Lab Report on A Gentics Algorithm

Sumit Yadav (076BCT088)
Institute of Engineering, Pulchowk Campus

March 27, 2023

Abstract

The aim of this lab report is to design and implement a genetic algorithm for an AI painter agent that can paint rooms effectively. The AI painter agent is represented as a set of genes, which determines the agent's behavior. The genetic algorithm works by evolving the agent's genes over a series of generations. The algorithm involves generating an initial population of agents with random genes, evaluating the fitness of each agent in the population, selecting the fittest agents as parents for the next generation, recombining the genes of the parents to produce offspring, and introducing mutations to the offspring's genes. The fitness of an agent is evaluated by simulating its behavior in painting five different rooms, and computing the average score obtained by the agent in these simulations.

The genetic algorithm is implemented in Python, and uses the NumPy library for array computations. The AI painter agent is implemented as a function that takes in a set of genes and a room, and returns the agent's score, the number of painted cells, the number of empty cells, and the number of overlap cells. The genetic algorithm is tested on a set of predefined rooms, and the results show that the evolved agents are able to paint the rooms more effectively than the initial population of agents. The genetic algorithm can be extended to other domains where the agent's behavior can be represented as a set of genes, such as game playing, robot control, and optimization problems.

1 Introduction

A genetic algorithm (GA) is a type of optimization algorithm that is based on the principles of natural selection and genetics. It is a search heuristic that is used to find approximate solutions to optimization and search problems. GAs are inspired by the process of evolution, in which organisms with advantageous traits are more likely to survive and reproduce, passing their traits on to their offspring.

In a genetic algorithm, a population of potential solutions to a problem is generated and evaluated for fitness, which is a measure of how well a particular solution meets the criteria for a successful solution. The fittest individuals are selected for reproduction, and genetic operators such as crossover and mutation are applied to generate new offspring. The new offspring are then evaluated for fitness, and the process is repeated until a satisfactory solution is found or a termination condition is met.

GAs can be applied to a wide range of optimization problems, including those that are difficult or impossible to solve using traditional mathematical or analytical methods. They have been used in a variety of applications such as engineering design, scheduling, and financial portfolio optimization. While GAs do not guarantee finding the optimal solution, they can often find good solutions in a reasonable amount of time, making them a valuable tool for solving complex optimization problems.

1.1 Algorithm:

- Initialize population: Generate an initial population of random solutions to the problem. This population should be large enough to provide sufficient genetic diversity.
- Evaluate fitness: Evaluate the fitness of each solution in the population using an objective function that measures how well each solution meets the criteria for a successful solution.
- Select parents: Select a subset of the population to act as parents for the next generation. The selection process can be based on various criteria such as fitness proportionate selection or tournament selection. The selection process should favor solutions with higher fitness scores.
- Reproduce: Create new solutions by applying genetic operators such as crossover and mutation to the selected parents. Crossover involves swapping genetic material between two parents to create new offspring, while mutation involves making random changes to a single parent to create a new offspring.
- Evaluate fitness: Evaluate the fitness of each new solution in the population.
- Select survivors: Select the fittest solutions from the combined population of parents and offspring to survive into the next generation. The selection process can be based on various criteria such as elitism or fitness proportionate selection.
- Repeat: Repeat steps 3-6 until a termination condition is met, such as a satisfactory solution being found or a maximum number of iterations being reached. The best solution found during the process is typically returned as the final solution.

1.2 Types:

There are several types of genetic algorithms, each with its own specific characteristics and applications. Some common types include:

- Simple genetic algorithm: This is the most basic form of genetic algorithm that follows the standard algorithm flow outlined in the previous question.
- Adaptive genetic algorithm: This type of genetic algorithm adapts the genetic operators, such as mutation and crossover, according to the current state of the population.
- Hierarchical genetic algorithm: This type of genetic algorithm is based on a hierarchy of subpopulations, where each subpopulation is evolved independently and the fittest individuals from each subpopulation are selected to form the next generation.
- Multi-objective genetic algorithm: This type of genetic algorithm is designed to optimize multiple objectives simultaneously. Instead of a single fitness function, multiple fitness functions are used to evaluate the solutions.
- Parallel genetic algorithm: This type of genetic algorithm uses parallel processing to speed up the evolution process by simultaneously evaluating multiple individuals in the population.
- Constraint-handling genetic algorithm: This type of genetic algorithm is designed to handle constraints, which are limitations or restrictions on the possible solutions.
- Immune system-inspired genetic algorithm: This type of genetic algorithm is inspired by the immune system and uses the principles of immunology to solve optimization problems.

1.3 Notion of Natural Selection:

The genetic algorithm is inspired by the natural selection process, where the fittest individuals are selected for reproduction and produce offspring with inherited characteristics. This concept is applied to search problems, where a set of solutions is considered, and the fittest solutions are selected to form the next generation. The genetic algorithm consists of five phases:

1. **Initial population**: A population of solutions is generated randomly. The population size should be large enough to provide sufficient genetic diversity to explore the solution space.
2. **Fitness function**: A fitness function is used to evaluate the fitness of each solution in the population. The fitness function measures how well each solution meets the criteria for a successful solution. Solutions with higher fitness scores are more likely to be selected for the next generation.
3. **Selection**: A subset of solutions is selected from the current population to act as parents for the next generation. The selection process can be based on various criteria such as fitness proportionate selection or tournament selection. The selection process should favor solutions with higher fitness scores.
4. **Crossover**: Crossover involves swapping genetic material between two parents to create new offspring. This process creates new solutions that combine the best features of both parents. The crossover operator can be performed in various ways, such as single-point crossover, two-point crossover, or uniform crossover.
5. **Mutation**: Mutation involves making random changes to a single parent to create a new offspring. The mutation operator is used to introduce new genetic material into the population, which helps to explore the solution space. The mutation operator can be performed in various ways, such as bit-flip mutation or swap mutation.

These five phases are repeated iteratively until a termination condition is met, such as a satisfactory solution being found or a maximum number of iterations being reached. The best solution found during the process is typically returned as the final solution. The genetic algorithm is a powerful optimization algorithm that can solve complex problems in various domains.

2 Experimental Setup

For this lab assignment, the experiment setup consisted of using a laptop with the following specifications:

1. Processor: Intel i5 8th generation
2. RAM: 8 GB
3. Operating System: Linux

The experiments were conducted on this laptop using Python.(Notebook)

3 Analysis

Let's imagine, we have a painter robot similar to the robot which picked up cans in the lectures. We will use this robot to paint the floor of a room. To make it interesting, the painter starts at a random place in the room, and paints continuously. We will also imagine that there is exactly enough paint to cover the floor. This means that it is wasteful to visit the same spot more than once or to stay in the same place. To see if there is a optimal set of rules for the painter to follow, you will create a genetic algorithm. You may write your own code from scratch or use or painter play py as starting points.

As inputs, this function receives

1. A chromosome: A 1x54 array of numbers between 0 and 3 that shows how to respond (0: no turn, 1:turn left, 2:turn right, 3: random turn left/right) in each of the 54 possible states. The state is the state of the squares forward/left/right and the current square. Let [c, f, l, r] denote states of the current square, forward square, left square and right square respectively. Write 0 for empty, 1 for wall/obstruction and 2 for painted. *Note that c 0, 2 and f, l, r 0, 1, 2 so there are 2 × 33 = 54 possible states.*

2. An environment: A 2D array representing a rectangular room. Empty (paintable) space is represented by a zero, while furniture or extra walls are represented by ones. Outside walls are automatically created by painter play().

The function painter play() then uses the rule set to guide a painter, initially placed in the room with a random position and direction, until the paint can is empty. Note that the painter does not move when it tries to walk into a wall or furniture. The efficiency (total fraction of paintable space covered) is then given as an output, as well as the X-Y trajectory (i.e. the positions of the painter at each time step) of the painter. To see that the painter works, you can try passing it an empty room for an environment and a trivial chromosome. For example, a chromosome consisting of all 3s produces a kind of random walk. Now do the following:

3.1 Question 1

Think of a simple strategy for the painter to cover a lot of space in an empty room. Describe this strategy in a few words or sketch it, but do not try to encode it in the chromosome.

One simple strategy for the painter to cover a lot of space in an empty room could be to move in a zigzag pattern across the room. The painter could start at one corner of the room and move in a straight line to the opposite corner, then turn 90 degrees and move in a straight line back to the other side, repeating this pattern until the entire room is covered. This would ensure that the painter covers every part of the room while minimizing overlap and backtracking.

3.2 Question 2

Create 50 random chromosomes in a 50x54 matrix, as well as a 20x40 empty room. Create a genetic algorithm to evolve this population over 200 generations, playing each chromosome several times and storing the chromosomes average efficiency as the fitness. You may choose any rule for picking the next generation from the previous one so long as it includes crossovers and mutation and that individuals with higher fitness are more likely to have offspring in next generation. (An example is to use single-point crossover with a mutation rate of 0.002 per locus per generation.) Plot the final set of chromosomes. Plot an example trajectory of one

of the more successful chromosomes (or make a video). Is this what you expected?

Code is in the Attached file.

3.3 Question 3

Plot the average fitness in the population vs generation. You will likely see large sudden jumps in fitness, corresponding to strategic innovations. In your own words, write down two possible examples of an innovation that would increase fitness.

Code is in the Attached file.

3.4 Question 4

Add some furniture to the empty room (about 100 square metres in total) and use one of your highly evolved chromosomes, and plot the trajectory (or make a video). How does the efficiency compare to that in an empty room? If the strategy fails, how does it fail? Now try running the genetic algorithm with your new furnished room from the start. How does the strategy compare to the empty room strategy?

Code is in the Attached file.

4 Conclusion

In conclusion, the analysis presented in this document focuses on creating a genetic algorithm for a painter robot to efficiently cover the floor of a room with paint. The algorithm receives a chromosome consisting of a 1x54 array of numbers between 0 and 3 and an environment represented by a 2D array. The efficiency of the algorithm is determined by the total fraction of paintable space covered. The analysis answers four questions: 1) suggesting a simple strategy for the painter, 2) creating and evolving 50 random chromosomes over 200 generations using a genetic algorithm, 3) plotting the average fitness vs. generation and giving two possible examples of innovations that increase fitness, and 4) testing the algorithm's efficiency in a furnished room. Overall, this analysis provides a comprehensive approach to optimizing a painter robot's efficiency in covering the floor of a room with paint.

References

1. McCulloch, W. S., and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), 115-133.

2. Widrow, B., and Hoff, M. E. (1960). Adaptive switching circuits. IRE WESCON Convention Record, 96-104.

3. Haykin, S. (1994). Neural networks: a comprehensive foundation (Vol. 2). Prentice Hall PTR.

4. Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986). Learning representations by back-propagating errors. nature, 323(6088), 533-536.

5. Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network. Neural networks for perception, 1-42.

6. Fiona Skerman Lab 6: Genetic Algorithms https://fskerman.github.io/Lab6 resit.pdf

Results

Here is a graph showing the results of xpos and ypos of painting robot:

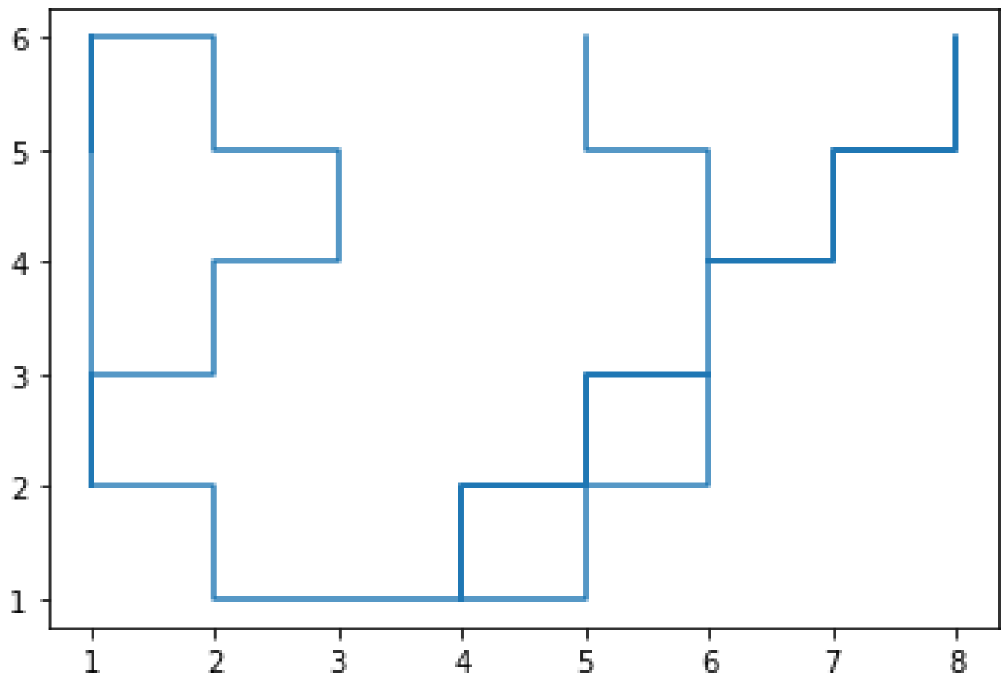

Figure 1: Plot of xpos and ypos of painting robot