Multidimensional Systems and Signal Processing, 16, 217–236, 2005

Speeding Up Fractal Image Compression by Genetic Algorithms

FARAOUN KAMEL MOHAMED Kamel_mh@yahoo.fr
Department of Computer Science, Evolutionary Engineering and Distributed Information Systems Laboratory, EEDIS, University of Sidi Bel-Abbès, Algeria

BOUKELIF AOUED aboukelif@yahoo.fr
Department of Electronics, Communications Networks, Architectures, and Multimedia Laboratory, University of Sidi Bel Abbès, Algeria

Received November 21, 2002; Revised April 23, 2004; Accepted July 1, 2004;
First online version published in May, 2005

Abstract. The main problem with all fractal compression implementation is execution time. Algorithms can spend hours to compress a single image. Most of the major variants of the standard algorithm for speeding up computation time have led to a bad-quality or a lower compression ratio. For example, the Fisher's [7] proposed classification pattern greatly accelerated the algorithm, but image quality was poor due to the search-space reduction imposed by the classification, which eleminates a lot of good solutions.

By using genetic algorithms to address the problem, we optimize the domain blocks search. We explore all domain blocks present in the image but not in exhaustive way (like a standard algorithm) and without omitting any possible block (solution) as a classification pattern does. A genetic algorithm is the unique method for satisfying these constraints. And it is a way to do be a random search because the genetic one is directed by fitness selection, which produces optimal solutions.

Our goal in this work is to use a genetic algorithm to solve the IFS inverse problem and to build a fractal compression algorithm based on the genetic optimization of a domain blocks search. we have also implemented standard Barnsley algorithm, the Y. Fisher based on classification, and the genetic compression algorithm with quadtree partitioning. A population of transformations was evolved for each range block, and the result is compared with the standard Barnsely algorithm and the Fisher algorithm = based classification.

We deduced an optimal set of values for the best parameters combination, and we can also specify the best combination for each desired criteria: best compression ratio, best image quality, or quick compression process. By running many test images, we experimentally found the following set of optimal values of all the algorithm parameters that ensure compromise between execution time and solutions optimality: Population size = 100, Maximum generations = 20, Crossover rate = 0.7, Mutation rate = 0.1, RMS limit = 5, Decomposition error limit = 10, Flips and isometrics count = 8.

In our proposed algorithm, results were much better than those obtained both vences and Rudomin [5] and Lankhorst [4] approaches.

1. Introduction

A major challenge of both theoretical and practical interest is the resolution of the inverse problem: finding an iterated function system (IFS) whose attractor is a target of two-dimensional (2D) shapes [1]. An exact solution can be found in some particular cases, but in general, no exact solution is known. Because the function to optimized is complex, most of them make some *a priori* restrictive hypotheses, such as use of affine IFS, with a fixed number of functions.

The major inconvenience of the current fractal compression algorithm is its high computational demands. To find existing redundancies (called *self-similarities* in fractal terms), this algorithm must perform many tests and comparisons between different areas of the compressed image. We cannot find easily similar parts in any natural images, so algorithm complexity is high, which lead to a slow compression process.

Genetic algorithms are used when we want to solve an optimization problem that is multimodal, is multidimensional and has a large search space with different optima [2]. Such problems do not have deterministic algorithms to get the global optimum, and if it exists, the algorithm is an exhaustive search along the solution space, which leads to consuming exponential time and machine resources. With NP-Hard problems, using a deterministic search space is impossible. The genetic algorithms traverse the search space using historical information to guess where to look for better search points. Algorithms of this kind are best suited for the problems described above, and their use to solve different complexes problem has proved their capacities. Both exploration of best solutions and exploration of the entire search space are ensured, and an appropriate optimal solution can be found in reasonable number of iterations.

The genetic algorithms are principally algorithms are principally destined for complex problems where no exact solution exists and an exhaustive search of the related search space leads to an Np-Hard problem, or high computation time [3]. Our goal is to accelerate the compression process by improving the standard compression algorithm with a genetic search technique.

This idea was exploited by some Authors in different ways [4] [5] because the optimization can be viewed from different angles and be applied on different parameters. Our approach is to use a genetic algorithm to optimize the search of similarities in the target image. The standard optimization methods are sufficient for the calculation of related parameters when the similarity is detected.

On the other hand, the fractal compression algorithm, or more precisely the IFS inverse problem resolution, can be considered to be a complex problem where no deterministic and definitive solutions exist. All proposed approaches try to find a compromise between time executions, reconstruction quality, and compression ratio. The exhaustive search proposed initially by Barnsley is time consuming [6] and can take from some hours to some days to compress a single image. The classification proposed by Fisher [7] accelerates the traditional algorithm, but image quality is poor due to the search space reduction, and the compromise is always difficult to find.

2. Fractal Compression Implementation

Different algorithms have been proposed to implement the fractal images compression, based on partitioned iterated function systems. The difference is generally in the way of partitioning images or in the metric used to compare domain and range blocs, but each algorithm follows the same steps. A general structure will be given in the following.

2.1. General Structure

A general structure for most proposed fractal compression algorithms, for both coding and decoding images, can be given by the following:

1. Encoding of an image I

- Set t = some tolerance level
- Partitioning I into uncovered ranges R_i.
- For each uncovered range R_i do

– Search over all D_i in the pool domains.
– If there's a w_i such that $d(R_i,w_i(D_i)) < t$, report w_i and compress it using adaptive arithmetic coding (or any other lossless compression scheme).
– split R_i into subranges and add them to the list of ranges to be covered.
– If the range can no longer be partitioned, return the minimum $d(R_i, w_i(D_i))$.
– Remove R_i from the uncovered list.

2. Decoding of a map $w = \cup\, w_i$

- Choose any image I_0, and then compute the image $w^n(I_0) = \cup\, W^n{}_i(I_0)$. When n is big enough, $w^n(I_0) \approx I_w \approx I$.

2.2. Fractal Coder with Standard algorithm

Algorithm standard-compression (Input I: 256 × 256 gray-scale image, Output W: Coded IFS)

```
1. Regularly decompose target image into ranges a blocks with B × B size
   (B = 4, 8, or 16)

2. **For** each range block $R$ in Ranges ($I$) **do**
       **For** each domain $D$ block in Dom ($I$) **do**
```

```
            Contract the block $D$;
            Compare $T(D)$ to $R$ within RMS
            Error and extract $s$ and $o$ parameters;
            **If** $RMS <$ Predefined-limit, **then**
                Write transformation to the output W;
                Go to 2;
            **Else** Choose the next possible domain block;
            **End If**;
        **End For**;
    **End For**.
```

Ranges (I) is the set of all ranges blocks obtained by the partitioning pattern, Dom(I) is the set of all possible domain blocks that can be obtained from the image according to a ranges blocks size.

The transformation parameters obtained for each block are quantified and coded on a fix number of bits. For each transformation, we have the following parameters:

- X_{dom} and Y_{Dom} coordinate of the domain block, coded on 8 bits each one.
- The scaling factor S: belongs to $[-1, +1]$ to ensure contraction of the transformation and can be quantified and coded on 5 bits (determined experimentally). The quantification formula is given by $S_{\text{code}} = \text{round}(0.5 + \text{S})*(2^{\text{sbits-1}})$.
- The contrast factor O ranging from 0 to 255, can be quantified and coded on only 7 bits (optimal quantification)by the formula. $O_{\text{code}} = \text{round}$ $(0.5 + O/(1 + |S|))*(2^{\text{obits}-1})$.
- The isometric flip parameters take its values between 0 and 7 for 8 possible flips and rotations. So it can be coded on 3 bits.

So we can code a transformation with all its parameters on $8+8+5+7+3=31$ bits. The decoding process is done by applying each transformation from the IFS to an initial random image I^0 after a sufficient number of iterations the reconstructed image will approximate the original one. For each pixel in I^0 the transformation is applied like the following:

$$I^1[i,j] := S * I^0[i,j] + O(i : l \dots L, j : 1 \dots H)$$

L and H are the image dimensions.

3. The Standard Genetic Algorithm

Pseudocode for the algorithm is given in Figure 1.

```
Procedure GA
Begin
        For $i$ := 1 to $n$ do begin
          Initialize $x_i$ ( * with random values *)
          $f(x_i)$:= evaluate($x_i$) (* determine fitness *)
        End
        While termination condition not satisfied do
             Begin
             $p_{1...n}$ := select from $x_{1...n}$ in random order
             For $i$ := 1 to $n$ step 2 do begin
             $x_i^l$, $x_{i+l}^l$: = crossover($p_i$ , $p_{i+1}$)
             mutate($x_i^l$)
             mutate($x_{i+l}^l$)
             End
             $x_{1...n}$ := $x_{1...n}^l$ (* copy new population *)
             For $i$ := 1 to $n$ do begin
                            $f(x_i)$ := evaluate($x_i$)
                            End
             End {while}
End
```

Figure 1. The standard genetic algorithm.

4. Genetic Algorithms and the IFS Inverse Problem

Genetic algorithms work with a population of individuals who are iteratively adapted toward the optimum by means of a random process of selection, recombination, and mutation. During this process, a fitness function measures the quality of the population, and selection favors those individuals of higher quality. Most of the evolutionary algorithms described in the literature for solving the IFS inverse problem follow the optimization problem. In this case, each individual is an IFS model consisting of a number of transformations, and its fitness is given by some convenient measure of similarity between the target image and the IFS attractor.

To generate the IFS code of a given image by the use of genetic algorithms, two different approaches of representation can be considered:

- Consider the whole IFS of the coded image as an individual, and then iterate the genetic algorithm on a population of IFS. Each IFS is constituted by a fix number of transformations (depending on the partition pattern) as genes.
- For each range bloc, we associate a population of transformation as individuals. Each transformation (individual) is represented by its parameters as genes.

4.1. Our Algorithm: Improving the Standard method with Genetic Algorithms

The fractal compression scheme for a single image can be seen in the following algorithm.

```
/* Genetic algorithm for FIC */

1. P ← Generate (LIFS) randomly
2. For all LIFS $p_i \in$ P do

   Evaluate fitness by applying ($p_i$) to generate an image measuring its distance
   (using the $L^1$ or $L^2$ metric) to the original image

3. While termination criteria not met Do

   Reproduce $p_i \in$ P according to evaluation
   Apply the desired mutation operator
   Apply the desired mating operator
   Evaluate new LIFS (as above)
   Replace the worst old strings with the best new strings.
   EndDo
```

4.2. The Fitness Function

In the case of IFS, the measure of quality for a given transformation is given by the RMS error between the coded range block, and the domain block determined by the transformation coordinates X_{dom} and Y_{dom}, and is transformed with corresponding luminance contrast values. This error is calculated using the root mean square equation. The fitness function is defined by the value of this error, which is inversely proportional to the efficiency of the corresponding individual.

4.3. Chromosomes Codification

A chromosome in our algorithm is constituted by five genes, from which only three genes are submitted to genetic modification, the two others are computed by the RMS equation. We have the following genes:

- $X_{\mathrm{dom}}, Y_{\mathrm{dom}}$, flip, which are optimized by a genetic search.
- Contrast O, and scaling S, which which are computed directly by RMS equation.

This will improve both compression speed and reconstruction quality. Figure 2 shows our chromosomal representation of the IFS:

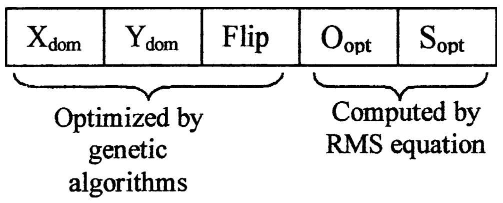

Figure 2. Choromosomal representation of individual IFS.

4.4. Genetic Operators

In every genetic algorithm, some principal operators must be defined. They operate on population's individuals to produce new offspring individuals and to improve fitness function values in the next generation.

The two principal operators used in our implementation are *crossover operator* and *mutation operator*. Their patterns and structures are presented in the following.

4.4.1. The Crossover Operator

The crossover operator combines two individuals in the current population to produce two offspring individuals included in the new generation. The main role of this operator is to create good new solutions based on the characteristics of the parents. To perform this operation, individuals from the current population are chosen randomly and proportionally to their fitness value. We apply this operator with a probability value fixed as a parameter of the algorithm Experimental results have shown that a value of 0.7 is good to ensure quick convergence of the algorithm.

We introduced a new pattern of crossover, and we used it in our implementation and obtained good results. The result coordinates for the offspring individuals are obtained by a linear combination of the parents' coordinates. A random number a is generated in the interval [0,1]. Then the new coordinates are calculated according to the following formula:

For the first offspring,

$$X_{\mathrm{dom}} = a^* X^1_{\mathrm{dom}} + (1-a)^* X^2_{\mathrm{dom}}$$

$$Y_{\mathrm{dom}} = a^* Y^1_{\mathrm{dom}} + (1-a)^* Y^2_{\mathrm{dom}}$$

For the second offspring,

$$X_{\mathrm{dom}} = (1-a)^* X^1_{\mathrm{dom}} + (1-a)^* X^2_{\mathrm{dom}}$$

$$Y_{\mathrm{dom}} = (1-a)^* Y^1_{\mathrm{dom}} + (1-a)^* Y^2_{\mathrm{dom}}$$

For the flip gene, we randomly affect a parent's value to each offspring individuals. This crossover pattern is very efficient. It allows us to explore all the image area if the two parents are in separated regions and to explore the nearest neighborhood if they are in the same region. So we lead to a good exploitation of the search space. Figure 3 illustrates the described crossover pattern.

4.4.2. Mutation Operator

Mutation operator is used in all implementations of genetic algorithms to introduce diversity in each population. Mutation is applied to individuals by changing pieces of their representations. These individuals are randomly selected according to their fitness value. At each position in the individual, we decide randomly, using a small mutation probability, whether the gene at this position is to be changed. This genetic operator allows the algorithm to explore new areas of the search space and find new possible optimal solution.

In our implementation, a mutation operator is applied to some selected IFSs from the current population. One of the three genes $X_{\text{dom}}, Y_{\text{dom}}$, and flip is selected randomly and changed with a random generated value. Figure 4 shows the general pattern of the mutation operator in our algorithm.

4.5. Termination Criteria

Any genetic algorithm must find the optimal solution for a given problem in a finite number of steps. In our implementation, two criteria can cause the termination of the algorithm when applied to a given range block:

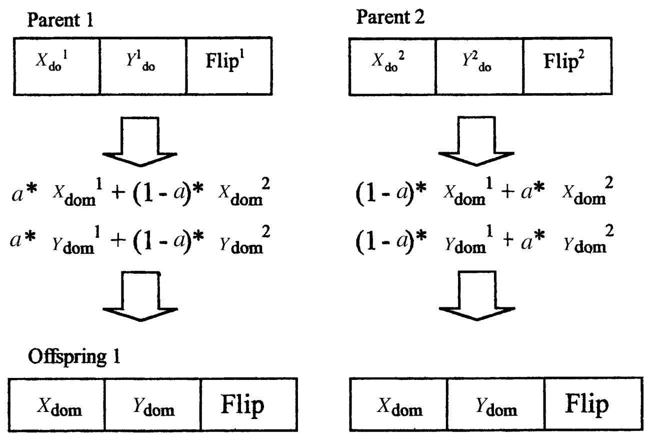
Parent 1
X_{do}^1
Y^1_{do}
Flip^1
Parent 2
X_{do}^2
Y^2_{do}
Flip^2
$a^* \; X_{\text{dom}}^1 + (1-a)^* \; X_{\text{dom}}^2$
$a^* \; Y_{\text{dom}}^1 + (1-a)^* \; Y_{\text{dom}}^2$
$(1-a)^* \; X_{\text{dom}}^1 + a^* \; X_{\text{dom}}^2$
$(1-a)^* \; Y_{\text{dom}}^1 + a^* \; Y_{\text{dom}}^2$
Offspring 1
X_{dom}
Y_{dom}
Flip
X_{dom}
Y_{dom}
Flip

Figure 3. The crossover operator pattern.

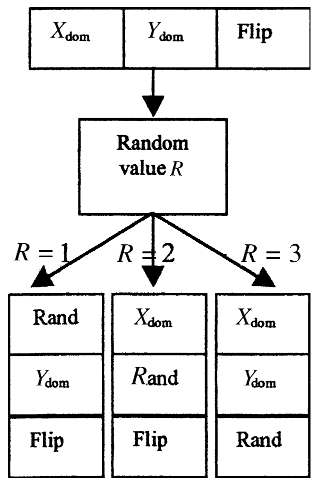

Figure 4. Mutation operator schema.

- An acceptable value of fitness for the best in individual in the population is reached;
- A maximum predefined count of generations is reached.

This maximum count is a predefined parameter of the algorithm; it was determined experimentally and fixed to 20 generations in our implementation.

4.6. The Parameters of the Algorithm

The behavior of the genetic algorithm can be controlled using many initial conditions and parameters. We can control convergence speed, solution quality, and algorithm evolution when adjusting and modifying these parameters. In our algorithm we have two different sets of parameters; those that control genetic evolution, and those that control the fractal compression pattern. The set of parameters of genetic control is given by the following

- *Population size*: Specify the number of individuals in each generated population (constant duraing all steps).
- *Crossover rate*: Specify the probability used to select individuals submitted to crossover operator.
- *Mutation rate*: Specify the probability used to select individuals submitted to mutation operator.
- *Maximum generations*: Specify the maximum number of generations to evaluate before assuming that a particular run has converged prematurely, and should be aborted.

The second set of parameters, which control and specify the fractal compression pattern, is equivalent to the one used in standard compression algorithm. It controls the partition size, the decomposition pattern, and the transformations quantification parameters, We have

- *The range block decomposition size* (used with regular partitioning);
- *The lowest block size used for ranges decomposition* (in the case of the Quadtree pattern);
- *The number of flips and isometrics* applied to each domain block to be compared with the current range block;
- *The decomposition error limit*, introduced to improve the Quadtree decomposition pattern, as above mentioned in the Quadtree algorithm descriptions;
- *The RMS error limit*, which is fixed to decide if a given transformation is accepted, (The fitness value of the best individual in each population is compared to this value, and if it is lower, the individual is selected as the optimal solutions, and the algorithm is stopped);
- *The number of bits used to quantify and code* luminance an contrast parameters, fixed experimentally to 5 and 7 bits, respectively.

Different combinations of these parameters were tested, and different results were obtained. For the genetic parameters, the mutation rate has generally a small value, but the crossover rate is generally greater. A high value for the mutation rate will lead to a Quasi-random population, and the algorithm becomes similar to a random walk in the search space. The population size is a critic parameter. Large sizes give fever numbers of generations, but the computation time is high, Small sizes give relatively slow convergence but a reduced computation time for each generation. Both population size and maximum generation count were determined experimentally and according to the obtained results for each values combination. In Table 1, the set of optimal values of all the seven algorithm parameters is given. These values ensure compromise between execution time and solutions optimality.

For the Quadtree decomposition, we set the lowest blocks size to 4×4 pixel to achieve highest reconstruction quality and to 8×8 to obtain acceptable quality with a high compression ratio.

Table 1. Optimal parameters of our genetic compression algorithm.

Population size	100
Maximum generations	20
Crossover rate	From 0.7 to 0.8
Mutation rate	0.1
RMS limit	5.0
Decomposition error limit	10.0
Flips and isometrics count	8

4.7. The Structure of the Algorithm

Based on all these elements, the genetic compression algorithm operate on an input image according to the following general steps:

Algorithm Genetic-Compression (Input I 256×256 gray-scale image, Output, W: Coded IFS)

```
1. Decompose the input image into range blocks according to the used partitioning
   pattern;
2.  For each block $R$ in Ranges($I$) do
      Generate a random population of chromosomes (transformations);
      While (No optimal domain block is found)
        and (population maximum is not reached) do

 • Compute fitness value for all individuals;
 • Apply crossover operator on population individuals (selected with the cross-
   over rate);
 • Apply Mutation operator on population individuals (selected with the cross-
   over rate);
 • Generate the new population;

        End Do;
      End While;
    Write obtained transformation parameters to the output W;
    End For;
```

5. Experimental Results

All tests were executed on a PIII-INTEL 800 MHz with 128 M0 of RAM size.

5.1. Standard Algorithm with Quadtree Partitioning (Table 2)

Table 2. Result for different images with Quadtree blocks decomposition.

Image	RMS limit	Execution time	Quality (dB)	Compression ratio
Lena	5.0	1 h 11 m 10 s	32.01	11.48:1
Boat	5.0	1 h 27 m 04 s	29.56	8.88:1
Peppers	5.0	1 h 5 m 17 s	33.07	24.56:1
Barb	5.0	1 h 22 m15 s	30.07	10.85:1

5.2. *Standard Algorithm Improved with Classification (Y. Fisher Approach)* (Table 3)

Table 3. Result for different images with Quadtree blocks decomposition using Fisher's classification algorithm.

Image	RMS limit	Execution time	Quality (dB)	Compression ratio
Lena	5.0	2 min 55 s	30.05	9.73:1
Boat	5.0	3 min 22 s	25.86	7.9:1
Peppers	5.0	2 min 45 s	29.36	10.16:1
Barb	5.0	4 min 12 s	21.07	8.62:1

5.3. *Genetic Algorithm with Quadtree Decomposition: The Best Solution*

The genetic compression algorithm was used with Quadtree partitioning. Different parameters were used for each test, and the obtained results are given in both table forms and graphical forms. Examples of reconstructed images are also given to illustrate reconstruction quality.

Table 4 shows different performances with different values of RMS error limit using fixed value for other parameters: population size = 100, mutations rate = 0.1, crossover rate = 0.7 and maximum generations count = 20.

5.3.1. *Effect of Parameters Controlling Fractal Compression*

We can see from the Figures 5 and 6 that image quality is inversely proportionate to RMS error limit. And compression rate is proportionate to that value. The compromise value acceptable of this parameter is 5.0, and it gives performances. The used images are Lena and Goldhil. The decomposition level error was equal to RMS limit to get the real influence of this parameter (Figures 7 and 8).

Table 4. Different compression results of Lena image while applying different values of RMS error limit.

RMS limit	Execution time	Quality (dB)	Ratio	Ranges count
0.0	2 m44 s	35.66	4.29:1	4069
2.0	1 m56 s	35.03	6.35:1	2770
4.0	49 s	34.89	9.28:1	2023
5.0	43 s	34.80	9.82:1	1792
8.0	36 s	34.50	9.95:1	1768
10.0	33 s	30.50	10.05:1	1750
15.0	21 s	22.33	13.66:1	1288
20.0	14 s	19.36	19.34:1	910
25.0	15 s	19.01	26.25:1	670

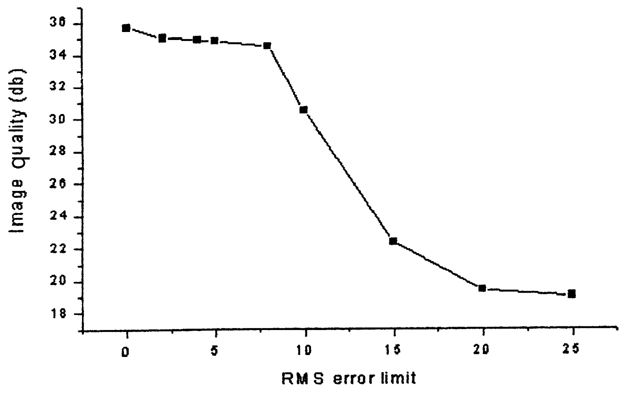

Figure 5. Lena image quality variation according to RMS limit values.

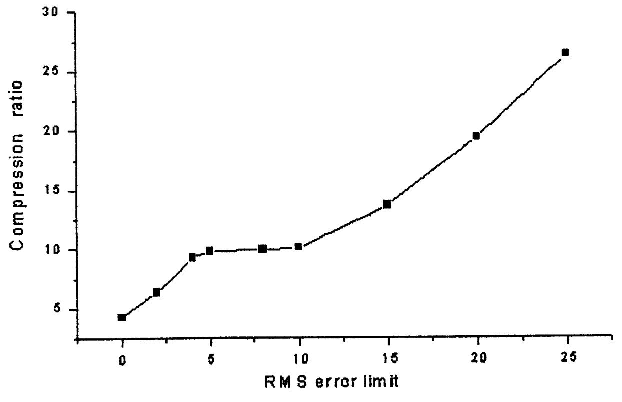

Figure 6. Lena image compression rate variation according to RMS limit values.

5.3.2. Effect of Parameter Controlling Genetic Evolution

The second parameter to be tested is the population size, Results obtained with different sizes of Population are illustrated in Table 5. The best value for this parameter is 100 individual. It is a good compromise between execution speed and image quality. The same work has been done with both crossover rate and mutation rate. These parameters are less important than RMS error or population size, but crossover rate must always be greater than mutation one to ensure the best selection of optimal solutions. All the obtained results are summarized in the Tables 6, 7 and Figures 9–15.

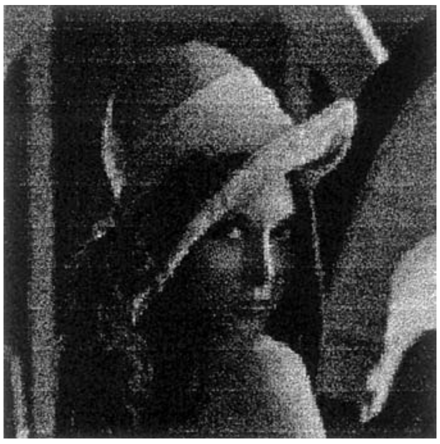

Figure 7. Decomposed Lena image with RMS = 5.0 (ratio 9.14:1).

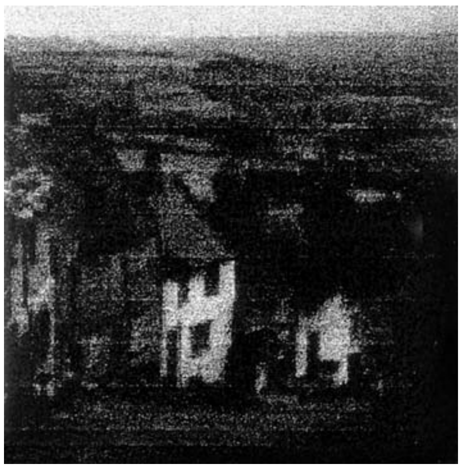

Figure 8. Decomposed Gold Hill image with RMS = 5.0 (ratio 9.14:1).

Table 5. Different compression results of Lena image while applying different values of Population size.

Population size	Execution time	Quality (dB)	Ratio	Ranges count
5	9 s	29.62	8.30:1	2119
10	11 s	29.98	8.35:1	2107
20	14 s	30.21	8.72:1	2017
50	23 s	32.11	9.36:1	1879
100	44 s	32.23	9.83:1	1789
250	2 m 24 s	33.74	10.35:1	1699
500	7 m 4 s	34.56	10.83:1	1624
1000	23 m 4 s	35.12	10.97:1	1603

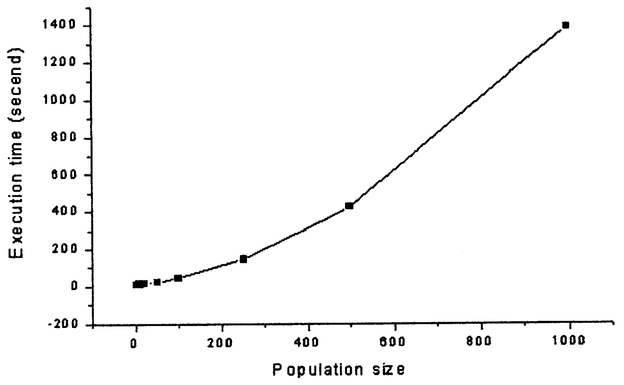

Figure 9. Lena image compression time variation according to population size values.

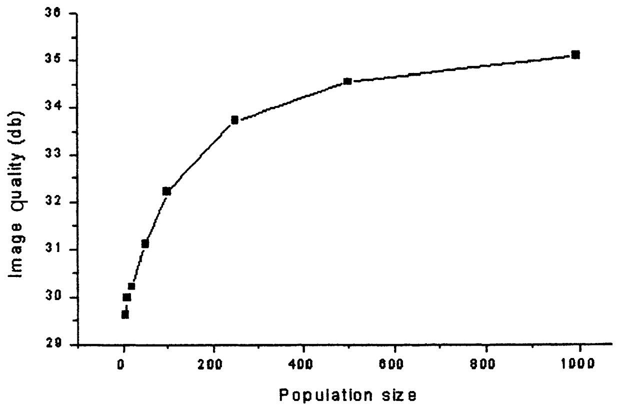

Figure 10. Lena image quality variation according to population size values.

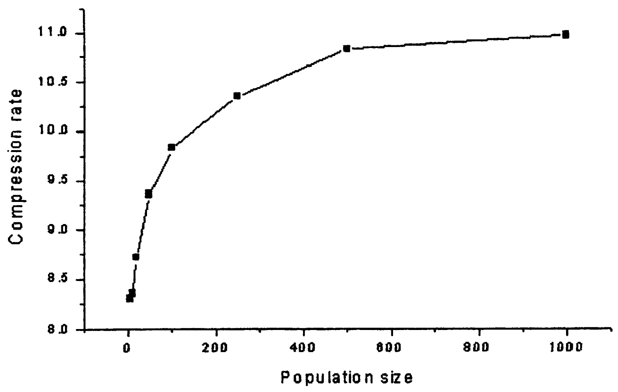

Figure 11. Lena image compression ratio variation according to population size values.

Table 6. Different compression results of Lena image while applying different values of crossover rate.

Crossover rate	Execution time	Quality (dB)	Compression tatio	Ranges count
0.2	35 s	31.62	7.9:1	2227
0.3	51 s	30.96	8.08:1	2176
0.4	52 s	30.01	8.13:1	2164
0.5	52 s	30.08	8.13:1	2164
0.6	55 s	30.12	8.18:1	2151
0.7	56 s	29.62	8.24:1	2135
0.8	1 m 3 s	31.06	8.15:1	2158
0.9	1 m 22 s	31.11	8.06:1	2182

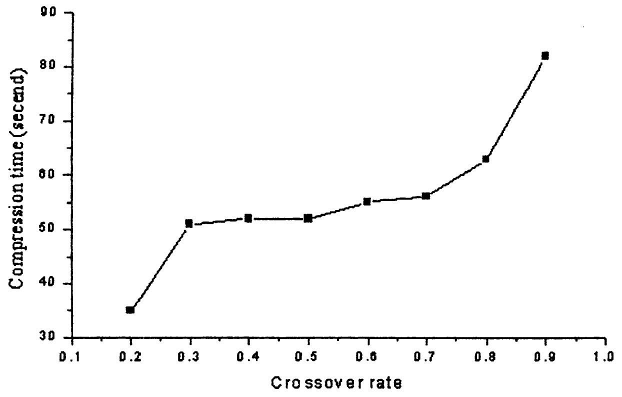

Figure 12. Boat image compression time variation according to crossover rate values.

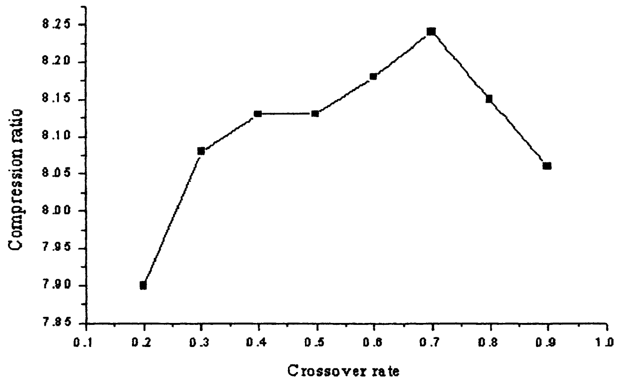

Figure 13. Boat image compression ratio variation according to crossover rate values.

Table 7. Different compression results of Boat image while applying different values of mutation rate.

Mutation rate	Execution time	Quality (dB)	Compression tatio	Ranges count
0.1	56 s	29.62	8.24:1	2135
0.2	58 s	29.65	8.19:1	2148
0.3	1 m 10 s	29.60	7.80:1	1254
0.4	1 m 12 s	29.12	8.24:1	2135
0.5	1 m 10 s	29.10	8.15:1	2159
0.6	1 m 12 s	29.11	8.14:1	2162

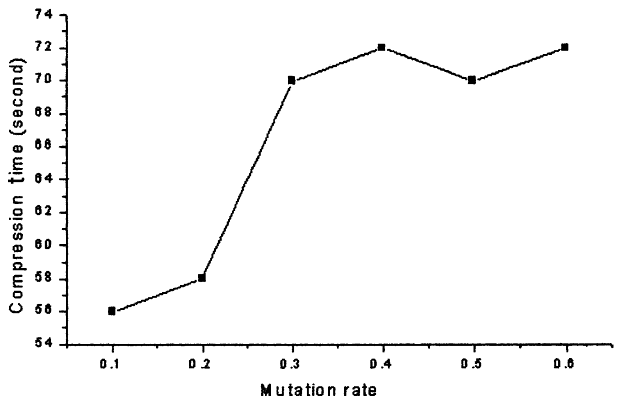

Figure 14. Boat image compression time variation according to mutation rate values.

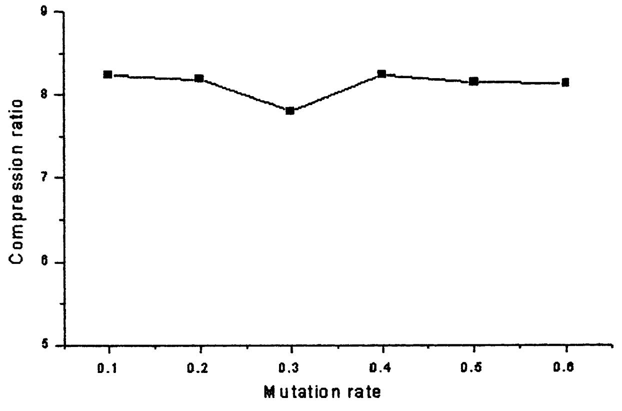

Figure 15. Boat image compression ratio variation according to mutation rate values.

6. Optimal Parameter Values

From the previous presented results, we can deduce an optimal set of values for the best parameters combination, and we can also specify the best combination for each desired criteria: best compression ratio, best image quality, or quick compression process. The following tables summarize different combinations obtained from to the previous tests. The compromise is always difficult to find. A global comparison of the three presented compression algorithms for which best results are given in the Tables 8, 9, and 10. Table 11 gives optimal parameters to use for faster compression speed. The last table (Table 12) give a summary of the obtained results on different images using different implemented algorithms.

Table 8. Optimal compromise parameters for the genetic compression algorithm.

Population size	100
Maximum generations	20
Crossover rate	0.7
Mutation rate	0.1
RMS limit	5.0
Decomposition error limit	10.0
Flips and isometrics count	8

Table 9. Parameters for the genetic compression algorithm which give best quality.

Population size	1000
Maximum generations	20
Crossover rate	0.8
Mutation rate	0.1
RMS limit	2.0
Decomposition error limit	5.0
Flips and isometrics count	8

Table 10. Parameters for the genetic compression algorithm which give best ratio.

Population size	500
Maximum generations	20
Crossover rate	0.8
Mutation rate	0.1
RMS limit	10.0
Decomposition error limit	20.0
Flips and isometrics count	8

Table 11. Parameters for the genetic compression algorithm which give faster compression.

Population size	50
Maximum generations	20
Crossover rate	0.8
Mutation rate	0.1
RMS limit	10.0
Decomposition error limit	20.0
Flips and isometrics count	8

7. Conclusion

From the presented results, we can see that the genetic search used to enhance the capabilities of the fractal compression schema has give better results than both standard algorithms proposed by Barnsley and the classification scheme used by Fisher, Table 12 shows that compression speed is very different (e.g., for the Lena image 1:10:11 with the standard algorithm, 2:55 with classification, and 42 s with the genetic algorithm), and this acceleration has not degrade the quality of the reconstructed image or the compression ratio. So the major important goal (the acceleration) is achieved. Furthermore, the different combinations of the genetic parameters can give the compression algorithms more flexibility and adatability to the different constraints of the compression.

The works presented in [4] and [5] are applied on binary images, and they can be considered as shape approximation methods. In their approach, the IFS is considered as a chromosome, and each transformation w_i is a gene, so the genetic

Table 12. Results obtained on different used images using Quadtree partitioning pattern.

Image	Rate	Quality (dB)	Time
Standard algorithm			
Lena	11.48 :1	32.01	1:10:11
Boat	8.88 :1	26.56	1:27:04
Barb	10.85 :1	27.07	1:22:15
Peppers	24.56 :1	33.07	1:05:17
Genetic algorithm			
Lena	9.88 :1	33.25	42 s
Boat	8.17 :1	28.05	50 s
Barb	8.49 :1	28.99	52 s
Peppers	10.16:1	30.24	45 s
Classification algorithm			
Lena	9.73 :1	30.05	2 m 55 s
Boat	7.9 :1	25.86	3 m 22 s
Barb	8.62 :1	29.65	4 m 12 s
Peppers	10.16 :1	29.36	2 m 45 s

convergence is not always possible, and exploration of the search space is more difficult. Our algorithm can be extended to colored images by applying the same principe to each color map (blue, red, and green).

8. Perspectives

Here, for a fixed size square block partition, a fractal code is required as in standard fractal coding, but for each range the best D codebook entries are kept in a list together with the optimal scaling and offsets parameters. We take N times this configuration as the starting population for the evolution. The offspring are built by randomly merging two neighboring blocks. The fractal code is modified by considering only the transformations kept in the lists of those two blocks. A selection is performed by keeping only the fittest configurations interms of collage error.

References

1. M.F. Barnsley and S. Demko, "Iterated Function Systems and the Global Construction of Fractals," *Proceedings of the Royal Society of London A*, vol. 399, 1985, pp. 243–275.
2. D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine, Learning, Reading*, Reading, MA: Addison-wesley, 1989.
3. L. Davis, *Handbook of Genetic Algorithms*, New York: Van Nostrand Reinhold, 1991.
4. Marc M. Lankhors, "Iterated Function Systems Optimization with Genetic Algorithms", University of Groningen Department of Computing Sciences. January 11, 1995.
5. L. Vences and I. Rudomin, "Fractal Compression of Single Images Sequences Using Genetic Algorithms", Institute of Technology, University of Monterrey, 1994. Available from ftp://ftp.informatik.uni-freiburg.de/papers/fractal/VeRu94.ps.gz.
6. M. Barnsely and L. Hurd, *Fractal Image Compression*, Wellesley, MA: AK Peters, 1993.
7. Y. Fisher, Fractal Image Compression: Theory and Application, New York: Springer-Verlag.