Genetic Algorithm

Chongwu Ruan

Math 190S - Hubert Bray

July 31, 2017

1 Introduction

GA is adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics. The basic techniques of the GA are designed to simulate processes in natural systems necessary for evolution, including inheritance, mutation, natural selection, and hybridization, specially those follow the principles first laid down by Charles Darwin of "survival of the fittest.". Since in nature, competition among individuals for scanty resources results in the fittest individuals dominating over the weaker ones.

In this paper, we will talk about how genetic algorithms work and what the future of genetic algorithms will be like.

2 GA Overview

GA is a kind of computational algorithm used to solve the optimization problems in mathematics. For an optimization problem, a certain number of candidate solutions (called individuals) can be abstractly expressed as chromosomes, allowing the population to evolve to better solutions. Evolution begins with a completely random individual population, followed by generation after generation. In each generation, the fitness of the whole population is evaluated, and a number of individuals (based on their fitness) are randomly selected from the

current population, and a new life population is generated by natural selection and mutation. The population becomes the next iteration of the algorithm current population.

In this part, we will introduce the simple genetic algorithm (SGA), a basic kind of GAs. It include only selection, crossover and mutation operators. Other kind of GA is just based on SGA.

2.1 The Mathematic Model of SGA

$$SGA = (C, E, P_0, N, \Phi, \Gamma, \Psi, T) \tag{1}$$

C—Individual coding method

E—Individual fitness evaluation function

P_0—Initial population

N—Population size

Φ—Selection operator

Γ—Crossover operator

Ψ—Mutation operator

T—Termination of genetic operations

For a practical application problem that needs to be optimized, the genetic algorithm for finding the optimal solution is usually constructed as follows:

step 1 Determine the decision variables and their various constraints, that is, to determine the individual's phenotype and problem solution space.

step 2 Establish the optimization model, that is, determine the type of objective function (the maximum or minimum value) and its mathematical description of the form or quantification method.

step 3 Determine the chromosome coding method that represents the feasible solution, that is, determine the individual genotype and the search space of GA.

step 4 Determine the decoding method, that is, determine the corresponding relationship or conversion relationship from the individual genotype to the individual phenotype.

step 5 Determine the quantitative evaluation method of individual fitness, that is, determine the conversion rule from the objective function to the individual fitness.

step 6 Design the genetic operators, that is , determine selection, crossover and mutation operators.

step 7 Determine the relevant operating parameters of the genetic algorithm, that is, determine parameters P_c, N, P_m, T.

2.2 Fundamental Elements of SGA

2.2.1 Chromosome Coding Method

Since the object of the genetic algorithm is a chromosome coding string, it can not directly deal with the solution data in the solution space. The solution space must be expressed as the genotype string structure data of the genetic space. SGA uses a fixed length binary symbol string to represent an individual in the population whose allele (the gene at the same gene position) is composed of the binary symbol set $\{0, 1\}$. The gene values of individuals in the initial population can be generated using a uniformly distributed random number.

Here is how the binary coding method works:

step 1 Determine the parameters to be optimized according to the specific problem.

step 2 Determine its range of variation for each parameter and use a binary number to represent it. For instance, if the variation range of parameter a is $[a_{min}, a_{max}]$, and we use m-bit binary number to represent a, then we have:

$$a = a_{min} + \frac{b}{2^m - 1}(a_{max} - a_{min}) \tag{2}$$

At this time, the parameter range should be determined to cover all the optimal space, and the length m should be taken as small as possible in order to minimize the complexity of SGA while meeting the accuracy requirement.

step 3 The binary number of all the parameters is concatenated into a long binary string, which is the operation object of the genetic algorithm.

2.2.2 Individual Fitness Evaluation

SGA does not use external information in evolutionary search and only search according to fitness function value of each individual in the population. Therefore, the selection of the fitness function is rather important. SGA usually directly uses the objective function as fitness function. However, because it should determine the probability that each individual in the population is inherited into the next generation with the probability which is proportional to the individual fitness, to compute the probability correctly, the fitness value of every individual should be non-negative. So we have to make some proper handling of objective function. In all the formulas below, $f(x)$ is the objective function and $Fit(x)$ is the fitness function.

To begin with, we talk about the most simple method. We directly transform the objective function into fitness function. If the objective function is a maximization problem, then

$$Fit(f(x)) = f(x) \tag{3}$$

If the objective function is a minimization problem

$$Fit(f(x)) = -f(x) \tag{4}$$

This kind of fitness function is really intuitive, however, it does not satisfy the requirement that the fitness value is always non-negative and some of the functions to be solved differ

greatly in the distribution of function values, which makes it hard for the resulting average fitness to reflect the average performance of the population. So we come up with another method. If the objective function is a minimization problem, then

$$Fit(f(x)) = \begin{cases} c_{max} - f(x), & f(x) < c_{max} \\ 0, & otherwise \end{cases} \tag{5}$$

and $c_{max}$is the maximum estimate of $f(x)$.

If the objective function is a maximization problem, then

$$Fit(f(x)) = \begin{cases} f(x) - c_{min}, & f(x) > c_{min} \\ 0, & otherwise \end{cases} \tag{6}$$

and $c_{min}$is the minimum estimate of $f(x)$. It is an improved version of the first method. But there are also some problems with this method, such as the toughness to accurately estimate the maximum and the minimum of the objective function. As for the third method, If the objective function is a minimization problem, then

$$Fit(f(x)) = \frac{1}{1 + C + f(x)}, \quad C \geq 0, C + f(x) \geq 0 \tag{7}$$

If the objective function is a maximization problem, then

$$Fit(f(x)) = \frac{1}{1 + C - f(x)}, \quad C \geq 0, C - f(x) \geq 0 \tag{8}$$

It is similar to the second method, and C is a conservative estimate of the bounds of objective function.

2.2.3 Genetic Operators

- Selection operator

  The selection operation is based on individual evaluation. It is more likely that the individual with higher fitness is inherited to the next generation and vise versa. So it can avoid the loss of the effective gene and make the better individuals survive with a higher probability. The computational efficiency is improved.

  The most popular selection operator is proportional model. It selects the corresponding individual with a probability that is proportional to the individual's fitness. Let f_i be the fitness of individual i, P_i be the probability that individual i is selected, Q_i be the cumulative probability of the individual i. They are calculated as below.

$$P_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{9}$$

$$Q_i = \sum_{j=1}^{i} P_j \tag{10}$$

  $i = 1, 2, \ldots, N$. And it is easy to see that $Q_N = 1$. Which individual is selected is determined by comparing these cumulative probabilities with a random number r uniformly distributed in the interval $[0, 1]$. If $r \in [0, Q_1]$, the individual 1 is selected, or $r \in (Q_{i-1}, Q_i]$, the individual i is selected.

- Crossover operator

  Crossover is a process of taking more than one parent solutions and producing a child solution from them. SGA uses the single point crossover. It generate the crossover pairs randomly in the mating pool according to a certain crossover probability, and then select a crossover point randomly, and then exchange the subsequent chromosome. Two new individuals are generated. When the length of the coding string of chromosome is m, we have $m - 1$ crossover points, so we can have $m - 1$ different results. To make

things clear, we have a picture below to illustrate the point.

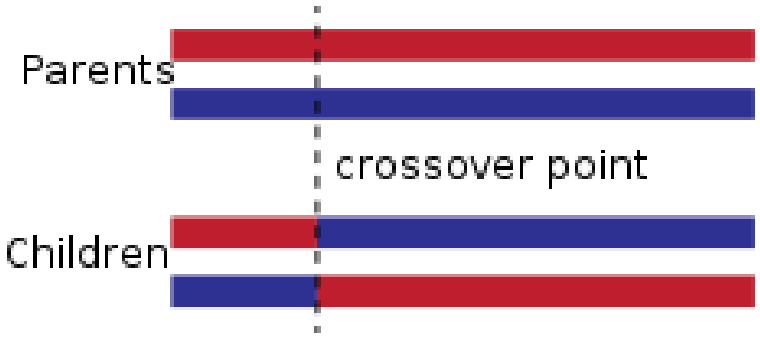

Figure 1: single point crossover
Source:from wikipedia

If we use binary coding string to represent the chromosome, we have an example as below:

parent A:1011|0101 → 1011|1010 child A'

parent B:1000|1010 → 1000|0101 child B'

- Mutation operator

  Mutation alters one or more gene values in a chromosome from its initial state. In mutation, the solution may change entirely from the previous solution. Hence GA can come to a better solution by using mutation. Mutation occurs during evolution according to a user-definable mutation probability. This probability should be set low. If it is set too high, the search will turn into a primitive random search. [1]SGA specifies several bits to mutate randomly according to the mutation probability and at the bits to mutate, the value of bits are modified. As for binary coding method, we change $1 \to 0$ and $0 \to 1$. For example, $1010101 \to 1000111$.

2.2.4 Operating Parameters

There are four basic operating parameters in SGAm which are P_c, N, P_m, T. They are related to the dimension of the variables and the geometry characteristic of the function. In general,

[1]https://en.wikipedia.org/wiki/Mutation_(genetic_algorithm)

the range of them are as below:

- N:population size. Because when we choose larger population size, we can deal with more solutions at the same time while the time each iteration costs will increase. So we always choose the range from 20 to 100.
- T:the number of generations when SGA terminates.It always ranges from 100 to 500.
- P_c:crossover probability. Larger the crossover probability, quicker the solution will converge to the domain of the optimal solution. However, too large crossover probability will make the solution converge too early. So we always choose the range from 0.4 to 0.99.
- P_m:mutation probability. Large the mutation probability, more diverse the samples are and less stable the solutions are. So we always choose the range from 0.0001 to 0.1.

2.3 The Algorithm

simple genetic algorithm

```
1: randomly initialize population(0)
2: determine fitness of population(0)
3: repeat
4:     select parents from population(t)
5:     perform crossover on parents creating population(t+1)
6:     perform mutation of population(t+1)
7:     determine fitness of population(t+1)
8: until best individual is good enough
```

3 Example: Maximizing a Function of One Variable

Consider the problem of maximizing the function:

$$f(x) = \frac{-x^2}{10} + 3x, \quad x \in [0, 31] \tag{11}$$

To solve this problem using GA, we need to encode the possible value of x as chromosomes. We choose to encode the x as a binary integer of length 5. Because it is clear to everyone that 5-bit binary integer range from 0(00000) to 31(11111).

To begin the algorithm, we select an initial population of 10 chromosomes at random. We can achieve this by tossing a fair coin 5 times for each chromosome, letting heads signify 1 and tails signify 0. Next we take the x-value that each chromosome represents and test its fitness with the fitness function. The result is shown in Table 1.

Table 1: Initial Population

Chromosome Number	Initial Population	x Value	Fitness Value $f(x)$	Selection Probability
1	0 1 0 1 1	11	20.9	0.1416
2	1 1 0 1 0	26	10.4	0.0705
3	0 0 0 1 0	2	5.6	0.0379
4	0 1 1 1 0	14	22.4	0.1518
5	0 1 1 0 0	12	21.6	0.1463
6	1 1 1 1 0	30	0	0
7	1 0 1 1 0	22	17.6	0.1192
8	0 1 0 0 1	9	18.9	0.1280
9	0 0 0 1 1	3	8.1	0.0549
10	1 0 0 0 1	17	22.1	0.1497
		Sum	147.6	
		Average	14.76	
		Max	22.4	

We select the chromosomes that will reproduce based on their fitness values, using the probability calculated by (9). Since our population has 10 chromosomes and each 'mating' produces 2 offspring, we need 5 matings to produce a new generation of 10 chromosomes. To create their offspring, a crossover point is chosen at random. Note that it is possible that crossover does not occur, in which case the offspring are exact copies of their parents. The result is shown in Table 2.

Table 2: Reproduction & Second Generation

Chromosome Number	Mating Pairs	New Population	x Value	Fitness Value $f(x)$
5	0 1|1 0 0	0 1 0 1 0	10	20
2	1 1|0 1 0	1 1 1 0 0	28	5.6
4	0 1 1 1|0	0 1 1 1 1	15	22.5
8	0 1 0 0|1	0 1 0 0 0	8	17.6
9	0 0 0 1|1	0 **1** 0 1 0	10	20
2	1 1 0 1|0	1 1 0 1 1	27	8.1
7	1 0 1 1 0	1 0 1 1 0	22	17.6
4	0 1 1 1 0	0 1 1 1 0	14	22.4
10	1 0 0|0 1	1 0 0 0 1	17	22.1
8	0 1 0|0 1	0 1 0 0 1	9	18.9
			Sum	174.8
			Average	17.48
			Max	22.5

Lastly, each bit of the new chromosomes mutates with a low probability. For this example, we let the probability of mutation be 0.001. With 50 total transferred bit positions, we expect $50 \cdot 0.001 = 0.05$ bits to mutate. Thus it is likely that no bits mutate in the second generation. For the sake of illustration, we have mutated one bit in the new population, which is shown in bold in Table 2. After selection, crossover, and mutation are complete, the new population is tested with the fitness function. The fitness values of this second generation are listed in Table 2. Comparing Table 2 to Table 1, we see that both the maximum fitness and average fitness of the population have increased after only one generation.[2]

4 The Future Direction of GA

GAs are promising methods for solving difficult technological problems, and for machine learning. More generally, GAs are part of a new movement in computer science that is exploring biologically inspired approaches to computation. Advocates of this movement believe

[2]Carr J. An introduction to genetic algorithms[J]. Senior Project, 2014: 1-40.

that in order to create the kinds of computing systems we need—systems that are adaptable, massively parallel, able to deal with complexity, able to learn, and even creative—we should copy natural systems with these qualities. Natural evolution is a particularly appealing source of inspiration.

Genetic algorithms are also promising approaches for modeling the natural systems that inspired their design. Most models using GAs are meant to be "gedanken experiments" or "idea models" (Roughgarden et al. 1996) rather than precise simulations attempting to match real−world data. The purposes of these idea models are to make ideas precise and to test their plausibility by implementing them as computer programs (e.g., Hinton and Nowlan's model of the Baldwin effect), to understand and predict general tendencies of natural systems (e.g., Echo), and to see how these tendencies are affected by changes in details of the model (e.g., Collins and Jefferson's variations on Kirkpatrick's sexual selection model). These models can allow scientists to perform experiments that would not be possible in the real world, and to simulate phenomena that are difficult or impossible to capture and analyze in a set of equations. These models also have a largely unexplored but potentially interesting side that has not so far been mentioned here: by explicitly modeling evolution as a computer program, we explicitly cast evolution as a computational process, and thus we can think about it in this new light. For example, we can attempt to measure the "information" contained in a population and attempt to understand exactly how evolution processes that information to create structures that lead to higher fitness. Such a computational view, made concrete by GA−type computer models will eventually be an essential part of understanding the relationships among evolution, information theory, and the creation and adaptation of organization in biological systems (e.g., see Weber, Depew, and Smith 1988).[3]

[3]Melanie M. An introduction to genetic algorithms[J]. Cambridge, Massachusetts London, England, Fifth printing, 1999, 3: 62-75.

5 Conclusion

Through all the discussion above, we can see how genetic algorithms work using the viewpoint of the biological evolution. In addition, what the field of genetic algorithms has achieved, and what are the most interesting and important directions for future research are also briefly summarized.

6 Reference

[1]https://www.doc.ic.ac.uk/ nd/surprise_96/journal/vol1/hmw/article1.html#overview

[2]https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)

[3]https://en.wikipedia.org/wiki/Mutation_(genetic_algorithm)

[4]Melanie M. An introduction to genetic algorithms[J]. Cambridge, Massachusetts London, England, Fifth printing, 1999, 3: 62-75.

[5]https://en.wikipedia.org/wiki/Genetic_algorithm

[6]Carr J. An introduction to genetic algorithms[J]. Senior Project, 2014: 1-40.

[7] 金芬. 遗传算法在函数优化中的应用研究 [D]. 苏州大学,2008.

[8] 李明. 遗传算法的改进及其在优化问题中的应用研究 [D]. 吉林大学,2004.