Research and Implementation on Genetic Algorithms for Graph Fitness Optimization

JIN MIN, WANG QIN, XI LIFENG
Computer Science and Information Technology College
Zhejiang Wanli University
Qianhu South Road 8, Ningbo
CHINA
http://www.zwu.edu.cn

Abstract: -- Graph fitness optimization is a difficult problem in data fitness. Genetic algorithms(GAs), which can yield accurate results if they start with suitable parameters, have been used to solve difficult problems with objective functions which usually are multi-modal, discontinuous, and nondifferentiable. In this paper, we design a genetic algorithm (GA) to optimize effect on self-affine fractal interpolation function (AFIF) and give result. The software was tested on realistic graphs. The validation and effectiveness of the method to be able to find the optimal fractal function are presented and demonstrated.

Key-Words: - Graph Fitness; Optimization; Genetic Algorithm; Fractal; AFIF

1 Introduction

Graph fitness is one of the most interesting, important, and successful applications of data fitness. For example, fingerprint recognition, image retrieval, stock analysis, and so on. It is a challenging problem to find the optimization on graph fitness.

There are certain optimization problems that become unmanageable using combinatorial methods as the number of objects becomes large. A typical example is the traveling salesman problem, which belongs to the NP-complete class of problems. For these problems, There are lots of algorithms, such as simulated annealing (SA), tuba search (TS), artificial neural networks, greedy algorithm and genetic algorithm (GA), etc, can be used to solve that problem. But each algorithm has advantage and disadvantage.

There is a very effective practical algorithm called simulated annealing (thus named because it mimics the process undergone by misplaced atoms in a metal when its heated and then slowly cooled). While this technique is unlikely to find the optimum solution, it can often find a very good solution, even in the presence of noisy data [1]. TS is easy to realized, and suitable for solving standard combination optimization problem. But it isn't suitable for solving binary array problem [2]. Artificial neural networks are able to be adaptive, teach itself and parallel process, but are difficulty to be realized by programs [3]. Greedy algorithms are simple and straightforward, but usually can't find optimization [4]. GA may be more effective in and easier to be realized by programs.

GA can yield accurate results providing that they are fed with suitable parameters to start with. The GA has been used to solve difficult problems with objective functions, which usually are multi-modal, discontinuous, and nondifferentiable[5]. These algorithms maintain and manipulate a family, or population, of solutions and implement a "survival of the fittest" strategy in their search for better solutions. This provides an implicit as well as explicit parallelism that allows for the exploitation of several promising areas of the solution space at the same time [6,7].

Fractal is a tool in data fitness and interpolation. It's better to fit rough curves and vibrating data, such as mountain range outlines, electrical-diagrams ...etc. AFIF is able to create complex graph by setting several parameters.

Given points $\{(x_i, y_i)\}_{i=0}^{N}$ in the plane, we suppose $\{\omega_1, \omega_2, \ldots, \omega_N\}$ is an iterated function system satisfying

$$\omega_i\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_i & 0 \\ c_i & d_i \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} g_i \\ h_i \end{pmatrix}, \quad (1)$$

$$\omega_i\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} x_{i-1} \\ y_{i-1} \end{pmatrix}, \ \omega_i\begin{pmatrix} x_N \\ y_N \end{pmatrix} = \begin{pmatrix} x_i \\ y_i \end{pmatrix}, \quad (2)$$

where $|d_i|<1$ and $a_i \in (0,1)$ for any i with $1 \le i \le N$. By (3) and (4), we notice that $\{\omega_i\}_{i=1}^N$ is determined by $\{(x_i, y_i)\}_{i=0}^N$ and. We always call $\{d_i\}_{i=1}^N$ vertical factors and $\{(x_i, y_i)\}_{i=0}^N$ interpolation points respectively.

Definition 1. Suppose $f(x)$ is a continuous function on the interval $[x_0, x_N]$. Let

$$\Gamma = \{(x, f(x)) : x \in [x_0, x_N])\} \quad (3)$$

be the graph of $f(x)$. We say that $f(x)$ is an affine fractal interpolation function(AFIF), if

$$\Gamma = \bigcup_{i=1}^N \omega_i(\Gamma). \quad (4)$$

For AFIF defined by formulas (1)-(4), the dimension $\dim_B \Gamma$ of the graph Γ satisfies the following dimension formula ([1]):

$$\sum_{i=1}^N |d_i| \cdot |a_i|^{\dim_B \Gamma - 1} = 1. \quad (5)$$

Remark: Formula (5) holds when the interpolation points do not lie in a line simultaneous_ ly and $\sum_{i=1}^N |d_i| > 1$. Any connected part of the graph Γ of AFIF has the same dimension $\dim_B \Gamma$.

From the concept of AFIF, if we have several interpolation points from original graph and suitable vertical factors, we can create graph, which approximate to the original graph.

As for the more information of AFIF, please refer to [8-9]. From the concept of AFIF, if we have several interpolation points from original graph and suitable vertical factors, we can create graph, which approximate to the original graph.

To fit a presented graph, it is not rectify parameters enough. It is a challenging problem to find the optimization of solution space.

Section 1 presents the basic GA, and in Section 2 the step of optimization of affine fractal interpolation function for graph fitness using a GA is described. Section 3 briefly describes the code, presents the list of parameters of the GA implementation and experiment results. Finally, some problems that deserve to be noted in implementing this method and their mend are given in section 4.

2 The GA's Introduction

The GA searches the solution space of a function through the use of simulated evolution, i.e., the survival of the fittest strategy. In general, the fittest individuals of any population tend to reproduce and survive to the next generation, thus improving successive generations. However, inferior individuals can, by chance, survive and also reproduce. The GA has been shown to solve linear and nonlinear problems by exploring all regions of the state space and exponentially exploiting promising areas through mutation, crossover, and

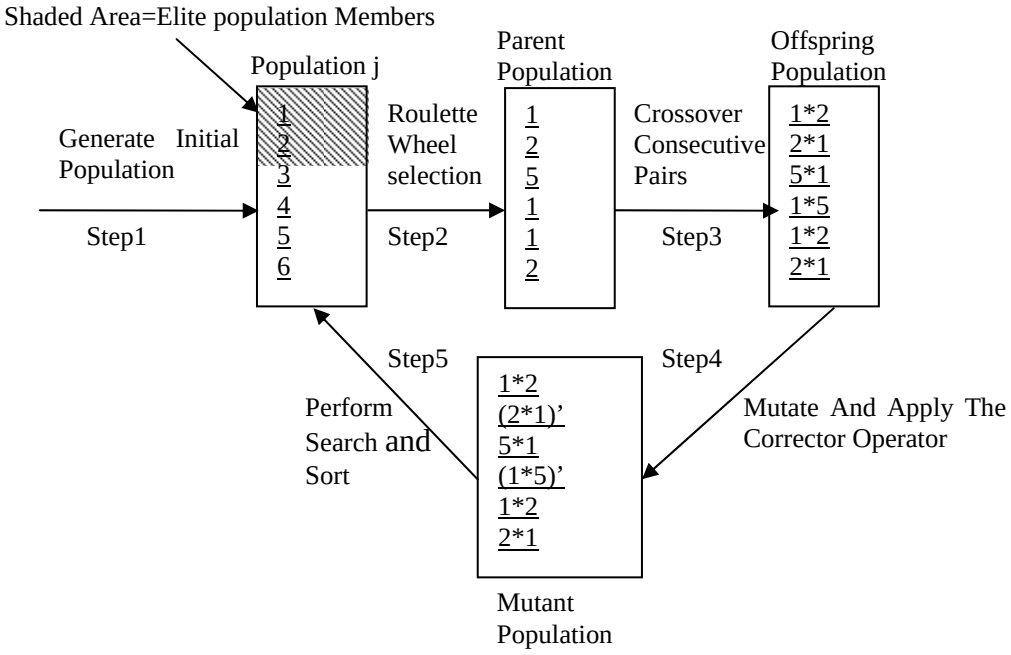

Fig.1 Schematic flow diagram for the GA program

selection operations applied to individuals in the population. A more complete discussion of genetic algorithms including extensions and related topics can be found in the books [10-13], and the Academic journals [14-23].

2.1 Characteristics of the GA

The GA is a search technique, based on the principles of natural evolution. The important terms and principles as follow:

(1) Codes on solution have evolution.

The codes of optimal problem solution are called chromosomes. Since the solution is coded, the research on optimization of function is based on codes. The one important topic of a GA is encoding and decoding.

(2) Law of natural selection decides which chromosomes have more offspring than others.

In the GA, fitness function is created by objective function, which will be optimized. The fitness functions ensure that the more chromosomes are fitting, the more offspring is generated.

(3) New chromosomes retain characteristics of parent chromosomes.

Crossover takes two chromosomes and produces two new chromosomes. The two new chromosomes retain characteristics of parent chromosomes from parent chromosomes gene.

(4) New chromosomes is different from parent chromosomes

Randomly mutant made the difference.

2.2 A Common Algorithm

The way in which our GA program operates is shown as a schematic flow diagram in Fig. 1. A basic procedure of GA is as follow.

It is a eight parameters function BGA.

BGA=(C,E,P0,M,So,Co,Mo,T)

In function, C -Individual coding method;

E-Evaluate fitness function;

P0-Initial population;

M- Initial population size;

So-Select operation;

Co-Crossover operation;

Mo-Mutation operation;

T- Termination conditions.

```
Procedure BGA
begin
    Initialize P(0);
    t=0;
    while (t≤T) do
        for I=1 to M do
            Evaluate fitness of P(t);
        End for
        for I=1 to M do
            Select operation to P(t);
        End for
        for I=1 to M/2 do
            Crossover operation to P(t);
        End for
        for I=1 to M do
            Mutation operation to P(t);
        End for
        for I=1 to M do
            P(t+1)= P(t);
        End for
        t=t+1;
    end while
end
```

The GA procedure is summarized as follow.

Step(1) Supply an initial population P(0) of N individuals and respective codes of function solutions, t: =0; (like step 1 of Fig.1)

Step (2) Calculate each of chromosomes Pi(t), which is in population P(t), fitness function

$$f_i = \text{fitness}(\text{Pi}(t));$$

Step (3) Calculate each of chromosome selection on probability

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}, \; i=1,2,\cdots, \; M, \qquad (6)$$

By p_i, a new population is generated from P(t)

newpop(t+1)={ $P_j(t)|j=1,2,\cdots, N$};

Remark: A chromosome in population P(t) may be repeatedly selected.

Step (4) Generate a population crosspop (t+1) by crossing consecutive pairs chromosomes from newpop(t+1). Crossing probability is Pc.

Step (5) Mutate a gene of a chromosome by small probability p. Generate a population mutpop(t+1), t:=t+1, a new population P(t)= mutpop(t) .

Step (6) Repeat step (3) until termination

Step (7) Print out best solution found

2.3 Fundamental Issues

The use of a genetic algorithm requires the determination of six fundamental issues: the creation of the initial population, chromosomes representation, the selection method of the newpop population, the genetic operators making up the crossover function, the mutant method, and

termination criteria. The rest of this section describes each of these issues.

Issue1. Initial population.

The GA must be provided an initial population as indicated in step (1). The most common method is to randomly generate solutions for the initial population. However, since GAs can iteratively improve existing solutions (i.e., solutions from other heuristics and/or current practices), the beginning population can be seeded with potentially good solutions, with the remainder of the population being randomly generated solutions.

Issue2. Chromosomes representation

For any GA, a chromosome representation is needed to describe each individual in the population of interest. The representation scheme determines how the problem is structured in the GA and also determines the genetic operators that are used. Each individual or chromosome is made up of a sequence of genes from a certain alphabet. An alphabet could consist of binary digits (0 and 1), floating point numbers, integers, symbols (i.e., A, B, C, D), matrices, etc. One useful representation of an individual or chromosome for function optimization involves genes or variables from an alphabet of floating point numbers with values within the variables upper and lower bounds.

Issue3. The selection method of the newpop population

A common selection approach assigns a probability of selection, Pi, to each individual, i based on its fitness value as indicated in step(2)(3). The chromosome, which has larger fitness function value, has more chance to be selected in new population. This selection method is call roulette wheel selection.

Issue4. The genetic operators making up the crossover function

Crossover (or mating) is the way in which "genetic" information from two parent chromosomes is combined to generate "offspring". In step(4), the parent chromosomes and crossing size are randomization.

Issue5. The mutant method

While the crossover operation leads to a mixing of genetic material in the offspring, no new genetic material is introduced, which can lead to lack of population diversity and eventually "stagnation"--where the population converges on the same, nonoptimal solution. The GA mutation operator helps to increase population diversity by introducing new genetic material. The common method is select one genetic to mutate by minimal probability.

Issue6. Termination criteria

The GA moves from generation to generation selecting and reproducing parents until a termination criterion is met. The most frequently used stopping criterion is a specified maximum number of generations. Another termination strategy involves population convergence criteria. In general, GAs will force much of the entire population to converge to a single solution. When the sum of the deviations among individuals becomes smaller than some specified threshold, the algorithm can be terminated.

3 Steps for a GA's Application and Analysis

Steps for a GA's application on AFIF for graph fitness are as follow:

Step1. Select a graph as a original graph. Compress axis of abscissas in the interval [0,1]. Remain aspect ratio.

Step2. Bisect axis of abscissas into EXN intervals. Let Terminal of every interval is interpolation points EXP_i, i=0,1,2……EXN. Select suitable vertical factor Exd_0(exi),exi=1,2,……EXN. Encode the vertical factors in EXL bits binary code. We use iteration method of AFIF to draw fractal graph. Select EXM sample point in every interval. Calculate sample variance EXS_0.

In this step, binary code for vertical factors of each intervals is genes, which is respective codes of function solutions. Geneses are joined as chromosome.

Step3. Repeat step 2 for EXN_c times. Select randomly vertical factors Exd_{exj}(exi)(exj is repeat times) in every times. Calculate sample variance EXS_j.

In this step, we have EXN_c pieces of chromosomes. Those chromosomes created initial population.

Step4. The initial population has EXN_c chromosomes. Every chromosome is described by vertical factors codes. Length of a chromosome is EXN*EXL. The fitness function of the chromosome No j is $EXF_j=1/ EXS_j$. From Section 2, we can evaluate fitness of the initial population. From select operation, which is based on evaluate fitness, we can create new population for evolution, than the population use crossover operation and mutation operation to create next population. Those operations will be repeated until terminal.

In this step, the smaller sample variance of chromosome is, the more selection on probability of chromosome is. The smaller sample variance is, the more fitness created graph and original graph is. The terminal conditions are very flexibility. Setting terminal condition decided for the time complexity of algorithms is important. Usually, we can set mutation times as terminal condition.

Step5. From suitable vertical factors and AFIF Interpolation points, we can create optimization fitness graph for original graph.

4 Experiments and Result

In this experiment, we use VB.NET to realize those steps.

4.1 Initial Population

Let intervals number EXN=8, length of vertical factors code EXL=4, sample point in every interval EXM=16, chromosomes of population number EXN_c=20.Here we give the original graph A as fig.2, which has been compressed. to obtain EXN interpolation points, EXP_i, i=0,2…EXN. By step2, their axes are as table 1.

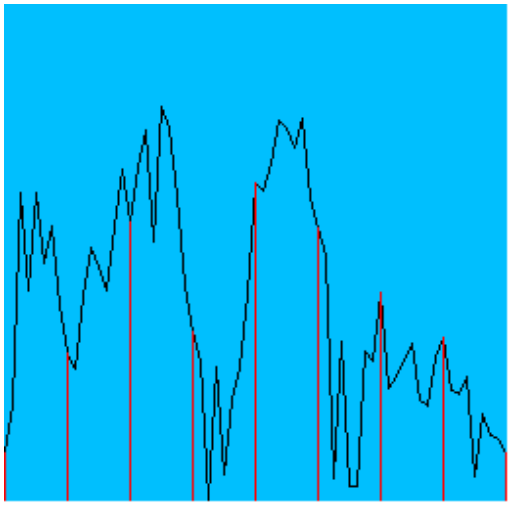

Fig.2 Original graph A

Table 1 The axes of interpolation points

EXP_0=(0,0.1)	EXP_1=(0.125,0.3)
EXP_2=(0.25,0.56)	EXP_3=(0.375,0.34)
EXP_4=(0.625,0.64)	EXP_5=(0.5,0.55)
EXP_6=(0.75,0.42)	EXP_7=(0.875,0.33)
EXP_8=(1,0.1)	

From randomization algorithm, we get vertical factors $Exd_0(exi)$(exi=1,2,……*EXN*) for each interval. For example as table 2,

Table 2 The vertical factors

$Exd_0(1)$=0.559	$Exd_0(2)$=0.677	$Exd_0(3)$=-0.573
$Exd_0(4)$=-0.574	$Exd_0(5)$=0.18	$Exd_0(6)$=0.803
$Exd_0(7)$=-0.448	$Exd_0(8)$=0.18	

Vertical factors must satisfy in formula (2). By function $x'=(x+1)/2$ ($x \in [-1,1]$), we can translate vertical factors in the interval [0,1]. Translate decimal to binary. We make up vertical factors code by four binary digital after decimal point. For example, $Exd_0(1)$=0.559, the code is 1100. A chrom- osome of population is made up by orderly jointing $Exd_{exj}(1)$ to $Exd_{exj}(8)$. For example, orderly jointing $Exd_0(1)$ to $Exd_0(8)$ is 110011010011001110011110 01001001, which is a chromosome code. The function Dtob is to translate decimal to binary as follow.

```
Function Dtob(ByVal x As Double) As String
Dim temp As Double, i As Integer
temp = (x + 1) / 2
Dtob = ""
For i = 1 To 4
Dtob = Dtob & Fix(temp * 2)
temp = temp * 2 - Fix(temp * 2)
Next
End Function
```

In this function, Parameter x is source decimal value. Binary value is string.

From AFIF and factors, which are described here_ inbefore, we can generate fractal graph. Calculate sample variance. The sub Bfit_Click is to create fitness graph by AFIF and draw it as follow.

```
Private Sub Bfit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Bfit.Click
    Dim i As Integer, k As Integer, x As Double, y As Double
    Dim j As Integer, s As Integer
    Dim pen As New Pen(Color.FromArgb(255, 255, 255, 255))
    Dim g As System.Drawing.Graphics, g2 As System.Drawing.Graphics
    Dim x1 As Integer, x2 As Integer, y1 As Integer, y2 As Integer
    rst(rsts) = ""
    For i = 1 To 8
      rst(rsts) = rst(rsts) & ds(i)
    Next
    k = 0
    p(0).x = inx(0)
    p(0).y = iny(0)
```

```
        For i = 1 To 8
           For j = 1 To 8
              k = k + 1
              x = k / 64
              y = fny(i, inx(j), iny(j))
              p(k).x = x
              p(k).y = y
           Next
        Next
        For i = 0 To 63
           x1 = Int(p(i).x)
           y1 = n - Int(p(i).y)
           x2 = Int(p(i + 1).x)
           y2 = n - Int(p(i + 1).y)
           g = Pic1.CreateGraphics
           g.DrawLine(pen, x1, y1, x2, y2)
           g2 = Pic3.CreateGraphics
           g2.DrawLine(pen, x1, y1, x2, y2)
        Next
        dis(rsts) = 0
        For i = 1 To 64
           dis(rsts) = dis(rsts) + (p(i).y - pr(i).y) ^ 2
        Next
        dis(rsts) = dis(rsts) / 64
        rsts = (rsts Mod 20) + 1
    End Sub
```

In this sub, function fny is as follow.

```
    Function fny(ByVal n As Integer, ByVal x As Double, ByVal y As Double) As Double
        c(n) = (iny(n) - iny(n - 1) - d(n) * (iny(8) - iny(0))) / (inx(8) - inx(0))
        h(n) = iny(n) - c(n) * inx(8) - d(n) * iny(8)
        fny = c(n) * x + d(n) * y + h(n)
    End Function
```

The initial population we selected as table 3. The initial population size is 20.

For operating easliy, we array the initial population from little value to large value by fitness, which countdown to sample variance EXS.

```
    For i = 1 To 19
       l = i
       temp = fitness(i)
       For j = i + 1 To 20
       If temp > fitness(j) Then
          temp = fitness(j)
          l = j
       End If
       Next
       If l <> i Then
          fitness(l) = fitness(i)
          fitness(i) = temp
          temps = rst(l)
          rst(l) = rst(i)
          rst(i) = temps
          temp = dis(l)
          dis(l) = dis(i)
          dis(i) = temp
          temp = gl(l)
          gl(l) = gl(i)
          gl(i) = temp
       End If
    Next
```

Table 3 The initial population

No	Chromosome	*EXS*
1	01001001110100100011010100101110	0.0568
2	01000001100001111101000101110101	0.0544
3	10101100011101001010110100000000	0.0682
4	11010010110001111010111010000110	0.0626
5	01001101101101000111110000100010	0.0706
6	01000101011010010010100001100111	0.0784
7	10001010101111000011100100001101	0.0825
8	11010110101010010000011010001001	0.0526
9	10011010110001110001000100001011	0.0471
10	11010001001010100010001010000011	0.0784
11	00111100000101111110000000011101	0.0905
12	10101101100111010110100010101101	0.0783
13	01111001000100001010110011101001	0.0925
14	11000101000011000111100101110101	0.0879
15	11011101101010100000010110100001	0.0672
16	00010011000011010001011000110111	0.1422
17	00101010101000101011001101111100	0.0476
18	01110110100111101000000000011000	0.0624
19	10101011111000101001000110011110	0.0265
20	00010111110111100110110100101010	0.1238

4.2 Population Evolution

By using evaluate fitness of initial population and select operation to initial population we can create new population.

Let crossing probability P_c=1, mutant probability p=0.05. Population evaluates 40 generations. The procedure of crossover operation to new population is as follow.

```
    k = 1
    For i = 1 To 10
       Randomize()
       wz = Int(Rnd() * 18 + 2)
       xh = Int(Rnd() * 19 + 1)
       Do While newpop(xh) = ""
              xh = Int(Rnd() * 19 + 1)
       Loop
```

```
        crosspop(k) = newpop(xh)
        newpop(xh) = ""
        temps = Mid(crosspop(k), wz)
        crosspop(k) = Mid(crosspop(k), 1, wz - 1)
        k = k + 1
        xh = Int(Rnd() * 19 + 1)
        Do While newpop(xh) = ""
              xh = Int(Rnd() * 19 + 1)
        Loop
        crosspop(k) = newpop(xh)
        newpop(xh) = ""
        crosspop(k - 1) = crosspop(k - 1) & Mid(crosspop(k), wz)
        crosspop(k) = Mid(crosspop(k), 1, wz - 1)
        crosspop(k) = crosspop(k) & temps
        k = k + 1
    Next
```

In this basic procedure, we use random function to select chromosomes, which is used to crossover. But we found if we only use random function to operate crossover operation to twenty pieces of chromosomes is inefficient. We use procedure to operate crossover operation as follow.

```
    k = 1
    For i = 1 To 9
       Randomize()
       wz = Int(Rnd() * 18 + 2)
       xh = Int(Rnd() * 19 + 1)
       Do While newpop(xh) = ""
    xh = Int(Rnd() * 19 + 1)
       Loop
       crosspop(k) = newpop(xh)
       newpop(xh) = ""
       temps = Mid(crosspop(k), wz)
       crosspop(k) = Mid(crosspop(k), 1, wz - 1)
       k = k + 1
       xh = Int(Rnd() * 19 + 1)
       Do While newpop(xh) = ""
    xh = Int(Rnd() * 19 + 1)
       Loop
       crosspop(k) = newpop(xh)
       newpop(xh) = ""
       crosspop(k - 1) = crosspop(k - 1) & Mid(crosspop(k), wz)
       crosspop(k) = Mid(crosspop(k), 1, wz - 1)
       crosspop(k) = crosspop(k) & temps
       k = k + 1
     Next
     For i = 1 To 20
        If newpop(i) <> "" Then
           crosspop(k) = newpop(i)
           newpop(xh) = ""
           k = k + 1
        End If
     Next
        wz = Int(Rnd() * 18 + 2)
        temps = Mid(crosspop(19), wz)
        crosspop(19) = Mid(crosspop(19), 1, wz - 1)
        crosspop(19) = crosspop(19) & Mid(crosspop(20), wz)
        crosspop(20) = Mid(crosspop(20), 1, wz - 1)
        crosspop(20) = crosspop(20) & temps
```

In this procedure, we use random function to operate crossover operation to eighteen pieces of chromosomes.

The procedure of mutation operation to crossover population is as follow.

```
    xh = Rnd() * 19 + 1
    wz = Rnd() * 19 + 1
    temps = Mid(crosspop(xh), wz, 1)
    If temps = "1" Then
        temps = "0"
    Else
       temps = "1"
    End If
    s1 = Mid(crosspop(xh), 1, wz - 1)
    s2 = Mid(crosspop(xh), wz + 1)
    crosspop(xh) = s1 & temps & s2
```

In those procedures,

xh-Number of chromosome in array;

wz-Locaion of a gene in a piece of chromosome.

Those two Variables are valued by random function.

We translate each four binary digitals of binary code of new generation chromosomes to decimal, which can be translated into the interval [-1,1] from function $x' = 2x - 1 (x \in [0,1])$. That is vertical factor for each interval. The function Btod is to translate binary to decimal as follow.

```
    Function Btod(ByVal x As String) As Single
    Dim temp(4) As Single, qz(4) As Single, i As Integer
          Btod = 0
          For i = 1 To 4
             temp(i) = Val(Mid(x, i, 1))
             qz(i) = 1 / 2 ^ i
             temp(i) = temp(i) * qz(i)
             Btod = Btod + temp(i)
          Next
          Btod = 2 * Btod - 1
    End Function
```

For example, x equal 1110 , the vertical factor is 0.75.

The complete generation evolution procedure is as follow.

```
Sub gene()
Dim i As Integer, j As Integer, temp As Single, temps As String, l As Integer, total As Single, k As Integer
Dim xh As Integer, wz As Integer, s1 As String, s2 As String
    For i = 1 To 20 'Calculate fitness
      fitness(i) = 1 / dis(i)
      total = total + fitness(i)
    Next
    For i = 1 To 20 ' Calculate
      gl(i) = fitness(i) / total
      pgl(i) = 2 * i / 21 + 0.5 'For Rounding easily
    Next
    For i = 1 To 19 'Create array sequence
      l = i
      temp = fitness(i)
      For j = i + 1 To 20
        If temp > fitness(j) Then
          temp = fitness(j)
          l = j
        End If
      Next
      If l <> i Then
        fitness(l) = fitness(i)
        fitness(i) = temp
        temps = rst(l)
        rst(l) = rst(i)
        rst(i) = temps
        temp = dis(l)
        dis(l) = dis(i)
        dis(i) = temp
        temp = gl(l)
        gl(l) = gl(i)
        gl(i) = temp
      End If
    Next
    k = 1
    For i = 1 To 20 'Create new population newpop by evaluate fitness
      If Int(pgl(i)) >= 1 Then
        For j = 1 To Int(pgl(i))
          newpop(k) = rst(i)
          k = k + 1
        Next
      End If
    Next
    k = 1
    For i = 1 To 9
      Randomize()
      wz = Int(Rnd() * 18 + 2)
      xh = Int(Rnd() * 19 + 1)
      Do While newpop(xh) = ""
        xh = Int(Rnd() * 19 + 1)
      Loop
      crosspop(k) = newpop(xh)
      newpop(xh) = ""
      temps = Mid(crosspop(k), wz)
      crosspop(k) = Mid(crosspop(k), 1, wz - 1)
      k = k + 1
      xh = Int(Rnd() * 19 + 1)
      Do While newpop(xh) = ""
        xh = Int(Rnd() * 19 + 1)
      Loop
      crosspop(k) = newpop(xh)
      newpop(xh) = ""
      crosspop(k - 1) = crosspop(k - 1) & Mid(crosspop(k), wz)
      crosspop(k) = Mid(crosspop(k), 1, wz - 1)
      crosspop(k) = crosspop(k) & temps
      k = k + 1
    Next
    For i = 1 To 20
      If newpop(i) <> "" Then
        crosspop(k) = newpop(i)
        newpop(xh) = ""
        k = k + 1
      End If
    Next
    wz = Int(Rnd() * 18 + 2)
    temps = Mid(crosspop(19), wz)
    crosspop(19) = Mid(crosspop(19), 1, wz - 1)
    crosspop(19) = crosspop(19) & Mid(crosspop(20), wz)
    crosspop(20) = Mid(crosspop(20), 1, wz - 1)
    crosspop(20) = crosspop(20) & temps
    xh = Rnd() * 19 + 1 ' Let crossing probability Pc=1, mutant probability p=0.05.Change one gene of a piece of chromosome.
    wz = Rnd() * 19 + 1
    temps = Mid(crosspop(xh), wz, 1)
    If temps = "1" Then
      temps = "0"
    Else
      temps = "1"
    End If
    s1 = Mid(crosspop(xh), 1, wz - 1)
    s2 = Mid(crosspop(xh), wz + 1)
    crosspop(xh) = s1 & temps & s2
    For i = 1 To 20 ' Next  population Evolution
      rst(i) = crosspop(i)
```

```
        Next
    End Sub
```

By step4, we can generate the final population. The procedure is as follow.

```
Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button3.Click
Dim i As Integer, j As Integer, k As Integer, e1 As System.EventArgs, obj1 As System.Object
        For i = 1 To genm
            For j = 1 To 20
                For k = 1 To 8
                    ds(k) = Mid(rst(j), (k - 1) * 4 + 1, 4)
                    d(k) = Btod(ds(k))
                Next
            Next
            gene()
        Next
        ListBox2.Items.Clear()
        For i = 1 To 20
            ListBox2.Items.Add(rstl(i))
        Next
End Sub
```

In this procedure, element of array ds is vertical factors binary code, element of array d is vertical factors' decimal value. We can see the final twenty pieces of chromosome of population in ListBox2.

The result of evolution is as table 4.

Table 4 The final population

No	Chromosome	*EXS*
1	11101001110000101000000101110101	0.0062
2	11101001110000101010000101110101	0.0049
3	11101001110000101010000101110101	0.0049
4	11101001110000101010000101110101	0.0049
5	11101001110000101010000101110101	0.0049
6	11101001110000101010000101110101	0.0049
7	11101001110000101010000101110101	0.0049
8	11101001110000101010000101110101	0.0049
9	11101001110000101010000101110101	0.0049
10	11101001110000101010000101110101	0.0049
11	11101001110000101010000101110101	0.0049
12	11101001110000101010000101110101	0.0049
13	11101001110000101010000101110101	0.0049
14	11101001110000101010000101110101	0.0049
15	11101001110000101010000101110101	0.0049
16	11101001110000101010000101110101	0.0049
17	11101001110000101010000101110101	0.0049
18	11101001110000101010000101110101	0.0049
19	11101001110000101010000101110101	0.0049
20	11101001110000101010000101110101	0.0049

From table 4, we can have the final optimization chromosome. Translate chromosome to eight vertical factors is as follow:

Exdf(1)=1110=0.75;
Exdf(2)=1001=0.125;
Exdf(3)=1100=0.5;
Exdf(4)=0010=-0.75;
Exdf(5)=1010=0.25;
Exdf(6)=0001=-0.875;
Exdf(7)=0111=-0.125;
Exdf(8)=0101=-0.375;

Using Vertical factors and interpolation points as table 1, we can create the optimization 64 sample points. The procedure of drawing graph by 64 sample points is as follow.

```
For i = 0 To 63
        x1 = Int(p(i).x)
        y1 = n - Int(p(i).y)
        x2 = Int(p(i + 1).x)
        y2 = n - Int(p(i + 1).y)
        g = Pic1.CreateGraphics
        g.DrawLine(pen, x1, y1, x2, y2)
        g2 = Pic3.CreateGraphics
        g2.DrawLine(pen, x1, y1, x2, y2)
Next
```

The optimization graph is as fig.3.

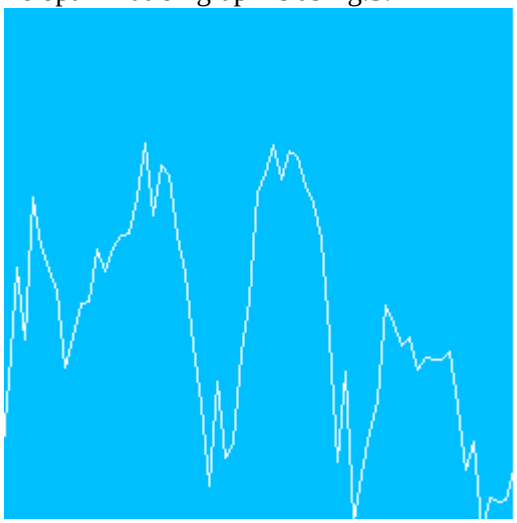

Fig.3 Optimization graph of AFIF a

4.3 Experimental Result

Comparing original graph A with optimization graph a as fig.4, we can find that coupling GA with AFIF yields optimization solution for graph fitness.

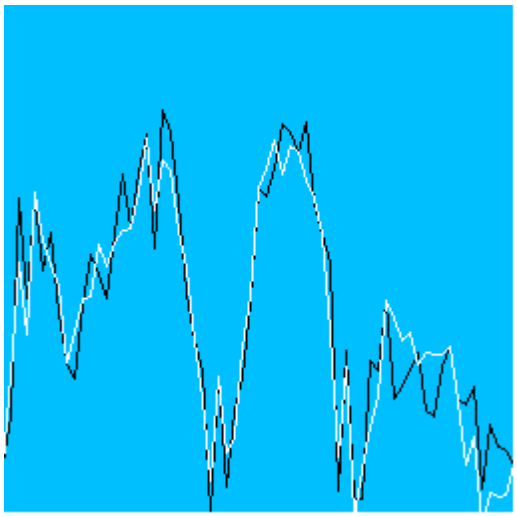

Fig.4 Original graph A and optimization graph a

5 Conclusion

The results obtained in this study show that GAs are powerful variable selection tools for graph fitness. Coupling GA with AFIF yields more precise predictions than other methods. Application of an AFIF, as a flexible nonlinear calibration, to the data selected by GA also improves the performance of the model considerably.

It is expected that other methods can improve GA's application. Some problems deserve to be noted in our experiment. The sample space size of the initial population must be appropriate. We should use schema theorem to generate the initial population. If selection method isn't suitable, convergence will be quick. The result of GA may not be the best. Using simulated annealing algorithm may be useful. Termination criteria should be carefully studied. For example, we can use sample variance as the termination condition. Anyway, the great merits of our proposed methods are low costing, simple, and rapid.

References:

[1] Lai K K, Chan J W M, *Developing a simulated annealing algorithm for the cutting stock problem*, Computers Ind, 1996.

[2] Glover f, Tabu search : part II, *ORSA Journal on Computing*, Vol.2, 1990, pp.4–32.

[3] Xinjun Yang, Junli Zheng, *Artificial neural networks*, Higher Education Press, 1992.

[4] Wenxun Xin, Jinxing Xie, *Modern Computing method for Optimization*, Tsinghua University Press, 1999.

[5] Horm J,et, A Niched Pareto Genetic Algorithm for Multiobjective Optimization, *Proc of 1st IEEE Conf on Evolutionary Computation*,1993, pp.82 – 87.

[6] Brindle A, *Genetic Algorithms for Function Optimizations*, University of Alberta, 1981.

[7] Fogel D B, An Introduction to Sumulated Evolutionary Optimization, *IEEE Trans. On Neural Networks*, Vol.5, No.1, 1994, pp.3–14.

[8] Barnsley, M F, Fractal functions and Interpolation, *Constr, Approx*, No. 2, 1986, pp. 303–329.

[9] Dalla L, Drakopoulos V, On the parameter identification problem in the plane and the polar fractal interpolation functions, *Approx. Theory*, Vol.101, 1999, pp.289–202.

[10] Davis L,*The Handbook of Genetic Algorithms*, Van Nostrand Reingold, 1991.

[11] Migdalas A, Pardalos P M, Varbrand P. *Multilevel Optimization: Algorithms and Applications*, Dordrecht: Kluwer Academic Publishers, 1998.

[12] Sheng Zhaohan, *Hierarchical Decision System: Stackelberg Problem*, Science Press, 1998.

[13] Golberg D E, *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading, MA: Addison-Westey, 1989.

[14] Bard J F, Moore J T, A Branch-and-Bound Algorithm for the Bilevel Programming Problem , *SIAM Journal on Scientific and Statistical Computing*, Vol.11, 1990, pp.281-292.

[15] Savard G, Gauvin J, The Steepest Descent Direction for the Nonlinear Bilevel Programming Problem, *Operations Research Letters*, Vol.15, 1994, pp.265-272.

[16] Vicente L, Savarg G, Judice J, Descent Approaches for Quadratic Bilevel Programming, *Journal of Optimization Theory and Applications*, Vol.81, 1994, pp.379-399.

[17] Bard J F, Convex Two-level Optimization, *Mathematical Programming*, Vol.40, 1988, pp.15-27.

[18] Hejazi S R, Memariani A, Jahanshanloo G, et al. Linear Bilevel Programming Solution by Genetic Algorithm, *Computers & Operations Research*, Vol.29, 2002, pp.1913-1925.

[19] Edmunds T, Bard J F, Algorithms for Nonlinear Bilevel Mathematical Programming, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol.21, 1991, pp.83-89.

[20] Muu L D, Quy N V, A Global Optimization Method for Solving Convex Quadratic Bilevel Programming Problems, *Journal of Global Optimization*, Vol.26, 2003, pp.199-219.

[21] Mastorakis, Nikos E, The Singular Value Decomposition (SVD) in tensors

(multidimensional arrays) as an optimization problem. Solution via Genetic Algorithms and method of Nelder-Mead, *WSEAS Transactions on Systems*, Vol.6, No.1, 2007, pp. 17-23.

[22] 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, No.3, 2005, pp. 225-232.

[23] Karnavas Yannis L, On the optimal control of interconnected electric power systems in a re-structured environment using genetic algorithms, *WSEAS Transactions on Systems*, Vol.4, No.8, 2007, pp. 1248-1258.