DATA STRUCTURE & ALGORITHM TERM PAPER

Genetic Algorithm Simulation

Zekun Zhang

April 7, 2019

Genetic Algorithm Simulation

Zekun Zhang (zz364)

ABSTRACT

Just as the most famous Charles Darwin said, "It is not the strongest of the species that survives, nor the most intelligent, but the one most responsive to change." So, in my term paper, I would like to show the process of the "evolution" with the famous algorithm which is called Genetic Algorithm (shorted as GA in the following part of this paper).

1 INTRODUCTION

My motivation to choose this topic is inspired by the biology reason why there's only one species that has such high intelligence and how the evaluation process is like. What's more, in the future, what the humans are going to be, smarter or stronger?

As we all know, there is an important term in biology that is called Natural Selection, which is the key idea of the whole Genetic Algorithm. So, what's that? You can just take a hypothetical situation that you are the commander of humans and we need to create more good people to make the world better. So, what are you going to do? You may take these following policies:

1. You select all the good people who are born by some genetic mutation and ask them to extend their generation by having their children. In the meantime, you can't let the normal ones having their children. **(this may cause some racial issues, but I must make a statement that it's just a science question and it's really a hypothesis and all these "policies" really can't represent my personal idea)**
2. Then repeat step 1 for a few generations.
3. You will notice that now you have an entire population of good people.

From Figure 1, we can easily know the basic idea of the GA. And we can understand it in this way: these policies can be considered as that we change the inputs(parents), then we would get better outputs(children). After repeating this step, then we would find that the bad values would be fewer.

And how genetic algorithm actually works? It basically tries to mimic human evolution to some extent. So, if we try to formalize a definition of GA, we can just say that it is an optimization algorithm, which tries to find out such values of inputs so that we get the better outputs values or results. The working flow of GA is also derived from biology, which is as shown in Figure 2.

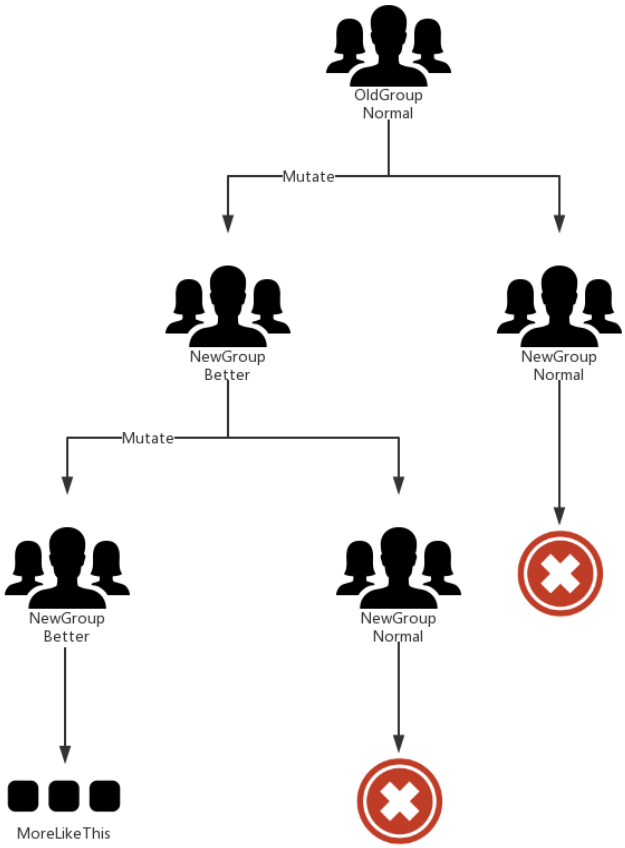


Figure 1

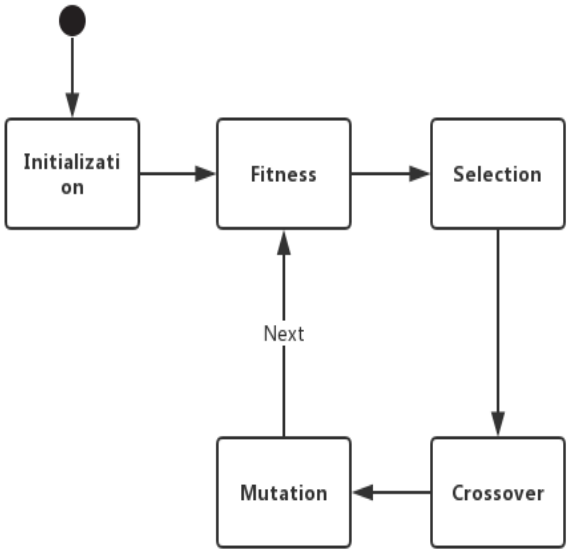


Figure 2

2 BACKGROUDS

2.1 Concept 1: Genes and chromosomes

In the genetic algorithm, we first need to map the problems to be solved into a mathematical problem, the so-called "mathematical modeling", then a feasible solution to this problem is called a "chromosome." As we all know that chromosome is a container for plenty of genes in biology. So, how can we make the computer know the sequences of genes? A feasible solution is generally composed of multiple elements (made up by 1's and 0's), and each of these elements is called a "gene" on the chromosome.

For example, we can see that from Figure 3 that by using a binary sequence, it becomes easier to input this binary sequence into our algorithm program and we can use this mathematical model to finish our simulation of GA.

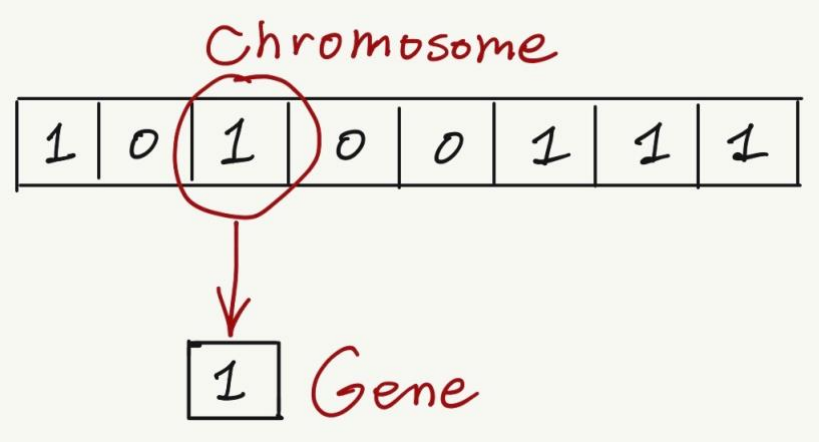


Figure 3

2.2 Concept 2: Natural Selection

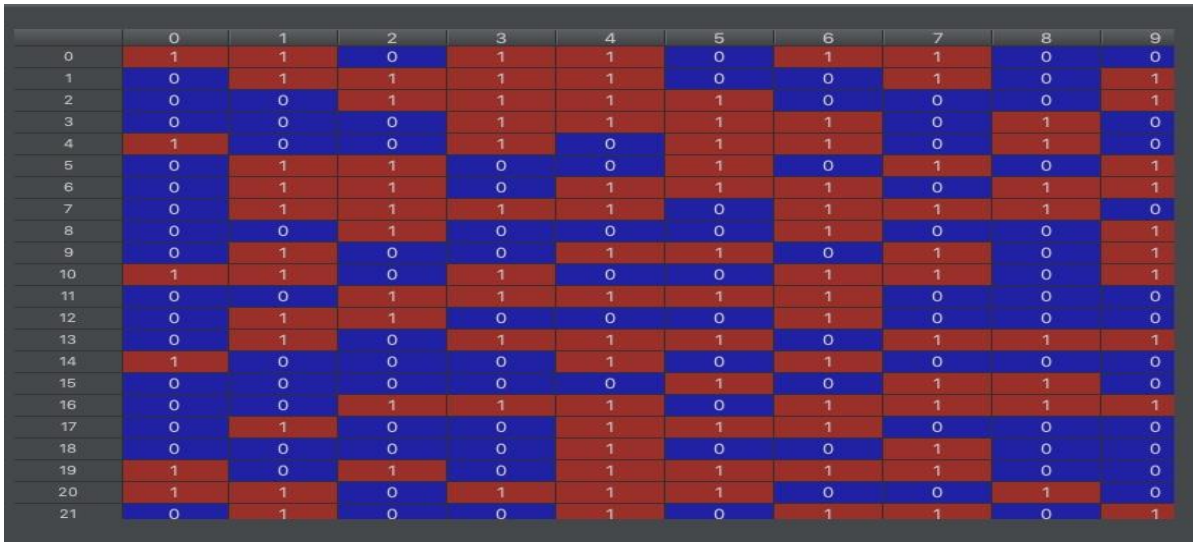
In the genetic algorithm, in order to simulate the whole process of the evaluation, we need to find a mathematical model for these steps. As we know that in the real world, natural selection must have two conditions: one is mutation and the other is competition. In nature, there seems to be a God who can choose better individuals in each generation and eliminate some individuals with poor environmental adaptability. This is done by the fitness function. The fitness function plays the role of "God" in the genetic algorithm. Since if the species doesn't fit the environment, it must have less chance to win the competition in the real world. "God" is relatively fair. In the meantime, if there's no mutation, a species would not make any difference in the competition. Then I use a random sequence generator to solve the mutation problem.

3 IMPLEMENTATIONS

3.1 Initialization

In the genetic algorithm, our first step would be defining our first population. So, our population will contain individuals, each having their own random set of chromosomes.
As I've mentioned I try to use binary code to encode the DNA sequence, after the random binary generator operation (the code snippet below), chromosomes become binary strings. Just as Figure 4, with this random function, we've already got 100 random individuals.

```
1.  pop = np.random.randint(2, size=(POP_SIZE, DNA_SIZE))  # initialize the pop DNA
```



	0	1	2	3	4	5	6	7	8	9
0	1	1	0	1	1	0	1	1	0	0
1	0	1	1	1	1	0	0	1	0	1
2	0	0	1	1	1	1	0	0	0	1
3	0	0	0	1	1	1	1	0	1	0
4	1	0	0	1	0	1	1	0	1	0
5	0	1	1	0	0	1	0	1	0	1
6	0	1	1	0	1	1	1	0	1	1
7	0	1	1	1	1	0	1	1	1	0
8	0	0	1	0	0	0	1	0	0	1
9	0	1	0	0	1	1	0	1	0	1
10	1	1	0	1	0	0	1	1	0	1
11	0	0	1	1	1	1	1	0	0	0
12	0	1	1	0	0	0	1	0	0	0
13	0	1	0	1	1	1	0	1	1	1
14	1	0	0	0	1	0	1	0	0	0
15	0	0	0	0	0	1	0	1	1	0
16	0	0	1	1	1	0	1	1	1	1
17	0	1	0	0	1	1	1	0	0	0
18	0	0	0	0	1	0	0	1	0	0
19	1	0	1	0	1	1	1	1	0	0
20	1	1	0	1	1	1	0	0	1	0
21	0	1	0	0	1	0	1	1	0	1

Figure 4

3.2 Fitness Function

The genetic algorithm performs N iterations during the run, and each iteration generates several chromosomes. The fitness function will score all the chromosomes generated in this iteration to judge the fitness of these chromosomes, then eliminate the chromosomes with lower fitness, and only retain the chromosomes with higher fitness, and then iterate through several iterations. The quality of the post chromosome will be getting better and better. The score after the fitness function is determined by the values of the individuals in the prediction plot. (It would be shown in the following part of this paper)

```
1.  # find non-zero fitness for selection
2.  def get_fitness(pred): return pred + 1e-3 - np.min(pred)
```

3.3 Selection

Now, we can select chromosomes with high fitness scores from our population which can mate and create their children. This step is the key to the genetic algorithm because better parents would have more chance to generate better off-springs. The general thought is that we should select better parents and allow them to produce off-springs. Therefore, I try to use Roulette Wheel Selection method, which can be easily explained like this: After each evolution, the fitness of each chromosome is calculated, and then the fitness probability of each chromosome is calculated using the following formula.

$$\text{Probability to be chosen} = \text{Fitness score} / \text{All the Fitness of others}$$

Then, in the crossover part, you need to choose the parent chromosome based on this probability. The higher the probability that a more adaptive chromosome is selected. This is why genetic algorithms can preserve good genes.

```
def select(pop, fitness):  # nature selection wrt pop's fitness
    idx = np.random.choice(np.arange(POP_SIZE), size=POP_SIZE, replace=True,
                           p=fitness / fitness.sum())
    return pop[idx]
```

3.4 Crossover

In the genetic algorithm, after the selection operation, each iteration for the already selected individuals is called a crossover. The process of intersection needs to find two chromosomes from the chromosomes of the previous generation, one is the father and the other is the mother. Then one of the two chromosomes would be cut and spliced together to create a new chromosome. This new chromosome contains a certain number of father's genes, and also contains a certain number of mother's genes. And we can simplify this process into the problem trying to find out the cross point with a random number generator. Also, in Figure 5, we can see the result of the crossover process.

```
def crossover(parent, pop):  # mating process (genes crossover)
    if np.random.rand() < CROSS_RATE:
        i_ = np.random.randint(0, POP_SIZE, size=1)  # select another individual from p
op
        cross_points = np.random.randint(0, 2, size=DNA_SIZE).astype(np.bool)  # choose
 crossover points
        parent[cross_points] = pop[i_, cross_points]  # mating and produce one child
    return parent
```

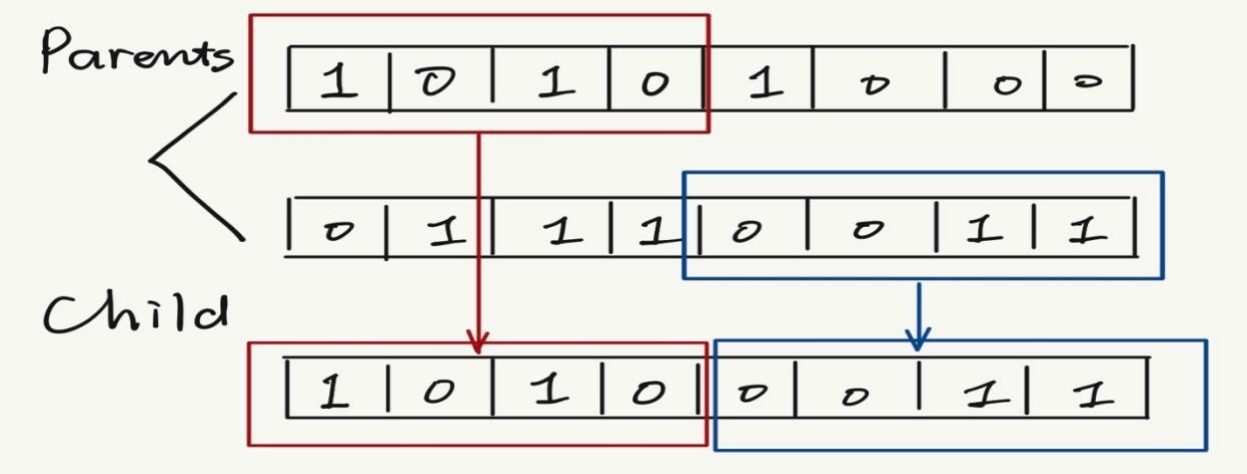


Figure 5

3.5 Mutation

Crossover only guarantees that every evolution leaves a good gene, but it's only selected from the original result set, and the genes snippets are still the same, only change the order of composition. This can only guarantee that after N evolutions, the calculation results are closer to the local optimal solution, and there is no way to reach the global optimal solution. In order to solve this problem, we need to introduce the mutation.

The variation is easy to understand. When we generate a new chromosome by crossover, we need to randomly select several genes on the new chromosome, and then randomly modify the value of the gene, thus introducing a new gene to the existing gene snippets, breaking through the limitations of the current search, and It is beneficial for the algorithm to find the global optimal solution, which means we can produce better off-springs.

And from the code snippet below, I just randomly change some of the 1's to 0's and 0's to 1's.

```
1.  def mutate(child):
2.      for point in range(DNA_SIZE):
3.          if np.random.rand() < MUTATION_RATE:
4.              child[point] = 1 if child[point] == 0 else 0
5.      return child
```

4 EVALUATION & ANALYSIS

In the genetic algorithm, the first thing we should consider is how we will get to know that we have reached our best possible solution? Or in other word, when to finish the process of evaluations, because in the GA, there's the mechanism of mutation.

So, basically there are different termination conditions, which are listed below:

1. There is no improvement in the population over x iterations.
2. We have already predefined an absolute number of generations for our algorithm.
3. When our fitness function has reached a predefined value.

According to these three conditions, I've predefined a 200-generations limitation for my code. Then I've got the following results: (Figure 6 is initial state, Figure 7 is after 80 gens, Figure 8 is finished 200 gens)

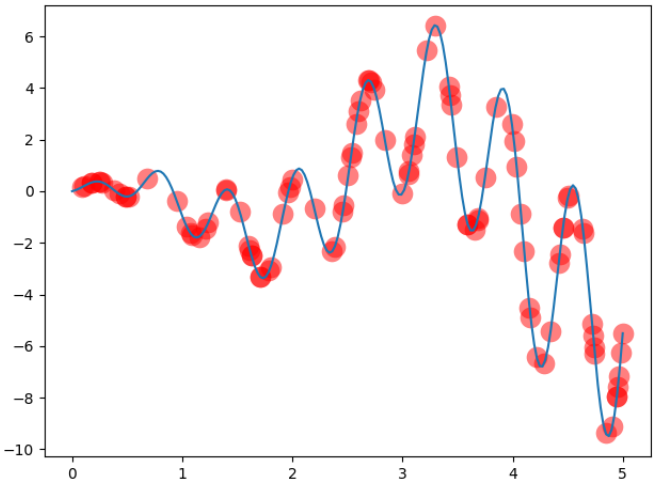


Figure 6

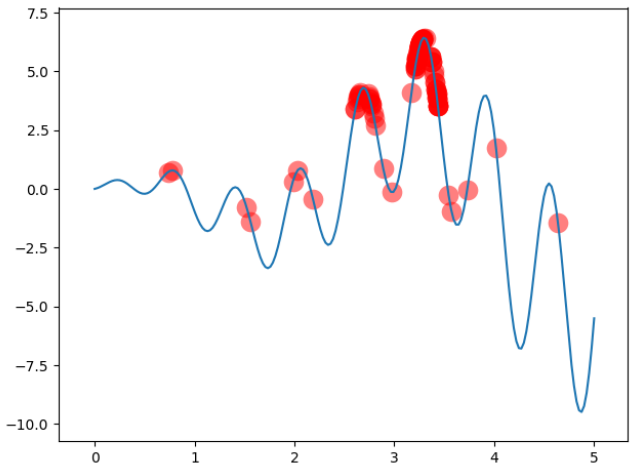


Figure 7

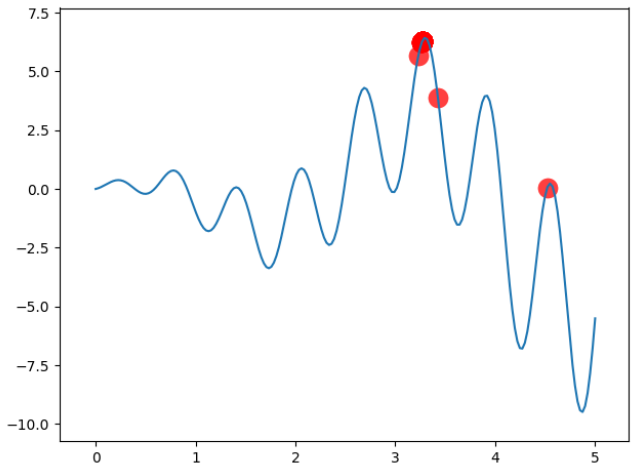


Figure 8

5 CONCLUSIONS

In this simulation project, I've already implemented this famous genetic algorithm and from the final results, it really works well. After around 80 generations, the output from the plot we can see that is near the best result. However, I must acknowledge that the performance of this code is not really good. It really takes time to finish the simulation work, and there's a risk of memory overflow, because in the process, there are lots of figures needed to be plotted. In future work, I may try to change the number of generation limits or the mutation rate to optimize this algorithm and the plotting intervals to solve some of the problem listed.

6 RELETED WORK

6.1 Sentences Match

We can use GA to do automatic sentences generator and sentences match work. We can consider "Hello World!" as our final goal, and firstly we just randomly generate some characters, then after a number of generations, we can get the final result — "Hello World!".

6.2 Traffic and Shipment Routing (Travelling Salesman Problem)

This is a famous problem and has been efficiently adopted by many sales-based companies as it is time-saving and economical. This is also achieved using the genetic algorithm.

7 REFERENCE

1. Mitchell, Melanie. *An introduction to genetic algorithms*. MIT press, 1998.

2. https://en.wikipedia.org/wiki/Genetic_algorithm

3. https://www.analyticsvidhya.com/blog/2017/07/introduction-to-genetic-algorithm/

4. https://morvanzhou.github.io/tutorials/machine-learning/evolutionary-algorithm/2-01-genetic-algorithm/

5. https://zhuanlan.zhihu.com/p/33042667