism. The specific features of this algorithm make it able to solve more complex problems. It is one of variant of GAs for real–valued optimization that is closer to the original genetic algorithm.

Let us assume that we are dealing with a free $N$ dimensional real-valued optimization problem, which mean $x = \mathbb{R}^N$ without constraints. In a real coded GA, an individual is represented as an $N$ dimensional vector of real numbers.

$$B = (x_1, x_2, ..., x_N)$$  \hspace{1cm} (4.5)

As selection does not involve a particular coding, no adaption needs to be made – all selection schemes are applicable without any restriction. What has to be adapted to this special structure are the genetic operations crossover and mutation.

**Crossover Operators for Real Coded GA**

Flat crossover: gives two parents $b^1 = (x^1_1, ..., x^1_N)$ and $b^2 = (x^2_1, ..., x^2_N)$, a vector of random values from the interval unit $(\lambda_1, ..., \lambda_N)$ is chosen and the offspring $b = (x'_1, ..., x'_N)$ is computed as a vector of linear combinations in the following way for all $i = 1, ..., N$.

BLX-$\alpha$ Crossover is an extension of flat crossover, which allows an offspring Allele to be also located outside the interval

$$[\min(x^1_i, x^2_i), \max(x^1_i, x^2_i)] \hspace{1cm} (4.6)$$

In BLX-$\alpha$ crossover, each offspring Allele is chosen as a uniformly distributed random value from the interval
\[ \{ \min(x^1_i, x^2_i) - I \cdot \alpha, \max(x^1_i, x^2_i) + I \cdot \alpha \} \quad (4.7) \]

Where \( I = \max(x^1_i, x^2_i) - \min(x^1_i, x^2_i) \). The \( \alpha \) has to be chosen in advance. For \( \alpha = 0 \), BLX-\( \alpha \) crossover becomes identical to flat crossover.

Simple Crossover is nothing else but classical one-point crossover for real vectors, i.e., a crossover site \( k \in \{1, \ldots, N-1\} \) is chosen and two off springs are created in the following way:

\[
\begin{align*}
    b^1 &= (x^1_1, \ldots, x^1_k, x^2_{k+1}, \ldots, x^2_N) \\
    b^N &= (x^2_1, \ldots, x^2_k, x^1_{k+1}, \ldots, x^1_N)
\end{align*}
\quad (4.8) \quad (4.9)
\]

Discrete Crossover is analogous to classical uniform crossover for real vectors.

An offspring \( b \) of the two parents \( b_1 \) and \( b_2 \) is composed from Alleles, which are randomly chosen either as: \( x^1_i \) or \( x^2_i \).

**Mutation Operators for Real-Coded GAs**

The following mutation operators are most common for real coded GAs:

Random Mutation: For a randomly chosen gene \( I \) of an individual \( b = (x_1, x_2, \ldots, x_N) \), the Allele \( x_i \) is replaced by a randomly chosen value from a predefined interval \([a_i, b_i]\).

Non-uniform Mutation: In non-uniform mutation, the possible impact of mutation decreases with the number of generations.

**Image Encryption Using Toral Automorphism**

Toral automorphisms are strongly chaotic mixing systems. A two dimensional toral automorphism is a spatial transformation applied to two dimensional square planar regions. It consists of constant elements representing the toral automorphism aping of the form:

\[
\begin{pmatrix}
    1 & 1 \\
    k & k+1
\end{pmatrix}
\]
It is constrained to have elements that belong to the set of positive integers and determinants of 1. The iterated application of $A$ on a point’s $r$ belonging to the integer lattice $L$ and having coordinates as $x$ and $y$ results in a dynamic system that can be represented as: $r_{n+1} = A r_{n} (mod 1), \ n=0,1,...255$ (4.10)

**Fitness Function:**

A fitness function value quantifies the optimality of a solution. The value is used to rank a particular solution against all the other solutions. A fitness value is assigned to each solution depending on how close it is actually to the optimal solution of the problem. Every pixel in the binary form will be the chromosomes of our genetic algorithm. The fitness value of each pixel will be calculated on the basis of the change into its brightness and colour strength of it after embedding a message into it.

```c
// br1 Brightness of original image
// br2: Brightness of stego image
// colr1: Colour strength of original image
// colr2: colour strength of stego image.

Int Fitness function (br1, colr1, br2, colr2)
{
    Fitness= mod((br1+colr1)-(br2+colr2));
    return (Fitness);
}
```
The Peak Signal to Noise ratio of the original and encrypted image should be calculated (PSNR) as:

```plaintext
// size_host: size of original image
// m*n: dimensions of original image
// size_stego: size of stego image
// sum=0
PSNR()
{
  For i=0 to size_host
  {
    sum=sum+(size_host(i)-size_stego(i)^2;
    mse=sum/size_host;
    psnr=10*log 10(m*n)/mse;
  }
}
```

**Steps for genetic algorithm execution**

1) Define the fitness function
2) Add fitness function into GA tool box, as shown in screen shot @simple_fitness
3) Set the number of variables for fitness function
4) Set other parameters for genetic algorithm
5) Click on start button
6) The optimum values are generated using GA according to fitness function

Use these values for image encryption
4.7. Results and Analysis

This section presents several experiments to perform a test on the effect of the proposed algorithms on image encryption. The proposed approaches are implemented and applied to three different images in types and sizes to verify their adaptability, quality, security and speed. The results are illustrate in this section to confirm the properties of the proposed approaches based on genetic algorithms and to compare them with the quality of image encryption. The security analysis performed in this approaches is visual analysis, histogram analysis, correlation coefficient. All these experiments prove that the proposed approaches in this chapter fulfill all the results completely and rightly. In this subsection, the discussions and analyses of the encryption algorithms based on genetic algorithms are presented. The following discussions and analyses are performed on original, encrypted and decrypted images. In these experiments, the algorithm parameters are set as follows: the images sizes are different; the length of images blocks is 64 pixels (8*8). Image encryption and decryption tests have been carried out using standard images of different sizes in gray scale and colour. Encrypted and decrypted outputs have been obtained from crossover and mutation of genetic algorithms and bit level using genetic algorithms and presented in the following figures. Figure 4.3 shows the original colour image of Lena and the corresponding cipher image and decrypted image by crossover and mutation operations of genetic algorithms. Figure 4.4 shows the original colour image of Lena and the corresponding cipher image and decrypted image by bit level using genetic algorithms. Figure 4.5 shows the original colour image of Lena and the corresponding cipher image and decrypted image by real coded genetic algorithm and toral automorphism. It may be noted that no trace of original image is visible in the encrypted image obtained from the proposed approaches in this chapter.
Encryption and Decryption Image Using Multiobjective Soft Computing Algorithm

Figure 4.3 Encryption and decryption of Image ‘Lena’ by cross over and mutation operations of genetic algorithms

Figure 4.4 Encryption and decryption of Image ‘Lena’ by bit level using genetic algorithms

Figure 4.5 Encryption and decryption of Image ‘Lena’ by real coded genetic algorithm and toral automorphism cipher image
4.7.1. Visual Analysis

The purpose of visual testing is to highlight the presence of the similarities between plain image and its cipher. Figures 4.3 - 4.5 show that the encrypted images do not contain any features of the plain images. Visual testing was performed on different images, which differ in sizes and formats and show that there is no perceptual similarity.

4.7.2. Statistical Analysis

Statistical analysis has been performed on the proposed image encryption approaches, demonstrating its superior confusion and diffusion properties, which strongly resist statistical attacks. This is shown by the test on the histograms of the enciphered images, and on the correlations of the adjacent pixels in the ciphered image.

4.7.2.1. Histogram Analysis

Histogram analysis gives the idea of statistical analysis attackers. Statistical analysis has been performed on the proposed approach, demonstrating its superior confusion and diffusion properties, which strongly resist statistical attacks. The proposed approaches give the original image and cipher image histogram. Figures 4.6 - 4.11 show the corresponding histograms of crossover and mutation operations of genetic algorithms, bit level using genetic algorithms and real coded genetic algorithm and toral automorphism. The histogram of the encrypted image is uniformly distributed and significantly different from the respective histograms of the original images. The cipher image histogram is completely horizontal, so if any statistical attackers attack, they will not be able to break the proposed security.
Figure 4.6 Original Image of Lena and Its RGB Histogram

Figure 4.7 Encrypted Image of Lena and Its RGB Histogram by Crossover and Mutation Operations of GAs
Figure 4.8 Original Image of Lena and Its RGB Histogram

Figure 4.9 Encrypted Image of Lena and Its RGB Histogram by bit level using GAs
Encryption and Decryption Image Using Multiobjective Soft Computing Algorithm

Figure 4.10 Original Image of Lena and Its RGB Histogram

Figure 4.11 Encrypted Image of Lena and Its RGB Histogram by real coded genetic algorithm and toral automorphism
4.7.2.2. Correlation Coefficient Analysis

The correlation coefficient displays the relationship between two neighbouring pixels. If correlation between two pixels is nearly 1, the image pixels are highly correlated but if it is nearly 0, the image pixels are highly uncorrelated. In the proposed method, the experiments on the images prove that the original image correlation is nearly one, and cipher image correlation is nearly zero. So, this saves the system from statistical attacks. Table 4.2 shows the results of experiments. The formula of correlation coefficient of two adjacent pixels is as follow:

$$C_r = \frac{N \sum_{j=1}^{N} (x_j y_j) - \sum_{j=1}^{N} x_j \sum_{j=1}^{N} y_j}{\sqrt{(N \sum_{j=1}^{N} x_j^2 - (\sum_{j=1}^{N} x_j)^2)(N \sum_{j=1}^{N} y_j^2 - (\sum_{j=1}^{N} y_j)^2)}}$$  \hspace{1cm} (4.11)

Where \(x\) and \(y\) are the values of two adjacent pixels in the image and \(N\) is the total number of adjacent pixels selected from the image.

The distribution of pixels of plane image and cipher image is shown in Table 4.2 – 4.4. In these tables horizontal, vertical and diagonal pixel distribution is presented for an image.

Table 4.2 Adjacent pixels correlation coefficients analysis by crossover and mutation operations of GAs

<table>
<thead>
<tr>
<th>Direction</th>
<th>Plain image of Lena</th>
<th>Cipher image of Lena</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Red</td>
<td>Green</td>
</tr>
<tr>
<td>Horizontal</td>
<td>0.9883</td>
<td>0.9775</td>
</tr>
<tr>
<td>Vertical</td>
<td>0.9824</td>
<td>0.9736</td>
</tr>
<tr>
<td>Diagonal</td>
<td>0.9736</td>
<td>0.9594</td>
</tr>
</tbody>
</table>

Table 4.3 Adjacent pixels correlation coefficients analysis by bit level using GAs

<table>
<thead>
<tr>
<th>Direction</th>
<th>Plain image of Lena</th>
<th>Cipher image of Lena</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Red</td>
<td>Green</td>
</tr>
<tr>
<td>Horizontal</td>
<td>0.9889</td>
<td>0.9782</td>
</tr>
<tr>
<td>Vertical</td>
<td>0.9836</td>
<td>0.9747</td>
</tr>
<tr>
<td>Diagonal</td>
<td>0.9752</td>
<td>0.9599</td>
</tr>
</tbody>
</table>
Table 4.4 Adjacent pixels correlation coefficients analysis by real coded genetic algorithm and toral automorphism

<table>
<thead>
<tr>
<th>Direction</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal</td>
<td>0.9894</td>
<td>0.9788</td>
<td>0.9779</td>
<td>-0.0173</td>
<td>-0.0099</td>
<td>-0.0270</td>
</tr>
<tr>
<td>Vertical</td>
<td>0.9843</td>
<td>0.9755</td>
<td>0.9696</td>
<td>-0.0091</td>
<td>0.0225</td>
<td>-0.0552</td>
</tr>
<tr>
<td>Diagonal</td>
<td>0.9758</td>
<td>0.9604</td>
<td>0.9557</td>
<td>-0.0086</td>
<td>0.0182</td>
<td>0.0190</td>
</tr>
</tbody>
</table>

4.7.3. Sensibility Analysis

4.7.3.1. NPCR and UACI Analysis

The NPCR is measures the change rate of the number of the cipher-image pixels when only one pixel of the plain-image is modified. The (UACI) means unified average changing intensity. Index measures the average intensity of differences between two images. Let us assume two ciphered images $C_1$ and $C_2$ whose corresponding plain images have only one-pixel difference. The gray-level values of ciphered images $C_1$ and $C_2$ at row $i$, column $j$ are labeled as $C_1(i,j)$ and $C_2(i,j)$ respectively. NPCR and UACI are defined as:

\[
NPCR = \frac{\sum_{i=1}^{W} \sum_{j=1}^{H} R(i,j)}{W \times H} \times 100\% \tag{4.12}
\]

\[
R(i,j) = \begin{cases} 
1 & C_1(i,j) \neq C_2(i,j) \\
0 & C_1(i,j) = C_2(i,j)
\end{cases}
\]

\[
UACI = \frac{\sum_{i=1}^{W} \sum_{j=1}^{H} \frac{|C_1(i,j) - C_2(i,j)|}{255}}{W \times H} \times 100\% \tag{4.13}
\]

Where $W$ and $H$ are the width and height of the image. In this example, the selected pixel is the last pixel of the plain-image. The NPCR and UACI at previous approaches are calculated and listed in Tables 4.5 - 4.7. The data show that the performance is satisfactory for the proposed approaches.
Table 4.5 NPCR and UACI by crossover and mutation operations of GAs

<table>
<thead>
<tr>
<th>Lena Image</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPCR</td>
<td>99.4983</td>
<td>99.5852</td>
<td>99.6760</td>
</tr>
<tr>
<td>UACI</td>
<td>0.1241</td>
<td>0.0262</td>
<td>0.0135</td>
</tr>
</tbody>
</table>

Table 4.6 NPCR and UACI by bit level using GAs

<table>
<thead>
<tr>
<th>Lena Image</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPCR</td>
<td>99.5358</td>
<td>99.5417</td>
<td>99.6820</td>
</tr>
<tr>
<td>UACI</td>
<td>0.1210</td>
<td>0.0233</td>
<td>0.0111</td>
</tr>
</tbody>
</table>

Table 4.7 NPCR and UACI by real coded genetic algorithm and toral automorphism

<table>
<thead>
<tr>
<th>Lena Image</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPCR</td>
<td>99.5904</td>
<td>99.9296</td>
<td>99.9842</td>
</tr>
<tr>
<td>UACI</td>
<td>0.0175</td>
<td>4.2705e-04</td>
<td>6.8942e-05</td>
</tr>
</tbody>
</table>

4.7.3.2. Peak Signal to Noise Ratio (PSNR) and Mean Square Error (MSE)

The peak signal-to-noise ratio (PSNR) and mean squared error (MSE) are two important metrics evaluation algorithms to measure image quality. These metrics are very simple and easy to use. PSNR is most easily defined via the mean square error (MSE) as:

$$PSNR = 10 \log_{10} \frac{255^2}{MSE}$$  \hspace{1cm} (4.14)

Where the mean square error (MSE) is defined as:

$$MSE = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} [I(i,j) - I'(i,j)]^2$$  \hspace{1cm} (4.15)

Where M and N are the number of pixels in the frame, I(i,j) and I'(i,j) are the number of pixels in the original and decrypted image, respectively. Greater PSNR value (>30dB) reveals better image quality. For encrypted image, smaller value of PSNR is
expected. Tables 4.8 – 4.10 show the results of PSNR and MSE for the proposed approaches based on genetic algorithm.

<table>
<thead>
<tr>
<th>Image</th>
<th>Lena</th>
<th>Bird</th>
<th>Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSNR</td>
<td>11.0624</td>
<td>11.4993</td>
<td>7.2987</td>
</tr>
<tr>
<td>MSE</td>
<td>5.0914e+03</td>
<td>4.6041e+03</td>
<td>1.2112e+04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lena Image</th>
<th>Lena</th>
<th>Bird</th>
<th>Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSNR</td>
<td>10.6394</td>
<td>11.1606</td>
<td>7.2595</td>
</tr>
<tr>
<td>MSE</td>
<td>5.6123e+04</td>
<td>4.9776e+03</td>
<td>1.2222e+04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lena Image</th>
<th>Lena</th>
<th>Bird</th>
<th>Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSNR</td>
<td>10.4586</td>
<td>10.6491</td>
<td>8.9791</td>
</tr>
<tr>
<td>MSE</td>
<td>5.8509e+03</td>
<td>5.5998e+03</td>
<td>3.2256e+03</td>
</tr>
</tbody>
</table>

### 4.7.4. Average Time of Encryption and Decryption

The executing time of encryption and decryption process by proposed approaches based on genetic algorithms has been implemented and conducted using Matlab-7.10.0 (R2010a) in AMD V 140 CPU @ 2.30 GHz with Windows-7 operating system. In this case, three separate images having the different type’s formats of JPG, TIF and BMP respectively and sizes 225 × 225 pixels, 194 × 259 pixels and 1153 × 2050 pixels have been used to measure the encryption time for each image by using the suggested approaches. The encryption time obtained using these images are given in tables 4.12. The average time of encryption is achieved by crossover and mutation operations of GAs, bit level using genetic algorithms and real coded genetic algorithm and toral automorphism while encrypting different images of different sizes and
formats. This shows that the crossover and mutation operations of GAs is 1.07 times less than bit level using genetic algorithms and 1.05 times less than the real coded genetic algorithm and toral automorphism. The average time of encryption and decryption image by using the previous approaches based on genetic algorithm are demonstrated in Table 4.11 and figure 4.12.

The fitness function bound are shown in figures 4.13 – 4.19.

Table 4.11 Average time of encryption and decryption by using genetic algorithms with three images different in size, type and formats

<table>
<thead>
<tr>
<th>Image / Approach</th>
<th>CO&amp;M</th>
<th>BL</th>
<th>RG&amp;TA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lena</td>
<td>12.051</td>
<td>12.051</td>
<td>13.024</td>
</tr>
<tr>
<td>Bird</td>
<td>11.760</td>
<td>11.760</td>
<td>12.542</td>
</tr>
<tr>
<td>Car</td>
<td>46.852</td>
<td>46.852</td>
<td>48.327</td>
</tr>
</tbody>
</table>

Figure 4.12 Average time of encryption and decryption images by using genetic algorithm
Figure 4.13 Fitness bound at 30
Figure 4.14 Fitness bound at 36
Figure 4.15 Fitness bound at 40
Figure 4.16 Fitness bound at 45
Figure 4.17 Fitness bound at 50
Figure 4.18 Fitness bound at 55
Figure 4.19 Fitness bound at 60
4.8. Conclusion

This chapter proposed three approaches of image encryption based on GAs. These approaches are described in section 4.6.1, 4.6.2 and 4.6.3. These approaches are used to produce a new encryption method by exploitation the powerful features of the crossover and mutation operations of genetic algorithms. The above approaches have been tested by different measuring analysis and showed their capability and quality for encrypting and decrypting images as demonstrated in tables and figures above.