Research on Performance Optimization of Several Frequently-Used Genetic Algorithm Selection Operators

Qili Xiao(✉), Jiqiu Li, and Changfan Xiao

Business School, Ningbo College of Vocational Technology, Ningbo, China
843848135@qq.com

Abstract. Genetic Algorithm is an intelligent algorithm for simulation of biological evolution, is widely applied to solve all kinds of problems. In this paper, several Frequently-used selection operators of Genetic Algorithm are programmed by C language, and are tested in an optimization problem.

Keywords: Genetic algorithm · Selection operator · Sierpinski carpet · Hausdorff measure

1 Introduction

Genetic Algorithm (GA) is a tool for computer engineers to simulate biological evolution. It is a probability search algorithm proposed by Professor John Holland of Michigan University. It makes use of the simple coding mechanism and the genetic mechanism of the natural organism to represent the complex phenomenon, so as to solve the difficult problem. GA has strong robustness and a wide range of applications. When we use GA to solve problems, it is not restricted by the restrictive assumption of the search space and it is not necessary to assume such as continuity, derivative existence and unimodal [1]. Using GA to solve practical problems, there are three main steps, namely encoding and decoding, the calculation of individual fitness, genetic operations. Genetic operations include selection operation, crossover operation and mutation operations. From the viewpoint of GA, the evolution of solution is completed by depending mainly on the selection mechanism and the crossover strategy. And the mutation is the repair and supplement of some genetic genes that may be lost in the process of the selection and crossover. For the overall situation of GA, mutation only is a basic operation. The crossover operation is based on the result of the selection operation, namely the object of the cross operation is the result of the selection operation. It can be seen that, in the process of using GA to solve practical problems, the selection operation occupies an important position and it is also a major factor in determining the convergence of GA. C language has many functions, such as rich functions, strong expression ability, flexible use, wide application and good portability. In this paper, the functions on several Frequently-used operator selection are programmed by C language and are tested in an optimization problem.

J. Abawajy et al. (eds.), *International Conference on Applications and Techniques in Cyber Security and Intelligence*, Advances in Intelligent Systems and Computing 580,
DOI 10.1007/978-3-319-67071-3_14

Frequently-used selection operations of GA mainly are the proportion choice, the strategy of preservation of the best individual, deterministic sampling and so on. In the following selection operator functions, the input parameters are *pop* and *popfitness*. The parameter *pop* is a two-dimensional array, which is used to represent the population. Each row of the parameter *pop* represents an individual coded by binary. The parameter *popfitness* is a one-dimensional array that is used to represent the fitness of each individual in the population. In this paper, the symbolic constant *popsize* is assumed to be the number of individuals in a group. And the symbolic constant *chromlength* is assumed to be the encoding length of the individual.

2 Proportional Selection

The method of proportional selection is also called the roulette wheel selection method. In this method, the being selected probability of each individual and the fitness of each individual is proportional. The individual fitness is higher, the greater the probability of being selected. Because the method is simple and easy to implement, it is the most frequently-used selection method of GA. The method of proportional selection that is programmed with C is as follows.

```
void SelectOperator()
{sum= 0;
 //Calculate Relative fitness
 for(i= 0; i< popsize;i++){sum= sum+ popfitness[i];}
 for(i= 0; i< popsize;i++){relative_profit[i]= popfitness[i]/ sum;}
 for(i= 1; i< popsize;i++)     //Calculate cumulative probability
   relative_profit[i]= relative_profit[i- 1] + relative_profit[i];
 for(i= 0; i< popsize;i++)       //Create a new group newpop
   {    p= rand()%1000/1000.0;
        k= 0;
        while(p> relative_profit[k]) {k++ ;}
      for(j= 0; j< chromlength;j++) {newpop[i][j]= pop[k][j];}
    }
}
```

In the above function, *relative_profit* expresses the relative fitness of individuals, and *newpop* expresses a new generation groups.

3 Save the Best Individual Strategy

In the process of using GA to solve problems, more and more excellent individuals will be produced with the evolutionary process of population. But because of randomness of genetic operations, such as selection, crossover, mutation and so on, the best individual in the current population is likely to be destroyed, so as to reduce the population average fitness and influence GA's operating efficiency and convergence speed. We also often retain the individual with the best fitness to the next generation, namely in the current population, the individual with the highest fitness, is not involved in the crossover operation and mutation operation. The individual with the highest fitness, will replace the individual with the lowest fitness. The steps are as follows.

(1) Find the individual with the highest fitness and the individual with the lowest fitness in the current population.
(2) If the fitness of the best individual in the current population is higher than that of the best individuals so far, the best individual in the current population is the best person to date.
(3) Replace the worst individuals in the current group with the best individuals so far.

The specific implementations of each of the above steps are as follows.

(1) Find out the best and worst of all

```
void findbestandworstindividual()

{for(j= 0;j< chromlength;j++)

      {bestchrom[j]= pop[0][j]; worstchrom[j]= pop[0][j];}

bestfitness= popfitness[0]; worstfitness= popfitness[0]; bestflag= 0; worstflag= 0;

for(i= 1;i< popsize;i++)

      { if(popfitness[i]> bestfitness)

          {bestfitness= popfitness[i]; bestflag= i;}

      else if(popfitness[i]< worstfitness)

          {worstfitness= popfitness[i]; worstflag= i;}

    }

for(j= 0;j< chromlength;j++)

  {bestchrom[j]= pop[bestflag][j]; worstchrom[j]= pop[worstflag][j];}

}
```

(2) Find out the best individual so far

```
void findcurrentbestindividual()
{ if(gen== 0)      //gen is the number of evolution
   {  currentbestfitness= bestfitness;
      for(j= 0 ; j< chromlength ; j+ +) currentbestchrom[j]= bestchrom[j];
  }
 else
  { if(bestfitness> currentbestfitness)
     {  currentbestfitness= bestfitness;
        for(j= 0 ; j< chromlength ; j+ +) currentbestchrom[j]= bestchrom[j];
      }
    }
}
```

(3) Replace the worst individual with the best individual

```
void performevolution(void)
{for(j= 0;j< chromlength;j++)
   {pop[worstflag][j]= currentbestchrom[j];
    popfitness[worstflag]= currentbestfitness;
   }
}
```

In the above function, the variables *bestchrom* and *worstchrom* respectively express the best individual and the worst individual. The variables *bestfitness* and *worstfitness* respectively express the fitness of the best individual and the fitness of the worst individual. The variables *bestflag* and *worstflag* respectively express the index of the best and the worst individual. The variable *currentbestfitness* expresses the fitness of best individual so far.

In fact, saving the best individual strategy is generally regarded as a part of the selection operation. And it is often with other methods to achieve the selection operation. The research shows that, the standard GA using proportion selection is not convergent. And the GA, with saving the best individual strategy, will converge to the global optimum solution [2].

4 Deterministic Sampling Selection

Using the above two methods to select individuals, the random of selection operation is very strong, and do not depend on one's will to change. Deterministic sampling selection method can artificially control the selection operation of the individual, and its basic idea is to select according to a definite way. The specific operation process is as follow.

(1) Calculate the survival number of each individual that will be in the next generation N_i.
(2) the survival number of each individual in the next generation is determined by the integral part of the N_i. $\sum_{i=1}^{M}[N_i]$ of the next generation is determined by this step, where M is the number of individuals in the population.
(3) the individuals will be descending sorted according to the decimal part of the N_i. And the first $(M - \sum_{i=1}^{M}[N_i])$ individuals will be put into the next generation.

Specific coding is as follows.

```
void SelectOperator()
{  sum= 0;
   //Calculate the parameter savenum[i] that is the expected survival number of each individual in the next generation.
    for(i= 0; i< popsize;i+ +){sum= sum+ popfitness[i];}
    for(i= 0; i< popsize;i+ +){savenum[i]= popsize*popfitness[i]/sum;}
   //generate the next generation
    k= - 1;
  for(i= 0; i< popsize;i++)
  { p=(int)savenum[i] ;
    if(p> 0)
    { for(j= 1;j< = p;j++)
      {
     k++ ;
        for(r= 0;r< chromlength;r++)newpop[k][r]=pop[i][r];
      }
    }
  }
for(i= 0; i< popsize;i++)
   { savenum[i]= savenum[i]-(int)savenum[i];  index[i]= i; }
for(i= 0; i< popsize- 1;i++)
    for(j= i+ 1; j< popsize;j++)
      { if(savenum[i]> savenum[j])
         {  temp= savenum[ i]; savenum[ i]= savenum[ j];savenum[j]= temp;
           p= index[i]; index[i]= index[j]; index[j]= p;
        }
     }
j= k+ 1;
for(i= 0; i< popsize- j;i++)
   {  k++ ;   for(r= 0;r< chromlength;r++) newpop[ k][ r]= pop[index[i]][r]; }
}
```

In the above function, the parament *savenum* is the expected survival number of individuals in the next generation. The parament *newpop* is the new group.

5 Application

Take a unit square in the Euclidean place 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 expect for the four ones lying on the vertexes of, 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. 1), we have $F_0 \supset F_1 \supset \cdots \supset F_k \supset \cdots$. The non-empty set $F = \bigcap_{k=U}^{\infty} F_k$ is called the Sierpinski carpet yielded by F_0. The Sierpinski carpet is the result that the following four functions are applied on a unit square F_0.

$$S_1 = x/4, \quad S_2 = x/4 + (3/4, 0), \quad S_3 = x/4 + (3/4, 3/4),$$
$$S_4 = x/4 + (0, 3/4)$$

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

For the Sierpinski carpet, the Hausdorff measure of the Sierpinski carpet $H(F) = \lim_{m \to \infty} \inf_{U \in F_m'} \frac{|U|}{\mu(U)}$. The Hausdorff measure of the Sierpinski carpet is the value of $\inf_{U \in F_m'} \frac{|U|}{\mu(U)}$ when $m \to \infty$ [3]. In fact, for a fixed finite m, if the exhaustive method is used, the calculation workload of $\inf_{U \in F_m'} \frac{|U|}{\mu(U)}$ is 2^{4^m}. In order to avoid large-scale mathematical calculation, GA can be used to solve the approximate Hausdorff measure value of Sierpinski carpet. In the experiment, we use the above three kinds of selection operator to solve the approximate Hausdorff measure of Sierpinski carpet. Experimental results show that three kinds of GA all can be used to calculate the exact value when $m < 6$. But GA based on deterministic sampling method needs the shortest time, and the standard GA has the longest running time.

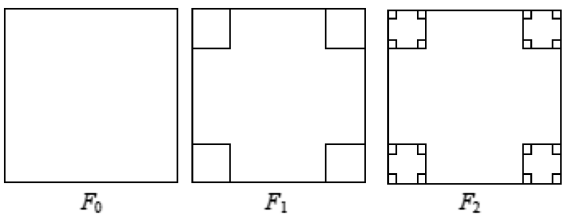

Fig. 1. The structure of the Sierpinski carpet

References

1. Wang, X., Cao, L.: Genetic Algorithm - Theory, Application and Software Implementation. Jiao Tong University Press, Xi'an (2002)
2. Chen, G., Wang, X., et al.: Genetic Algorithm and Its Application. People's Posts and Telecommunications Press, Beijing (1996)
3. Lifeng, X., Zhongdi, C.: Some frontier problems of fractal geometry-calculation of fractal measure. J. Zhejiang Wanli Univ. **1**, 1–3142 (2001)