An Improved
Fractal Image Compression Approach
by Using Iterated Function System
and Genetic Algorithm

YANG ZHENG
School of Computer and Information Science, Shanghai University of Electric Power
Shanghai, 200090, P.R. China
qfeng.1114@sina.com

GUANRONG LIU AND XIAOXIAO NIU
School of Computer Science and Technology, Wuhan University of Technology
Wuhan, 430070, P.R. China

Abstract—This paper presents an improved method of generating a binary image affine IFS
(iterated function system) by using genetic algorithm. We adopt a natural variable-length genotype
encoding to represent an individual. The multiobject fitness function is also applied in this algorithm.
In addition, a distributed version of the binary image compression algorithm is implemented. Both
theoretical analysis and experimental results show a higher compression ratio with better quality
images by using the proposed algorithm.

Keywords—Fractal image compression, Iterated function system (IFS), Genetic algorithms, Dis-
tributed, Parallel.

1. INTRODUCTION

Fractals are geometric patterns that are self-similar and can be used especially in computer
modeling of irregular patterns and natural phenomena. Hutchinson in his paper [1] developed
the idea using the mathematical theory known as iterated function systems (IFS) [2]. IFS were
later successfully used in modeling natural patterns such as clouds, leaves and trees, and fractal
image compression. Barnsley and his research group realized the potential of iterated function
systems and have indicated that IFS can be used for fractal image compression. They developed
an interactive system for the generation and solution of inverse problem for IFS [3–6]. Jacquin,
a graduate student of Barnsley, has completed an automatic fractal encoding system scheme in
his dissertation and proposed a new theory called local or partitioned iterated function system
(PIFS) [7].

Fractal image compression has been a popular technique for achieving high compression ratios.
An increasing number of research work with applications appeared in this area. Fractal objects
are objects whose geometry is generally a result of their self-similar structure [8] and can be
zoomed infinitely. Under the condition of satisfying a certain limit, the target of compression is
to acquire the simplest description of the image, which is regarded as an array of attractors of contraction maps on a complete metric space. Recently, new research in seeking efficient iterated function systems focuses specifically on wavelet fractal, selections and optimization of fractal definition domain, heuristic fractal, and IFS segment. With new developed methodologies, the evolutionary algorithm has emerged as a solution for fractal image compression.

The evolutionary computation is a parallel solution method which uses the idea of and gets inspirations from the natural evolutionary process [9]. Due to its intrinsic parallelism and some intelligent properties such as adaptation, self-organizing, and self-learning, the genetic algorithm (GA) and more generally the evolutionary algorithm (EA) are currently efficient stochastic optimization tools, and are widely used in various application fields. Based on the idea of the genetic algorithm, we construct a search match algorithm, which is known as the genetic search method. With the characteristics of the fractal image compression, we adopt a variable-length encoding for individual gene, and propose a new selection scheme about fitness function and location of crossover mutation. Experimental results have shown that our method yields a better compression ratio and improved fidelity, and overcame efficiently some disadvantages of the traditional search method. The genetic algorithm itself poses intrinsic parallel properties suitable for parallel computers on a larger scale. This paper explains the idea of a distributed GA using the proposed sequential algorithm. Finally a conclusion and future work are discussed.

2. THEORETICAL FOUNDATIONS OF FRACTAL IMAGE COMPRESSION

The fractal image compression algorithm is based on the fractal theory of self-similar and self-affine transformation. In this paper, the basic theory involved in fractal image compression is restricted to complete metric spaces which can be represented, for convenience, by the region \( I^2 = [0, 1]^2 \). Some basic definitions, following those given in [4], are listed here.

(1) Iterated Function System Definition

An iterated function system consists of a complete metric space \((X, d)\) together with a finite set of contraction mappings \( w_i : X \rightarrow X \) where \( i = 1, 2, \ldots, N \), with the corresponding contraction factors \( c_i \). The notation for IFS is \( \{X; w_1, w_2, \ldots, w_N\} \) and its contraction factor is \( C = \max_{1 \leq i \leq N} \{c_i\} \).

(2) The IFS Condensation Theorem

Let \( \{X; w_1, w_2, \ldots, w_N : C \in [0, 1]\} \) be a hyperbolic IFS \((X)\) is typically \([0, 1]\) or \([0, 1]^2\). Then the transformation \( W : H(X) \rightarrow H(X) \) defined by

\[
W(B) = \bigcup_{i=1}^{N} w_i(B), \quad \forall A \in H(X),
\]

is a contraction mapping on the complete metric space \((H(X), h)\), with contraction factor \( C \), where \( h \) is known as the Hausdorff metric. In other words,

\[
h(W(A), W(B)) \leq C \cdot h(A, B), \quad \forall A, B \in H(X).
\]

There exists a unique fixed point \( A \in H(X) \) which satisfies

\[
A = W(A) = \bigcup_{i=1}^{N} w_i(A),
\]
and is known as an attractor of the IFS. The fixed point may be obtained by an iterative scheme such as the one below, i.e.,

\[ A = \lim_{n \to \infty} W^{(n)}(B), \quad \forall B \in H(X), \]

where \( W^{(n)}(B) = W(W^{(n-1)}(B)) \) with the initial guess being chosen as \( W^{(0)}(B) = B \).

\( (a) \) (b)

Figure 1. Approximating a maple leaf based on the collage theorem (see (b)).

(3) The Collage Theorem

Let \((X, d)\) be a complete metric space and \(\{X; w_1, w_2, \ldots, w_N : C \in [0, 1]\}\) be an IFS with the attractor \(A\). Let \(\varepsilon \geq 0\) be a given real number. Suppose for any \(L \in X\),

\[ h(F, A) \leq h(L, \bigcup_{i=1}^{N} w_i(L)) / (1 - C) \leq \varepsilon / (1 - C), \quad \forall F \in H(X). \]

An example is shown in Figure 1 which approximates a maple leaf (see (b)) using the collage theorem by using four IFS maps (see (a)). It is easily seen that the maple leaf is viewed as an approximate union of shrunken copies of itself in different orientations.

2.1. Affine Transformation in \(I^2\)

Consider the IFS \(\{I^2; w_1, w_2, \ldots, w_N\}\), with \(I^2 = [0, 1]^2\), where each \(w_i\) is defined as the affine transformation as follows.

\[ w_i \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} = Ax + T. \quad (1) \]

Here

\[ x = \begin{bmatrix} x \\ y \end{bmatrix} \in I^2 \]

and the coefficients of \(A\) and the components of \(T\) are real numbers. The coefficients of the diagonal of \(A\) stretch, shrink, and mirror a given point horizontally or vertically, the skew diagonals of \(A\) are used for skewing and rotation, and the vector \(T\) is used for translating the given point \([10]\).
Figure 2. A spleenwort fern.

For instance, using equation (1), we can construct interesting fractals such as the famous spleenwort fern (see Figure 2). The IFS \( \{I^2; w_1, w_2, \ldots, w_N\} \) can be expressed as shown in Table 1.

In a random iterative algorithm, the value of \( p_i \) can be taken to be

\[
p_i \approx \frac{|a_id_i - b_ic_i|}{\sum_{i=1}^{N} |a_id_i - b_ic_i|}, \quad i = 1, 2, \ldots, N,
\]

where \( N = 4 \) in Table 1 is the number of maps. Other situations may be treated empirically (and for convenience \( p_i = 0.25 \)). In the previous exciting episode, this fern can be described completely in terms of only \( 25 (= 4 \times 6 + 1) \) IFS parameters.

According to the collage theorem, this paper deals with fractals, which are binary image (represented, for convenience, by the region \( I^2 = [0, 1]^2 \)). There is an associated grey-level function \( g(x, y) \), which may assume a finite nonnegative value. From the point of view of continuous binary image \( B \), each pixel can assume only two discrete values with 0 or 1 (white and black are represented 0, 1, respectively). The parameters in Table 2 ensure every affine transformation is generated automatically and is contractive.

### Table 1. IFS code for a spleenwort fern.

<table>
<thead>
<tr>
<th>( W )</th>
<th>( a )</th>
<th>( B )</th>
<th>( c )</th>
<th>( d )</th>
<th>( e )</th>
<th>( F )</th>
<th>( p )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.16</td>
<td>0</td>
<td>0</td>
<td>0.25</td>
</tr>
<tr>
<td>2</td>
<td>0.85</td>
<td>0.04</td>
<td>-0.04</td>
<td>0.85</td>
<td>0</td>
<td>1.6</td>
<td>0.25</td>
</tr>
<tr>
<td>3</td>
<td>0.2</td>
<td>-0.26</td>
<td>0.23</td>
<td>0.22</td>
<td>0</td>
<td>1.6</td>
<td>0.25</td>
</tr>
<tr>
<td>4</td>
<td>-0.15</td>
<td>0.28</td>
<td>0.26</td>
<td>0.24</td>
<td>0</td>
<td>0.44</td>
<td>0.25</td>
</tr>
</tbody>
</table>

### Table 2. Parameters satisfy the conditions.

<table>
<thead>
<tr>
<th>( w(0, 0) \in I^2 )</th>
<th>( 0 \leq e \leq 1 )</th>
<th>( 0 \leq f \leq 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w(1, 0) \in I^2 )</td>
<td>(-e \leq a \leq 1 - e)</td>
<td>(-e \leq b \leq 1 - e)</td>
</tr>
<tr>
<td>( w(0, 1) \in I^2 )</td>
<td>(-f \leq c \leq 1 - f)</td>
<td>(-f \leq d \leq 1 - f)</td>
</tr>
<tr>
<td>( w(1, 1) \in I^2 )</td>
<td>(0 \leq a + b + e \leq 1)</td>
<td>(0 \leq c + d + f \leq 1)</td>
</tr>
</tbody>
</table>
The contraction factor $\delta$ is given by

$$\delta = \max_{1 \leq i \leq n} (\delta_i)$$  \hspace{1cm} (2)

where $\delta_i$ can be calculated as

$$\delta_i = \sqrt{\frac{1}{2} \left( a^2 + b^2 + c^2 + d^2 + \sqrt{(a^2 + b^2 + c^2 + d^2)^2 - 4(ad - bc)^2} \right)}.$$

2.2. Similarity Measurement

How does one measure an affine IFS whose attractor is close to or looks like the target image? For binary images, the $S$ metric function is used to measure the quality of an approximation as below [11].

$$S(A, B) = \frac{|A \cap B|}{|A \cup B|}$$  \hspace{1cm} (3)

Here $|A \cap B|$ represents the number of black pixels after image $A$ intercept with image $B$, and $|A \cup B|$ represents the number black pixel after the of union of images $A$ and $B$.

Equation (3) expresses the fact that the smaller $S$ is, the worse similarity is. In this paper an extension of this function to measure the similarity has the form

$$S \left( B, \bigcup_{i=1}^{N} w_i(B) \right) = \frac{|B \cap \left( \bigcup_{i=1}^{N} w_i(B) \right)|}{|B \cup \left( \bigcup_{i=1}^{N} w_i(B) \right)|}.$$  \hspace{1cm} (4)

3. GENETIC ALGORITHMS FOR FRACTAL IMAGE COMPRESSION

Genetic algorithms (GA) are based on the principle of natural selection and natural genetics [12] and have been applied successfully to numerical and combinatorial optimization, statistical computation. Furthermore, GA is also an efficient searching method for approximations to global optimal in the huge and complicated space in relatively short time. For fractal image compression, we hope ultimately to find the best IFS, and hence genetic algorithms suit the current application.

3.1. Chromosome Encoding

For fractal image compression, the encoding of chromosome [11], which means the individual of a population, is actually the coding of IFS. Using the definition of an affine transformation in $I^2 = [0, 1]^2$ for binary images (see equation (1)), an efficient IFS with the parametric data in Table 1 is a workable chromosome. The encode of a gene is defined by

$$w_{ki} = [a_{ki}, b_{ki}, c_{ki}, d_{ki}, e_{ki}, f_{ki}],$$

where $k$ is the $k^{th}$ gene of the chromosome, $i$ is the $i^{th}$ contraction affine transformation of IFS, and $w_{ki}$ is the affine transformation. A chromosome can be encoded by the sequence of a gene. The sequence is

$$F_k = [w_{k1}, w_{k2}, w_{k3}, \ldots, w_{kn}],$$

where $k$ is the locus of the chromosome in population, $n$ represents the number of efficient affine transformations in the IFS, and $F_k$ is the IFS, called the chromosome. The concept is best described by the Java code below.
class Chromosome
{
    static int MAXMAPS=5, NPARS=6;
    double gene[MAXMAPS][NPARS];
    int mapnum;
}
}
class Population
{
    int POPSIZE=50;
    // POPSIZE is variable-length
    Chromosome p [POPSIZE]=new Chromosome();
    ...
    // genetic operation
    ...
}

Here the variables POPSIZE, MAXMAPS, and NPARS denote the size of the population, the maximum number of affine transformations of the IFS, and the actual number of affine transformations of the IFS, respectively.

3.2. Fitness Function

The evolution is driven by a fitness function [11] that is maximized during the process. The fitness value is selected to reflect a desirable trait in the members of the population [10]. In the case of fractal image compression, we measure three function values for evaluating chromosomes. The fitness function is the embodiment of multiobject optimal problem. We have three objectives to fulfill, including

(1) maximize the similarity measurement $S$,
(2) minimize the compression factor $\beta$, and
(3) minimize the contraction factor $\delta$.

The first objective can be done through equation (4). In order to achieve (2) define the penalty function $R_\eta : \mathbb{N} \to [0,1]$ by

$$R_\eta(\beta) = \exp \left( -\frac{\beta^2}{4\eta^2} \right).$$

The parameter $\eta$ is the expected value of $\beta$, which determines the penalty standard value. For example $\text{STDIFS N} = 4$ is the standard number of affine transformation. In this paper, parameter beta ($\beta$) equals to number of affine transformation and is modified between different runs in the range of $[0, 1]$ as the algorithm mentioned below.

$\text{STDIFS N} \leftarrow$ set penalty standard value;
if $\beta > \text{STDIFS N}$ then
    This function will penalize $\beta$ value;
else if $\beta < \text{STDIFS N}$ then
    This function will guerdon $\beta$ value;
end if

Finally the third objective requires another penalty function $P_\sigma : [0,1] \to [0,1]$ defined by

$$P_\sigma(C) = (1 - C^{10}) \exp \left( -\frac{C^2}{4\sigma^2} \right).$$

For the same reason, the parameter $\sigma$ is expected value of $C$, which is the standard compression factor, e.g., $\text{STDSCP} = 0.5$. The contraction factor $C$ is calculated by equation (2), and is used to
modify the method as described above. For a given space $I^2 = [0,1]^2$, let $B$ be a target image, and $A$ be attractor of the IFS, then the fitness function can be written as

$$F_B(C) = S \left( B, \bigcup_{i=1}^{N} w_i(B) \right) R_\psi(\beta) P_x(C).$$

(5)

### 3.3. Genetic Operators

There are three basic operators [11] in GA includes selection, crossover (recombination), and mutation.

**Selection** is an evolution operator that chooses a chromosome from the current generation of population for inclusion in the next generation of population. Before making them into the next generation of population, the selected chromosomes may undergo crossover or mutation, which depends upon the probability of crossover and mutation. The offspring consist of the next generation of population. We adopt roulette wheel selection strategy, in which the chance of a chromosome being selected is proportional to its fitness, and heuristic selection strategy. Heuristic selection acquires better individuals and avoids the evolution of population being degenerated during the evolution process. This mean that population set is set as best + BestParent + r * offspring, and then POPSIZE := best + r. This paper adopts best := 5, which represents the new generation of population containing chromosomes of the five best parents.

**Crossover** is a genetic operator that combines or mates two chromosomes (parents) to produce a new chromosome (offspring). The idea behind crossover is that the new chromosome may be better than both of their parents if it takes the best characteristics from each of the parents. Crossover occurs during evolution according to a user-definable crossover probability ($P_c$). In our experiments, we adopted one point crossover operator that randomly selects a crossover point within a chromosome before interchanges the two parent chromosomes at this point to produce two new offspring (see Figure 3). There is also the possibility for the length of offspring beyond the allowed maximum, and hence we should measure and modify offspring.

**Mutation** is also a genetic operator that alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With these new gene values, the genetic algorithm may be able to arrive at a better solution than was previously possible [13]. Mutation is an important part of the genetic search as it helps to
prevent the population from stagnating at any local optima. Mutation occurs during evolution according to a user-definable mutation probability \( P_m \), and this probability should usually be set fairly low (0.01 is a good first choice) [4].

Affine transformation in any plane can be a decomposed product of four types of basic transformation [14–16]. According to the characteristics of the contraction affine transformation, we designed four types of affine transformation mutator gene: rotation \( A_\theta \), scaling \( A_s \), stretching \( A_e \), and cutting \( A_c \). In other words,

\[
A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \text{or} \quad T = \begin{bmatrix} e \\ f \end{bmatrix}
\]

multiplies the associated transformation matrices, such as \( A_\theta \), \( A_s \), \( A_c \), \( A_e \). To acquire better offspring, the jitter mutation was considered and the jitter range function \([1,11]\) is defined by

\[
\alpha_{\text{new}} = \alpha_{\text{old}} + (1 - F_B(c))\varepsilon,
\]

where \( F_B(c) \) is the fitness of IFS (see equation (5)), and \( \varepsilon \) is a scaling factor chosen uniformly at random over an interval \([-1,1]\). When \( F_B(c) \to 1 \), the best transformation results will be influenced by a tiny mutation. In some cases there are unfitted transformations which appear after the mutation where chromosomes must be measured and modified.

3.4. The Algorithm

The pseudolanguage below describes the algorithm.

```plaintext
// initialization population
Population pop [MaxGap];
// MaxGap be maximum generation
int t=0;
pop[0]←generate(population)
/*generate randomly POPSIZE suitable chromosome */
Evaluate (pop[0]); //compute fitness
while (max fitness < target fitness or t < max generation) do
    pop[t+1]←select(pop[t],best);
    /*according to evaluation result, picking out five (best=5) best
    individuals into the next generation.*/
    pop[t+1]←crossover(pop[t],Pc);
    /* according to the IFS crossover probability P_c, picking out two
    parents to generate offspring (that is new individuals which are
    belong to the next generation). */
    pop[t+1]←mutation(P[t+1],Pm);
    /* apply the mutation operator to some individuals in pop[t+1]
    by P_m, creating new offspring.*/
    Modified (pop[t+1]);
    // assuring that chromosomes are suitable.
    Evaluate (pop[t+1]);
    t←t+1;
end while
```

4. SEQUENTIAL NUMERICAL RESULTS

Tests were performed using the algorithm described in Section 3. The original binary image "Sierpinski triangle", a 128 × 128 pixel array, two bits (0 = white, 1 = black) per pixel, is depicted in Figure 4. Table 3 shows the control parameters used in the GA method. We used these
Table 3. Definition of control parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Representation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>POPSIZE</td>
<td>Population size</td>
<td>50</td>
</tr>
<tr>
<td>Best</td>
<td>Selection number of best individual</td>
<td>5</td>
</tr>
<tr>
<td>( p_m )</td>
<td>IFS probability of mutation</td>
<td>0.1</td>
</tr>
<tr>
<td>( P_c )</td>
<td>IFS probability of crossover</td>
<td>0.5</td>
</tr>
<tr>
<td>stdIFS ( n )</td>
<td>Standard number of affine transformation</td>
<td>4</td>
</tr>
<tr>
<td>STDCP ( \sigma )</td>
<td>Standard contraction factor</td>
<td>0.5</td>
</tr>
<tr>
<td>MaxGap</td>
<td>Maximum generation</td>
<td>1000</td>
</tr>
</tbody>
</table>

Table 4. Parameters of the best solution of the IFS for the Sierpinski triangle.

<table>
<thead>
<tr>
<th>( W_1 )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5706581477090169</td>
<td>-0.007540638498583</td>
<td>-0.007540638498583</td>
<td></td>
</tr>
<tr>
<td>0.5706581477090169</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.37037450782307213</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.37037450782307213</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( W_2 )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5619923003136568</td>
<td>-0.22268091457791062</td>
<td>-0.00810324946971588</td>
<td></td>
</tr>
<tr>
<td>0.5619923003136568</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4236714293827556</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4236714293827556</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( W_3 )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5005243924668505</td>
<td>-0.00810324946971588</td>
<td>-0.00810324946971588</td>
<td></td>
</tr>
<tr>
<td>0.5005243924668505</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.020008425049636194</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.020008425049636194</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

parameters for 1000 generations and three runs (Pentium III processor, 256 Mbytes memory). Table 4 lists the coefficients of the best IFS solution, and the best image of the 999th generation is shown in Figure 5. Solutions for other parameters are found in Table 5. Despite the fact that our system did not manage to produce a 100% correct solution for the Sierpinski triangle problem, we can represent the IFS of the "Sierpinski triangle" by using only 3 * 6 = 18 real numbers similar to those used by the fractal compression encoding of the Sierpinski triangle image.
5. A DISTRIBUTED ALGORITHM

Since the structure of a fractal is of infinite complexity and our system adopts the extensive sampling space, the seeking of potent resolution is enormous. Speeding up the computation becomes the priority. The fundamental approach is to use distributed or parallel computing. Java RMI and Java multithreads were used to rewrite the algorithm mentioned above. Based on the genetic algorithm and IFS, the distributed fractal binary image compression algorithm may be written as below.

STEP 1. Define the long-distance interface for the class Chromosome—ChromosomeInterface.java:

```java
import java.rmi.*; //must be introduced
import java.io.*;
interface ChromosomeInterface extends Remote
{
    public double getgene(int m, int n) throws RemoteException;
    public int getmapn() throws RemoteException;
    public void setgene(int m, int n, double val) throws RemoteException;
    public void setmapn(int val) throws RemoteException;
}
```

STEP 2. Define the long-distance interface for class Population—PopulationInterface.java, the method for realization is as Step 1.

STEP 3. Define the long-distance interface for class Yimage (class for original image)—YimageInterface.java.

STEP 4. Define class Chromosome—Chromosome.java, and realize the method defined in the long-distance interface:
import java.rmi.server.UnicastRemoteObject;
import java.rmi.*;

class Chromosome extends UnicastRemoteObject implements ChromosomeInterface
{
    static int MAXMAPS=5,NPARS=6;
    private double gene[][]={new double[MAXMAPS] [NPARS+1];
    private int mapnum;
    Chromosome() throws RemoteException
    {
        //must be have
        public void setgene(int m,int n,double val) throws RemoteException{ ...
        public double getgene(int m,int n) throws RemoteException{ ...
        public void setmapn(int val) throws RemoteException{ ...
        public int getmapn() throws RemoteException{ ...
    }

    STEP 5. Define Population.java by extending the class UnicastRemoteObject.
    STEP 6. Define Yimage.java by extending the class UnicastRemoteObject.
    STEP 7. Define class Server—Server.java:
    import java.rmi.*;
    class Server
    {
        public static void main(String args[])
        {
            try
            {
                System.out.println("Server:Creating a Server");
                String name1="//zhengyang/Population"; //URL
                Population pp=new Population();
                System.out.println("Server:Binding it to "+name1);
                Naming.rebind(name1,pp); //Binding
                String name2="//zhengyang/Yimage";
                System.out.println("Server:Binding it to "+name2);
                Yimage yy=new Yimage();
                Naming.rebind(name2,yy);
                System.out.println("Server ready");
                catch(Exception e)
                {
                    System.out.println("ServerImpl:an exception occured:"+
                    +e.getMessage());
                    e.printStackTrace();
                }
            }
            
        }
    }
    STEP 8. Create RMI client class—fractal.java, there are multithreads included in this class
    which can transfer service on remote server in parallel.
    import java.io.*;
    import java.lang.Math;
    import java.rmi.*;
    public class fractal extends Thread
    {
        int thnum; //the number of thread
        static int th1=1,th2=2;
    fractal(int num)
{thnum=num;}
static String name;//the name of thread
public void run()
{
    switch(thnum)
    {
        case 1:{try
            {String name1=""+args[0]+"/Population";
            System.out.println("Locating the object Population");
            String name2=""+args[0]+"/Yimage";
            System.out.println("Locating the object Yimage");
            PopulationInterface pp=(PopulationInterface)Naming.lookup
                (name1);
            YimageInterface yy=(YimageInterface)Naming.lookup
                (name2);
            yi=yy.setyi();
            ...
            }
        catch(Exception e){System.err.println("Exception");
            System.err.println(e);
        }
        break; }//case 1
        case 2:{...
        break; }//case 2
    }
}
public static void main(String[] args)
{
    if (args.length<2)
    {System.out.println("please like this: java Client <server name
            or server IP>!");
        System.exit(0); }
    Thread first=new fractal(1);   
    first.start();
    Thread two=new fractal(2);   
    two.start();
}

The following steps are performed on different computer systems.

STEP 9. Compile codes of the defined classes and interfaces.

STEP 10. Startup registry on the server: prompt > rmiregistry.

STEP 11. Using rmic command to create stub class and skeleton class in the remote server. Note
that Steps 10 and 11 cannot be interchanged.
prompt > rmic Chromose
prompt > rmic Population
prompt > rmic Yimage

STEP 12. Startup server program java server.


We use the same parameters (see Table 3) for 1000 generations and one run on two PCs
(Pentium III processor, 256 bytes memory). Table 6 lists the coefficients of the best IFS solution,
Figure 6. Compare original image of Sierpinski triangle with decoding image.

Table 6. Parameters of this best solution of the IFS for the Sierpinski triangle.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1</td>
<td>0.48960515714935116</td>
<td>0.0594436839515899</td>
<td>-0.009433428700234212</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>W2</td>
<td>0.5508846344408153</td>
<td>0.011771287004961117</td>
<td>0.010362212542184845</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>W3</td>
<td>0.5116521069809691</td>
<td>-0.004616436718016512</td>
<td>0.00639578403335489</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td></td>
<td>0.5115723758846739</td>
<td>0.39067459203072424</td>
<td>0.020491956160032458</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>0.44823007344658905</td>
<td>0.118873679031798</td>
<td>-0.01886857400468423</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td></td>
<td>0.5728082653796672</td>
<td>0.3714986760558275</td>
<td>0.3760372617818562</td>
</tr>
</tbody>
</table>

Table 7. Other parameters of generation 1000.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contraction factor β</td>
<td>0.512453584304027</td>
</tr>
<tr>
<td>Similarity metric S</td>
<td>0.9258011872824389</td>
</tr>
<tr>
<td>Fitness f</td>
<td>0.355457463551795</td>
</tr>
<tr>
<td>Run-time time</td>
<td>1</td>
</tr>
<tr>
<td>Time</td>
<td>About 45 min</td>
</tr>
</tbody>
</table>

The decoding image of the 999th generation is shown in Figure 6 (other parameters solution see Table 7).

6. CONCLUSION AND FUTURE WORK

The variable-length chromosome encoding and the extensible search space is proposed in this paper. The algorithmic performance and efficiency have been improved greatly by introducing some modified GA parameters, multiobject fitness functions, a crossover operator, and manifold mutation operators. Furthermore, a distributed algorithm was implemented. The experimental results showed that the present algorithm has tremendous ability in searching for best solutions. Despite some suboptimal cases good solutions may be obtained for a very important problem in fractal image compression, i.e., finding efficient and good IFS such that the decoded image is similar to original one and has high image quality.

Further work currently being considered by the authors includes colour image compression and various distributed implementations.
REFERENCES