A Novel Solution for the Hausdorff Measure Computation of Sierpinski Carpet

QILI XIAO, LIFENG XI, JIFANG LI
School of Computer Science and Information Technology
Zhejiang Wanli University
Ningbo, Zhejiang, 315100
P. R. CHINA
http://www.zwu.edu.cn

Abstract: - The computation of the Hausdorff measure of fractals is the basic problem in fractal geometry. However, this is very difficult. The genetic algorithm is one of optimization algorithms to resolve complicated problems of wide scope, and has great capabilities in self-organizing, self-adaptation and self-learning. Lifeng Xi professor put forward to the thought of computing the Hausdorff measure of fractals using the genetic algorithm several years ago. In this paper, we mainly discuss the realization of the genetic algorithm on the Sierpinski carpet with compression ratio 1/4 in detail, including the encoding and decoding method, generation of the initial population, fitness computation, and genetic operators. Finally the Hausdorff measure of the Sierpinski carpet with compression ratio 1/4 is obtained. Experimental results show that the genetic algorithm is an effective and universal method of calculation of the Hausdorff measure.

Key-Words: - Sierpinski carpet; Hausdorff measure; genetic algorithm

1 Introduction

The computation and estimation of the Hausdorff dimension and measure of fractals are important problem in fractal geometry [1-3]. Generally, the computation of the Hausdorff dimension, especially Hausdorff measure, is very difficult. Koch curve, Sierpinski gasket and the Sierpinski carpet are the three well-known self-similar fractals on R^2. Their Hausdorff dimensions are known, but their Hausdorff measures remain unknown [4-9].

The genetic algorithm is adaptive methods which may be used to solve search and optimization problems [10-12]. The genetic algorithm is based on the genetic processes of biological organisms. Over many generations, natural populations evolve according to the principles of natural selection and "survival of the fittest". By mimicking this process, the genetic algorithm is able to "evolve" solutions to real world problems, if they have been suitably encoded. The basic principles of the genetic algorithm were first laid down rigorously by Holland [13, 14].

The genetic algorithm works with a population of "individuals", each representing a possible solution to a given problem. Each individual is assigned a "fitness score" according to how good a solution to the problem it is [15]. The highly-fit individuals are given opportunities to "reproduce", by "cross breeding" with other individuals in the population. This produces new individuals as "offspring", which share some features taken from each "parent". The least fit members of the population are less likely to get selected for reproduction, and so "die out" [16].

A whole new population of possible solutions is thus produced by selecting the best individuals from the current "generation", and mating them to produce a new set of individuals. This new generation contains a higher proportion of the characteristics possessed by the good members of the previous generation [17, 18]. In this way, over many generations, good characteristics are spread throughout the population. By favouring the mating of the more fit individuals, the most promising areas of the search space are explored. If the genetic algorithm has been designed well, the population will converge to an optimal solution to the problem

In the genetic algorithm, the evaluation function or objective function provides a measure of performance with respect to a particular set of parameters [19]. The fitness function transforms that measure of performance into an allocation of reproductive opportunities. The evaluation of a string representing a set of parameters is independent of the evaluation of any other string. The fitness of that string, however, is always defined with respect to other members of the current population. In the genetic algorithm, fitness is

defined by: f_i / f_A where f_i is the evaluation associated with string i and f_A is the average evaluation of all the strings in the population.

Fitness can also be assigned based on a string's rank in the population or by sampling methods, such as tournament selection. The execution of the genetic algorithm is a two-stage process. It starts with the current population. Selection is applied to the current population to create an intermediate population. Then recombination and mutation are applied to the intermediate population to create the next population. The process of going from the current population to the next population constitutes one generation in the execution of the genetic algorithm [20].

The standard genetic algorithm can be represented as follows:

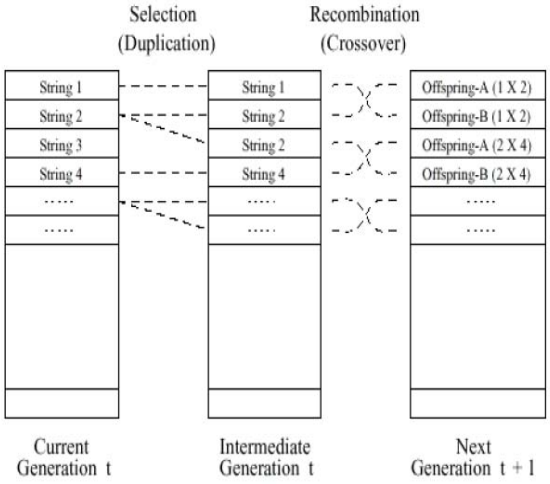

Fig. 1: standard genetic algorithm

In the first generation the current population is also the initial population. After calculating f_i / f_A for all the strings in the current population, selection is carried out. The probability that strings in the current population are copied (i.e. duplicated) and placed in the intermediate generation is in proportion to their fitness.

Individuals are chosen using "stochastic sampling with replacement" to fill the intermediate population [21, 22]. A selection process that will more closely match the expected fitness values is "remainder stochastic sampling". For each string i where f_i / f_A is greater than 1.0, the integer portion of this number indicates how many copies of that string are directly placed in the intermediate population. All strings (including those with f_i / f_A less than 1.0) then place additional copies in the intermediate population with a probability corresponding to the fractional portion of f_i / f_A. For example, a string with f_i / f_A = 1:36 places 1 copy in the intermediate population, and then receives a 0:36 chance of placing a second copy. A string with a fitness of f_i / f_A = 0:54 has a 0:54 chance of placing one string in the intermediate population. Remainder stochastic sampling is most efficiently implemented using a method known as stochastic universal sampling. Assume that the population is laid out in random order as in a pie graph, where each individual is assigned space on the pie graph in proportion to fitness [23]. An outer roulette wheel is placed around the pie with N equally-spaced pointers. A single spin of the roulette wheel will now simultaneously pick all N members of the intermediate population.

After selection has been carried out the construction of the intermediate population is complete and recombination can occur [24, 25]. This can be viewed as creating the next population from the intermediate population. Crossover is applied to randomly paired strings with a probability denoted p_c. (The population should already be sufficiently shuffled by the random selection process.) Pick a pair of strings. With probability p_c "recombine" these strings to form two new strings that are inserted into the next population. Consider the following binary string: 1101001100101101. The string would represent a possible solution to some parameter optimization problem. New sample points in the space are generated by recombining two parent strings. Consider this string 1101001100101101 and another binary string, yxyyxyxxyyyyxyxxy, in which the values 0 and 1 are denoted by x and y. Using a single randomly-chosen recombination point, 1-point crossover occurs as follows:

11010 \/ 01100101101

yxyyx /\ yxxyyyxyxxy

Swapping the fragments between the two parents produces the following offspring:

11010yxxyyyxyxxy and yxyyx01100101101

After recombination, we can apply a mutation operator. For each bit in the population, mutate with some low probability p_m. Typically the mutation rate is applied with 0.1%-1% probability. After the process of selection, recombination and mutation is complete, the next population can be evaluated. The process of valuation, selection, recombination and mutation forms one generation in the execution of the genetic algorithm.

Lifeng Xi and Zhongdi Cen put forward to the thought of computing the Hausdorff measure of fractals using the genetic algorithm [26]. However, they didn't continue to compute the Hausdorff measure aimed at certain a fractal. Under the direction of Lifeng Xi professor, we studied further how to compute the Hausdorff measure of the Sierpinski gasket with the genetic algorithm. The

more exact Hausdorff measure of the Sierpinski gasket with compression ratio 1/3 at present was obtained [27].

In this paper, we will mainly focus on how to use the genetic algorithm to compute the Hausdorff measure of the Sierpinski carpet with compression ratio 1/4. Section 2 outlines the Sierpinski carpet knowledge. Then we will discuss the encoding and decoding method, fitness computation in detail in Section 3. The experimental results and the future work will be given in Section 4. Finally, the summary will be given in Section 5.

2 Sierpinski Carpet

Take a unit square in the Euclidean plane R^2 and denote it by F_0. Dividing each side of F_0 into four equal parts, sixteen equal small squares are got with length 1/4. Removing the interior of all small squares except for the four ones lying on the vertexes of F_0, we get a set denoted by F_1. If the above procedure is repeated for each small square in F_1, the set F_2 is obtained. Repeating the above procedure infinitely (such as Fig. 2), we have $F_0 \supset F_1 \supset ... \supset F_m \cdots$. The non-empty set $F = \cap_{m=0}^{\infty} F_m$ is called the Sierpinski carpet yielded by F_0 [26- 28].

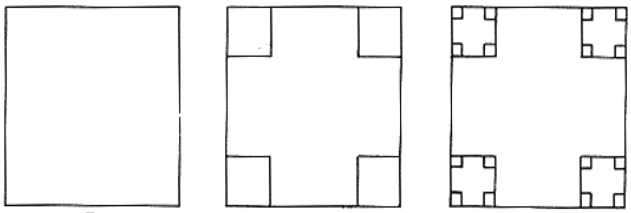

Fig. 2: the structure of the Sierpinski carpet

By [1, 3], the Hausdorff dimension s of F is 1, and the Hausdorff measure $H(F)$ of F meets the following condition [29, 30],

$$H(F) = \lim_{m \to \infty} \inf_{U \in F_m'} \frac{|U|}{\mu(U)}$$

where are the following four mappings.

$S_1(x) = x/4$

$S_2(x) = x/4 + (3/4, 0)$

$S_3(x) = x/4 + (3/4, 3/4)$

$S_4(x) = x/4 + (0, 3/4)$

and $F_{i_1 i_2 \ldots i_m} = S_{i_1} \circ S_{i_2} \circ ... \circ S_{i_m}(F_0)$, $F_m' = \{U \mid U$ is a union of some small squares $F_{i_1 i_2 \ldots i_m}$ in the m-th structure$\}$

3 Realization of Genetic Algorithm

3.1 individual coding

Before the genetic algorithm can be run, a suitable coding (or representation) for the problem must be devised. We also require a fitness function, which assigns a figure of merit to each coded solution. During the run, parents must be selected for reproduction, and recombined to generate offspring.

It is assumed that a potential solution to a problem may be represented as a set of parameters. These parameters (known as genes) are joined together to form a string of values (often referred to as a chromosome). For example, if our problem is to maximise a function of three variables F(x; y; z), we might represent each variable by a 10-bit binary number (suitably scaled). Our chromosome would therefore contain three genes, and consist of 30 binary digits. The set of parameters represented by a particular chromosome is referred to as a genotype. The genotype contains the information required to construct an organism which is referred to as the phenotype [31, 32]. For example, in a bridge design task, the set of parameters specifying a particular design is the genotype, while the finished construction is the phenotype.

The fitness of an individual depends on the performance of the phenotype. This can be inferred from the genotype. It can be computed from the chromosome using the fitness function. Assuming the interaction between parameters is nonlinear, the size of the search space is related to the number of bits used in the problem encoding. For a bit string encoding of length L; the size of the search space is 2^L and forms a hypercube. The genetic algorithm samples the corners of this L-dimensional hypercube. Generally, most test functions are at least 30 bits in length; anything much smaller represents a space which can be enumerated. Obviously, the expression 2^L grows exponentially. As long as the number of “good solutions” to a problem is sparse with respect to the size of the search space, then random search or search by enumeration of a large search space is not a practical form of problem solving. On the other hand, any search other than random search imposes some bias in terms of how it looks for better solutions and where it looks in the search space.

With the wide application of genetic algorithm, many coding methods are put forward. There are three kinds binary coding method, float coding method, symbol coding method. Binary coding method is the most familiar coding method that an individual is made of some 0 or 1. Binary coding has the advantage [9, 17, 33]. It is the advantage that coding and decoding is easy to operate. The other

advantage is that the implement of the genetic operation is easy.

During the process of calculating the Hausdorff measure, the method of binary code with fixed length is used. The encoding method is as follows: if $F_{i_1 i_2 \cdots i_m}$ is contained in the set $U \in F_m^{'}$, then its corresponding code is 1, otherwise its corresponding code is 0. In this way, a code list of 4^m length can show whether each one of the 4^m equal small squares in the m-th structure is chosen or not. In order to solve the problem easily, the origin is regarded as the starting point here, and every small square is encoded according to the anticlockwise order.

Now let's state the detail encoding method with the second structure. In the second structure, there are 4^2=16 equal small squares altogether. If we regard the origin as the starting point and encode every one according to the anticlockwise order, the marks of the 16 squares are as shown in Fig. 3. We choose every one of the 16 squares one by one according to their marks from small to large. If a square is chosen, we use 1 to represent on the corresponding position of the individual code, otherwise use 0. For example, 1001 0010 0100 0000 represents that four squares of the 16 squares in the second structure are chosen and their marks are 1, 4, 7 and 10 respectively (such as Fig. 3).

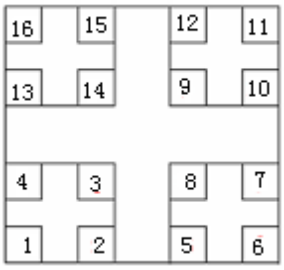

Fig. 3: small squares of the second structure

3.2 generation of initial population

The population is the foundation of evolution of the genetic algorithm. The character of the population decides the search capability of the genetic algorithm. And the astringency of the genetic algorithm is determined by the astringency of the population [9, 31]. During the calculation of the Hausdorff measure, individuals of initial population are randomly generated by the system.

3.3 decoding

During the calculation of the Hausdorff measure, an individual represents a choice. Then it is the decoding method that any binary digit 1 of the individual represents which of the 4^m equal small squares is chosen, namely that the corresponding small square is the result which functions act on F_0. The detail of decoding is as follows:

(1) Firstly, divide the whole code into four equal length groups, every small square that is represented by 0 or 1 in the first group is inside the square $S_1(F_0)$, and every small square that is represented by 0 or 1 in the second group is inside the square $S_2(F_0)$, and every small square that is represented by 0 or 1 in the third group is inside the square $S_3(F_0)$, and every small square that is represented by 0 or 1 in the forth group is inside the square $S_4(F_0)$.

(2) Secondly, respectively divide every group above into four equal length groups, every small square that is represented by 0 or 1 in the first group of every group above is respectively inside the square $S_1 \circ S_1(F_0)$ and $S_1 \circ S_2(F_0)$ and $S_1 \circ S_3(F_0)$ and $S_1 \circ S_4(F_0)$, and every small square that is represented by 0 or 1 in the second group of every group above is respectively inside the square $S_2 \circ S_1(F_0)$ and $S_2 \circ S_2(F_0)$ and $S_2 \circ S_3(F_0)$ and $S_2 \circ S_4(F_0)$, and the rest may be deduced by analogy.

(3) Finally, continue dividing every group of the second step into four equal length groups until there is only one digit 0 or 1 in every group.

Now let's state the detail decoding method with the third structure. In the third structure, there are 4^3=64 equal small squares altogether. For the individual 0100 0110 0000 0000 0010 0000 0011 0000 0000 0000 0101 0000 0010 0000 0010 1100, it is obvious that the corresponding small square represented by every 1 of the individual is the result which functions act on F_0 through the following step.

(1) Divide the whole code into four equal length groups, the following four groups can be obtained.

0100 0110 0000 0000
0010 0000 0011 0000
0000 0000 0101 0000
0010 0000 0010 1100

Every small square that is represented by 0 or 1 in the first group 0100 0110 0000 0000 is inside the square $S_1(F_0)$. Every small square in the second group 0010 0000 0011 0000 is inside the square $S_2(F_0)$. Every small square in the third group 0000 0000 0101 0000 is inside the square $S_3(F_0)$. Every small square in the forth group 0010 0000 0010 1100 is inside the square $S_4(F_0)$.

(2) Respectively divide every group above into four equal length groups, the following sixteen groups are obtained.

For the first group of the first step, there are four groups following.

0100
0110
0000

0000

Every small square that is represented by 0 or 1 in the first group 0100 is inside the square $S_1 \circ S_1(F_0)$. Every small square in the second group 0110 is inside the square $S_1 \circ S_2(F_0)$. Every small square in the third group 0000 is inside the square $S_1 \circ S_3(F_0)$. Every small square in the forth group 0000 is inside the square $S_1 \circ S_4(F_0)$.

For the second group the first step, there are four groups following.

0010
0000
0011
0000

Every small square that is represented by 0 or 1 in the first group 0010 is inside the square $S_2 \circ S_1(F_0)$. Every small square in the second group 0000 is inside the square $S_2 \circ S_2(F_0)$. Every small square in the third group 0011 is inside the square $S_2 \circ S_3(F_0)$. Every small square in the forth group 0000 is inside the square $S_2 \circ S_4(F_0)$.

For the third group of the first step, there are four groups following.

0000
0000
0101
0000

Every small square that is represented by 0 or 1 in the first group 0000 is inside the square $S_3 \circ S_1(F_0)$. Every small square in the second group 0000 is inside the square $S_3 \circ S_2(F_0)$. Every small square in the third group 0101 is inside the square $S_3 \circ S_3(F_0)$. Every small square in the forth group 0000 is inside the square $S_3 \circ S_4(F_0)$.

For the forth group of the first step, there are four groups following.

0010
0000
0010
1100

Every small square that is represented by 0 or 1 in the first group 0010 is inside the square $S_4 \circ S_1(F_0)$. Every small square in the second group 0000 is inside the square $S_4 \circ S_2(F_0)$. Every small square in the third group 0001 is inside the square $S_4 \circ S_3(F_0)$. Every small square in the forth group 1100 is inside the square $S_4 \circ S_4(F_0)$.

(3) Continue dividing every group of the second step into four equal length groups until there is only one digit 0 or 1 in every group. The small square represented by 1 of the individual 0100 0110 0000 0000 0010 0000 0011 0000 0000 0000 0101 0000 0010 0000 0010 1100 is $S_2 \circ S_1 \circ S_1(F_0)$, $S_2 \circ S_2 \circ S_1(F_0)$, $S_3 \circ S_2 \circ S_1(F_0)$, $S_3 \circ S_1 \circ S_2(F_0)$, $S_3 \circ S_3 \circ S_2(F_0)$, $S_4 \circ S_3 \circ S_2(F_0)$, $S_2 \circ S_3 \circ S_3(F_0)$, $S_2 \circ S_4 \circ S_4(F_0)$ respectively from left to right.

The following is the detail programming code to realize decoding with C programming language [34, 35, 36].

```
k=0;  l=j;
for (r=m; r>0; r=r-1)
{
p=pow(4,r);
l=l%p;
if ( l<0.25*p) { func[k]='a'; }
else  if (l<0.5*p) { func[k]='b';}
else  if(l<0.75*p) {func[k]='c';}
else  if (l<p) { func[k]='d';}
k=k+1;
}
```

REMARK 1 Variable *j* is used to keep the j-th figure of an individual code.

REMARK 2 In the whole programming code, variable *m* expresses the *m*-th construction during the generation of Sierpinski carpet.

REMARK 3 Array *func*[*m*] is used to keep the *m* functions that act on the unit square F_0.

REMARK 4 In the programming code, ‘a’ expresses the function S_1, ‘b’ expresses the function S_2, ‘c’ expresses the function S_3, and ‘d’ expresses the function S_4.

3.4 Fitness evaluation

Once initial individuals are randomly generated according to the way mentioned above, we then have to optimize the individuals to obtain the best results. Therefore, a numerical fitness function is needed in order to quantitatively evaluate the suitability of each individual. In the genetic algorithm, the goodness and badness of an individual is distinguished by individual fitness. The good individuals are selected by individual fitness, and preserved in the next generation, so that more and more good individuals generate in the next generation. During the calculation using the genetic algorithm, fitness function influences on astringency of the genetic algorithm and the pace of constringency [37-39].

Here fitness function is $F(U)= \mu(U)/|U|$, then finding minimum value is transformed into finding maximum value, where $\mu(U)$ represents the value of dividing the number of 1’s in the individual code by 4^m, and $|U|$ represents the maximum distance of two random vertexes of eight vertexes of two random small squares. In order to calculate the maximum distance of any set U, a construction function series of every square in U must be found

first, and then the coordinates of all vertexes of all small squares in U can be calculated.

Step 1 Calculate the coordinates of all vertexes of all small squares. The detail calculation is as follows.

(1) Work out the construction function series of all small squares as introduced above.

(2) Calculate the coordinates of the left bottom vertex of all small squares. The detail is as follows.

Suppose *A*, *B*, *C* and *D* are four equal small squares in the *m*-th structure, and they are inside the square *E* that is one of the squares in the (*m*-1)-th structure (such as Fig. 4). If the coordinates of the left bottom vertex of the square *E* is (x_0, y_0), the coordinates of the left bottom vertex of small squares *A*, *B*, *C* and *D* are listed as follows:

$$(x_A, y_A) = (x_0 , y_0)$$
$$(x_B, y_B) = (x_0, y_0) + (3/4*l , 0)$$
$$(x_C, y_C) = (x_0, y_0) + (3/4*l , 3/4*l)$$
$$(x_D, y_D) = (x_0, y_0) + (0 , 3/4*l)$$

Where l represents the length of the side of the square *E*, and $l=1/4^{m-1}$.

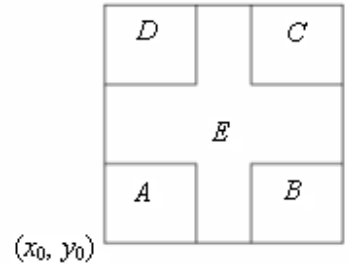

Fig. 4: square *E* and four small squares of the next structure

Suppose the corresponding functions of any a small square are in the array variable func[m]. The following programming code can calculate the coordinates of the left bottom vertex of the small square.

```
double **loc;
loc=(double**)malloc(sizeof(double*)*4*
chromlength);
for(i=0;i<4*chromlength;i++)
loc[i]=(double*)malloc(sizeof(double)*2);
for (k=0;k<m;k++)
{
 pw=1/pow(4, k);
 switch  (func[k])
case 'a':
{
 loc[line][0]=loc[line][0];
 loc[line][1]=loc[line][1];
 break;
}
case 'b':
{
 loc[line][0]=loc[line][0]+0.75*pw;
 loc[line][1]= loc[line][1];
 break;
}
case  'c':
{
 loc[line][0]=loc[line][0]+0.75*pw;
 loc[line][1]= loc[line][1]+ 0.75*pw;
 break;
}
case 'd':
{
 loc[line][0]=loc[line][0];
 loc[line][1]= loc[line][1]+ 0.75*pw;
 break;
}
}
```

REMARK 5 In the whole paper, variable loc is an array that is used to preserve the vertexes of small squares.

REMARK 6 In the whole paper, variable chromlength is used to preserve the length of individual code.

(3) Calculate the coordinates of the other vertexes of all small squares. The detail is as follows: If a small square is one of squares in the m-th structure and the coordinates of the left bottom vertex is (x, y), the coordinates of other vertexes in anticlockwise order are the following:

(x, y) + ($1/4^m$, 0)
(x, y) + ($1/4^m$, $1/4^m$)
(x, y) + (0, $1/4^m$)

The detail programming code is following.

```
x= loc[line][0];
y= loc[line][1];
pw=1/pow(4,m);
for ( k=2 ; k< 5; k++)
{
 line=line+1;
 switch (k)
 case 2:
{
 loc[line][0]=x+pw;
 loc[line][1]= y ;
 break;
}
case 3:
{
 loc[line][0]=x+pw;
 loc[line][1]=y+pw;
break;
}
case 4:
{
```

```
loc[line][0]= x;
loc[line][1]=y+pw;
break;
}
}
```

Step 2 Calculate the individual fitness.

(1) Find the coordinates of the small squares chosen in the current individual according to the result of step 1.

(2) Calculate the diameter $|U|$ and the number of 1's in the individual code.

(3) Calculate the corresponding fitness.

The detail programming code is following.

```
if (num==0)   popfitness[i]= 0;
else
{
 for (j=0 ; j< line ; j++)
   for (k=j+1 ; k<= line ; k++)
   {
     x1=loc[j][0];
     x2=loc[k][0];
y1=loc[j][1];
y2= loc[k][1];
s=sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
if (s>diameter)   diameter=s;
}
popfitness[i]= num/(diameter*chromlength);
}
```

REMARK 7 Variable *num* is used to preserve the 1's number of an individual.

REMARK 8 Variable *popfitness* is an array that is used to preserve individual fitness.

3.5 Genetic operations

Characteristic of the genetic algorithm is that the imitating of the selection, crossover and mutation of biology inheritance and evolution obtains the best solution of problem. The selection, crossover and mutation processes are the most important parts of the genetic algorithm.

3.5.1 Selection

This operator selects chromosomes in the population for reproduction. The fitter the chromosome, the more times it is likely to be selected to reproduce. Selection embodies the theory of survival of the fittest. The evaluation of individuals is the basis of the selection operation. The purpose of selection is to improve the astringency and the computation efficiency. The familiar selection operators are proportion selection, preserving the best individual, expectation selection. Proportion selection is the basal and common selection operation. The standard genetic algorithm uses proportion selection. But the standard genetic algorithm has no astringency [40]. The selection of preserving the best individual is used in order to obtain the best solution of problem. The selection of preserving the best individual can guarantee that the individual with the biggest fitness appearing while the implement of the genetic algorithm can be preserved [41].

3.5.2 Crossover

This operator exchanges subsequence of two chromosomes to create two offspring. For example, the strings 10000100 and 11111111 could be crossed over after the third locus in each to produce the two offspring 10011111 and 11100100.

Crossover operator is the most important operation of genetic operation in the genetic algorithm. Crossover operation is the main method to generate the new individual. Crossover operation decides the whole search capability of the genetic algorithm. The search capability of the genetic algorithm is improved rapidly. The familiar crossover operators are one-point crossover, two-point crossover and multi-point crossover. The traditional genetic algorithm uses 1-point crossover, where the two mating chromosomes are each cut once at corresponding points, and the sections after the cuts exchanged. However, many different crossover algorithms have been devised, often involving more than one cut point. DeJong investigated the effectiveness of multiple-point crossover, and concluded that 2-point crossover gives an improvement, but that adding further crossover points reduces the performance of the genetic algorithm. The problem with adding additional crossover points is that building blocks are more likely to be disrupted. However, an advantage of having more crossover points is that the problem space may be searched more thoroughly.

During the calculation of the Hausdorff measure, we use the three crossover above. At the same time, we seek after a new crossover operation "and / or" crossover [42-44]. "And / or" crossover displays the more excellent capability. The detail realization is following.

(1) Select two individuals $F1$ and $F2$.

(2) Generate a new individual with "and" Boolean calculation on F1 and F2.

(3) Generate a new individual with "or" Boolean calculation on F1 and F2.

The detail programming code is the following.

```
for (i=0;i<popsize-1;i=i+2)
{
 p=rand()%1000/1000.0;
 if (p<pc)
```

```
 for (j=0;j<chromlength;j++)
{
 pop[i][j]=pop[i][j] && pop[i+1][j];
pop[i+1][j]=pop[i][j] || pop[i+1][j];
}
}
```

3.5.2 Mutation

Mutation is traditionally seen as a "background" operator, responsible for re-introducing inadvertently "lost" gene values (alleles), preventing genetic drift, and providing a small element of random search in the vicinity of the population when it has largely converged. It is generally held that crossover is the main force leading to a thorough search of the problem space.

This operator randomly flips some bits in a chromosome. For example, the string 00000100 might be mutated in its second position to yield 01000100. Mutation can occur at each bit position in a string with some probability, usually very small (e.g., 0.001).

The advantage of mutation operation is the following [45]. One is to improve the local search capability. The other is to maintain the diversity of population and to prevent pre-maturation.

During the calculation of the Hausdorff measure, we mainly use gene mutation.

4 Experimental Results and Analysis

During the calculation of the Hausdorrf measure, the values of the key parameters are the following. The crossover probability p_c=0.9, the mutation probability p_m=0.001, the population size *popsize*=50, the generation number *gen*=100. According to above-mentioned ideas, we have developed the whole programming code. The result of the experiment is what Table 1 shows.

Table 1: values of H (F)

m	$H(F)$	computation times (unit: second)
1	1.41421	1
2	1.41421	1
3	1.41421	1
4	1.41421	3
5	1.41421	120
6	1.41421	21106
7	1.41421	4200658

From Table 1, we can conclude that the Hausdorff measure of the Sierpinski carpet with compression ratio 1/4 H (F) =1.41421. Obviously, the result is uniform with the result of the 30th reference [30]. At the same time, the experimental results have proved further that the exact Hausdorff measure of the Sierpinski carpet with compression ratio 1/4 is 1.41421.

It can be seen easily that the Hausdorff measure can be obtained quickly when m≤6 from Table 1. But when m>6, the calculating time increases sharply. During the experiment, we found that much time is spent on calculating the fitness. Therefore, the future work is to improve the method of calculating the fitness further.

5 Summary

It is noticed that the search space expands rapidly with the increase of times of recursions due to the structure of fractals. So far it is hard to calculate the exact value of Hausdorff measure by computer. Therefore, it seems necessary to find an effective algorithm to calculate or estimate the Hausdorff measure. The major advantage of the genetic algorithm is the flexibility and robustness as a global search method. They can deal with highly nonlinear problems and non-differentiable functions as well as functions with multiple local optima. In fact, the above experiment has proved that genetic algorithm is a kind of effective method to calculate or estimate the Hausdorff measures at present.

References:

[1] Falconer, K., Techniques in Fractal Geometry, Wiley, 1997.

[2] Qiuli Guo, Hausdorff Dimension of Level Set Related to Symbolic System. International Journal of Nonlinear Science. Vol. 3, 2007, 63-67.

[3] Qiuli Guo, Haiyi Jiang, Lifeng Xi, Hausdorff Dimension of Generalized Sierpinski Carpet, International Journal of Nonlinear Science, Vol. 2, 2006, 153-158.

[4] Li Wenxia, Xiao Dongmei, Dimensions of Measure on General Sierpinski Carpet, Acta Mathematica Sinica, Vol. 19, 1999, 81- 85.

[5] Zuoling Zhou: Hausdorff Measure Of Self-Similar Set: Koch Curve. Science In China(A). Vol. 28, 1998, 103-107.

[6] Dai Xinrong, An Estimate of Hausdorff Measure of Sierpinski Gasket, Journal of Zhejiang University of Technology, Vol. 29, 2001, 86-90.

[7] Zhou Zuoling, Hausdorff measure of self-similar set: Koch curve, Science in China (A). Vol. 28, 1998, 103-107.

[8] Wang Xinhua, Hausdorff Measure of Sierpinski Carpet, Progress in Natural Science, vol. 11, 2001, 383- 387.
[9] Zeng Chaoyi, Xu Shaoyuan, The Hausdorff Measure of a Sierpinski Sponge, Mathematics in Practice and Theory, vol. 9, 2007, 141-143.
[10] Zhou Ming, Sun Shudong, Theory and Application of Genetic Algorithm, National Defence Industry Press, 2002.
[11] Zhang Lin, Zhang Bo, Research on the Mechanism of Genetic Algorithms, Journal of Software, Vol. 11, 2000, 945-952.
[12] Hou Guangkun, Luo Jiangpeng, Modeling Idealized Parallel Genetic Algorithms, Journal of Software, Vol. 5, 1999, 557- 560.
[13] Xiao Jun, Application Of Genetic Algorithms, Computer Science, Vol. 32, 2005, 246- 247.
[14] Yu Nong, Li Yushu, Wang Runsheng, Optimal Morphological Filters Using Genetic Algorithm for Automatic Target Detection, Journal of Software, Vol. 24, 2001, 337-346.
[15] Esmaelzadeh, R.; Naghash, A.; Mortazavi, M. Rendezvous Trajectory Optimization Using Real Genetic Algorithm Combined with Gradient Method. WSEAS Transactions on Systems, Vol. 5, 2006, 2875-2880.
[16] Li Minqiang, Kou JiSong, Theory and Application of Genetic Algorithm, Beijing Science & Technology Press, 2002.
[17] Wang Xiaoping, Cao Liming, Theory, Application and Software Realization of Genetic Algorithm, Publishing Company of Xi'an Jiaotong University, 2002.
[18] Dubey, Manisha; Sharma, Avdhesh; Agnihotri, Gayatri; Gupta, Pankaj, Optimal Tuning of Parameters of Fuzzy Logic Power System Stabilizer Using Genetic Algorithm, WSEAS Transactions on Systems, Vol. 4, 2005, 225-232.
[19] Joachins, Parallel Geneticalgorithms: Theory and Applications, Isopress, 1993.
[20] Zheng Liping, Hao Zhongxiao, A Review on the Theory for the Genetic Algorithm, Vol. 21, 2003, 50-54.
[21] Dai Xiaohui, Li Mingqiang, Kou Jisong, Survey on the Theory of Genetic Algorithms, Control and Decision, Vol. 15, 2000, 263-269.
[22] Benitez-Perez, Hector; Saavedra-Hernandez, H.; Ortega-Arjona, J.L. On-Line Reconfiguration For A Type of Networked Control System Using Genetic Algorithms, WSEAS Transactions on Systems, Vol. 6, 2007, 167-172.
[23] Ren Ping, Genetic Algorithms, Journal of Engineering Mathematics, Vol. 16, 1-8.
[24] Mahfoud, S., Mani, G., Financial Forecasting Using Genetic Algorithms, Applied Artificial Intelligence, Vol. 10, 1996, 543-565.
[25] Fogel D B., Evolutionary Computation: Toward a New Philosophy of Machine Intelligence, IEEE Press, 1995.
[26] Xi Lifeng, Cen Zhongdi, Some Front Problems of Fractal- Computation of Hausdorff Measure, Journal of Zhejiang Wanli University, Vol. 14, 2001, 1-3.
[27] Qili Xiao, Lifeng Xi, Hausdorff Measure Estimation of Sierpinski Gasket Based on Genetic Algorithm. Computer Engineering and Applications, Vol. 44, 2008, 61-63.
[28] Huang Chunchao, A Elementary Proof for Hausdorff Mearsure of Sierpinski Carpet, Acta Mathematica Sinica, Vol. 43, 2000, 599- 603.
[29] Zhao Yanfen, Huang Jinghua, The Hausdorff Measure of Sierpinski Carpet on Rectangle, Journal of Hubei University, Vol. 21, 1999, 185- 189.
[30] Zhou Zuoling, Wu Min, The Hausdorff Measure of a Sierpinski Carpet, Science in China (A). Vol. 29, 1999, 138-144.
[31] Chen Guoliang, Wang Xufa, Genetic Algorithm and Its Application, People Post Publishing Company, 2001.
[32] Cui Xunxue, Lin Chuang, A Preference-Based Multi-Objective Concordance Genetic Algorithm, Journal of Software, Vol. 16, 2005, 761-770.
[33] Dai Xiaohui, Li Minqiang, Koujisong, Study on the Performance Analysis of Genetic Algorithms, Journal of Software, Vol. 12,2001, 742- 750.
[34] Yu Xinning, Wang Wenpeng, Zhang Jun, Module Programming of Genetic Algorithm, Vol. 13, 2003, 4-7.
[35] Michalewicz, Z., Genetic Algorithms + DataStructures = Evolution Programs, Springer-Verlag, 1994.
[36] Koza, J.R., Genetic Programming, MIT Press, Cambridge, MA, 1991.
[37] He Jun, Huang Houkuang, Kang Lishang, The Computational Time of Genetic Algorithms for Fully Deceptive Problem, Chinese J.Computers, Vol. 22, 1999, 999-1003.
[38] Yang Qiwen, Jiang Jingping, Zhang Guohon, Improving Optimization Speed for Genetic Algorithms, Journal of Software, Vol. 12, 2001, 270- 275.
[39] Zhang Jin, Li Dongli, Li Ping, Comparative Study of Genetic Algorithms Encoding Mechanism, Journal of China University of Mining & Technology, Vol. 31, 2002, 637-640.

[40] Xu Zongben, Nie Zankan, Zhang Wenxiu, Almost Sure Convergence of Genetic Algorithms: a Martingale Approach, CHINESE J. COMPUTERS, Vol. 25, 2003, 785-793.
[41] Wu Haoyang, Chang Bingguo, Zhu Changchun, Liujunhua, A Multigroup Parallel Genetic Algorithm Based on Simulated Annealing Method, Journal of Software, Vol. 11, 2000, 416-420.
[42] Tu Huiyuan, Crossover Operator Analysis of Genetic Algorithms, J. Wuhan Univ., Vol. 51, 2005, 22-24.
[43] Gong Daoxiong, Ruan Xiaogang, A New Crossover Operator, Computer Engineering and Applications, Vol. 6, 2004, 7-11.
[44] Ren Qingsheng, Ye zhongxing, Zeng Jin, Qi Feihu, Search Capability of Crossover Operator, Journal of Computer Research & Development, Vol. 36, 1999, 1317-1322.
[45] Wang Jiyi, Wu Yanxian, Implementation of Adaptive Multiple Bit Mutation Genetic Algorithm, Computer Science, Vol. 30, 2003, 141-143.