Using Genetic Algorithm to Solve Function

Qijian Ma*
qma4@student.cccs.edu

Akacia Schauermann
@student.cccs.edu

Corbin Miller
@student.cccs.edu

Abstract

In this paper, our team will present a fully detailed breakdown of a revolutionary genetic algorithm model that has been designed to solve calculus questions through the process of evolution, rather than the traditional formulaic approach. By utilizing various genetic operations such as mutation, crossover, and selection, this model is able to generate and optimize solutions, resulting in highly efficient answers that are tailor-made for each unique problem. This paper aims to provide a comprehensive understanding of the intricate workings of this innovative approach, highlighting its potential to revolutionize the field of calculus and beyond.

1 Introduction

The **genetic algorithm(GA)** is an optimization technique that utilizes a population-based approach to find the best solutions for intricate problems. It draws inspiration from natural selection and evolution, where the strongest individuals are more likely to survive and pass on their genes to the next generation. The GA process involves selecting, crossing over, and mutating a population of potential solutions, which generates new and superior solutions progressively. By repeatedly refining these potential solutions, the GA algorithm can efficiently search and converge towards the optimal solution.

To assess the effectiveness of the genetic algorithm, the team selected two functions with multiple variables and varying complexity levels to evaluate how well the model could be trained and to measure the accuracy of the results. By selecting diverse functions, the team was able to assess the model's ability to handle complex problems with multiple variables and to determine how effective the algorithm was in optimizing the objective functions.

2 Model Architecture

The model build base on two part, **the generation loop** and **the descendant loop** what is inside the generation loop. Since the purpose of this model is to find the unknown variables in the function, therefore, the genetic algorithm model is **unsupervised**, which the first generation of the model is two arrays generated by random floats and integer.

2.1 Generation Loop

The generation loop is a key part of the genetic algorithm model and is responsible for generating a diverse population of candidate solutions to a problem, which are then evaluated, selected, recombined, and mutated to generate new candidate solutions for the next generation.

2.2 Descendant Loop

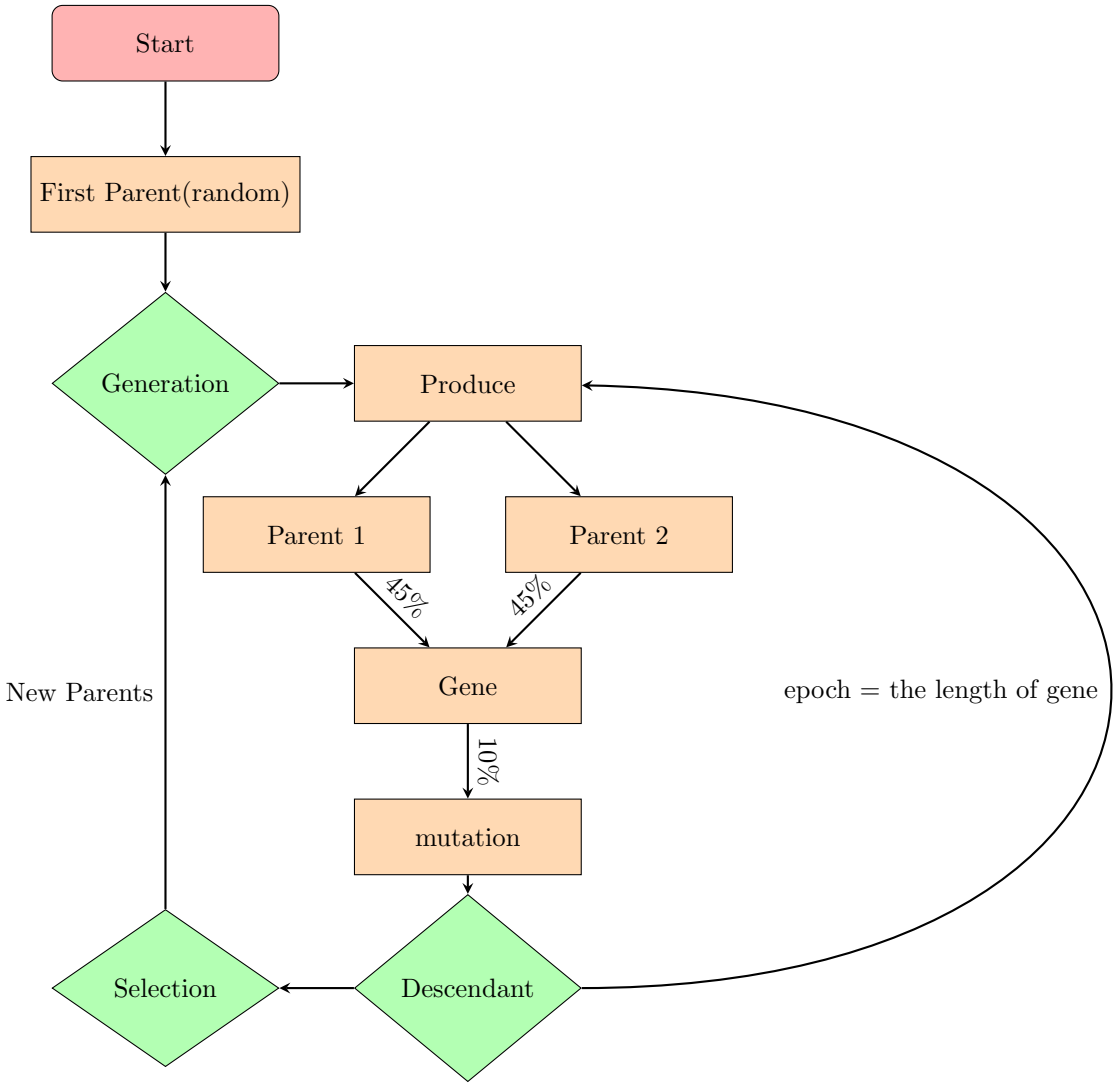
The descendant loop, on the other hand, involves applying genetic operators, such as crossover,mutation and selection, to the current population of candidate solutions to generate a new population of solutions. The descendant loop is critical in driving the evolution of the population towards better solutions, and it is repeated until a satisfactory solution is obtained or a termination criterion is met.

2.2.1 Crossover

During crossover, the genetic algorithm selects pairs of candidate solutions from the two of the best in previous generation and exchanges genetic information between them.

2.2.2 mutation

In mutation, the genetic algorithm selects candidate solutions from the population and randomly changes their genetic material. This random change can occur at the individual gene level, where a gene value is randomly replaced by random number.



3 Test Model 1

Here is the formula for the first model:

$$e^x = 12.5 \tag{1}$$

3.1 Steps

1. Define the problem: In this case, we want to find the value of x that satisfies the equation $e^x = 12.5$.

2. Define the fitness function: The fitness function evaluates how well a candidate solution (in this case, a possible value of x) satisfies the equation $e^x = 12.5$. One possible fitness function could be the absolute difference between the left-hand side (e^x) and the right-hand side (12.5) of the equation. The closer this difference is to 0, the better the candidate solution.

3. Generate an initial population: The initial population is a set of candidate solutions, each represented as a chromosome in the genetic algorithm. In this case, the chromosomes could be a set of random values of x within a certain range.

4. Evaluate the fitness of the initial population

3.2 Training Process

The process involves several steps, beginning with the random generation of two arrays, each containing a single number through the AnE function. After generating the arrays, the genetic algorithm is used to set up the number of generations and the number of descendants in each generation. For this particular formula, a total of **200 generations** was chosen, with each generation containing **500 descendants** and **50% mutation rate** on every gene. The next step in the process is to use a fitness function to evaluate the quality of each candidate solution. The fitness function should take as input a candidate solution (i.e., a value of x) and return a numerical value that represents how well that solution satisfies the equation $e^x = 12.5$. The fitness function used in this case could be the absolute difference between the left-hand side (e^x) and the right-hand side (12.5) of the equation. The closer this difference is to 0, the better the candidate solution, which will be use for the next generation.

3.3 Result

The training procedure for the model took an impressively short amount of time, clocking in at approximately 1 second to complete. Upon completion, the model generated a final result of **2.52572862**. In order to derive the answer from this result, it was necessary to apply it to a specific formula.

Once the result was applied to the formula, the calculated answer came out to be **12.49999969**. It is essential to compare this experimental output with the actual, known result to gauge the accuracy and efficiency of the model. To do this, the difference between the experimental result and the actual result was calculated.

Interestingly, the difference between the two results was found to be a mere **3.264774566247297e-07**. This extremely small number signifies that the model's performance was highly accurate, as the discrepancy between its output and the actual result was nearly negligible.

In conclusion, the rapid 1-second training process and the minimal difference between the experimental and actual results demonstrate that the model is not only efficient but also highly accurate in its calculations. This should instill confidence in the model's ability to produce reliable outcomes in similar tasks and applications.

4 Test Model 2

Here is the formula for the second model:

$$(5e^a + 8b + 7\sin(c))\sin(5d) = 10 \tag{2}$$

4.1 Model Structure

Initially, the model, which is identical to Model 1, generated two separate arrays, with each array containing four random integers or floating-point numbers, utilizing the AnE function. This model is comprised of a total of **5000 generations**, and within each generation, there are **10000 descendants**. Additionally, the model has a mutation rate of **20%**, which plays a significant role in its overall functioning and development.

4.2 Result

The training process for the model was completed in a duration of **6 minutes and 52 seconds**. Upon completion, the resulting values for **parent 1** were[[**0.11851513, 61., 35.98283619, 72.23023833**]].

$$10 - (5e^{0.11851513} + 8(61) + 7\sin(35.98283619°)) * \sin(5 * (72.23023833°)) = 6.126006368845083e^{-8} \tag{3}$$

After incorporating these values into the relevant formula, the experimental result obtained was approximately [**9.9999991**], which is very close to the actual result. When comparing this experimental result to the actual one, the margin of error was found to be a minuscule [**6.126006368845083e**-8]. This indicates that the result is relatively accurate and closely aligned with the true outcome. As the model continues to undergo further training and improvements, it is highly possible that the genetic algorithm will eventually achieve complete accuracy, reaching a perfect **100%** match with the actual result.

5 Conclusion

In summary, the main goal of this paper was to offer a break down of the genetic algorithm, specifically focusing on its capacity to tackle calculus functions. This highlights a prime example of the effective application of the ever-evolving field of machine learning to intricate mathematical problems, such as those encountered in calculus. Throughout the course of this project, our team has gleaned invaluable knowledge and understanding of the principles, methods, and approaches involved in both machine learning and calculus.

As we advanced through the various phases of the project, it became increasingly clear that there exists tremendous potential for future progress and enhancements in the domain of genetic algorithms and their application to calculus functions. As a result, our team is highly driven and committed to sustaining the existing model while simultaneously pursuing the development of an even more advanced version. Our overarching objective is to produce a highly sophisticated and effective model that can consistently yield precise results and contribute to a deeper comprehension of the relationship between machine learning and calculus.

For more information and the access of the code