IMPLEMENTATION AND ANALYSIS OF GENETIC ALGORITHMS

A
MAJOR PROJECT REPORT
Submitted by

ISHA NEGI	**MAYUR GARG**
(03714802715)	(05214802715)

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

Under the Guidance
of
Ms. Deepti Gupta
(Assistant Professor, CSE)

Department of Computer Science and Engineering
Maharaja Agrasen Institute of Technology, PSP area, Sector – 22, Rohini, New Delhi – 110085
(Affiliated to Guru Gobind Singh Indraprastha, New Delhi)
(MAY/JUNE 2019)

MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering

CERTIFICATE

This is to be Certified that this MAJOR project report “IMPLEMENTATION AND ANALYSIS OF GENETIC ALGORITHMS” is submitted by “ISHA NEGI (03714802715) and MAYUR GARG (05214802715)” who carried out the project work under my supervision.

I approve this MAJOR project for submission.

Dr. Namita Gupta	Ms. Deepti Gupta
(HoD, CSE)	(Assistant Professor)
	(Project Guide)

ABSTRACT

Revolutionary enhancements have been made in the field of Artificial Intelligence and Machine Learning. One of the key aspects of the latter is to solve problems of which little is known in advance. Such problems encompasses a very broad search space and it is, more often than not, tedious to design a situation specific algorithm. In such cases, adaptive methods such as Evolutionary algorithms are preferred. One of the most widespread used evolutionary algorithms is Genetic Algorithms. Genetic algorithms mimic the processes involved in natural selection and evolution to repeatedly improve on its solution and arrive at an optimal or near optimal result. Genetic algorithms, or evolutionary algorithms in general, are extensively used to solve many search and optimisation problems specially NP-hard problems where it is computationally infeasible to arrive at an optimal solution but quite often a near optimal solution is sufficient. Other problems which can be solved by genetic algorithms include problems where it is unclear as to how an algorithm must be designed to solve it. Such problems such as the designing character movement in games can be tackled by Genetic algorithms as long as we can define a fitness function to evaluate the effectiveness of our result. In the suggested system, we have aimed to implement Genetic Algorithms to tackle the Infinite Monkey Problem, Travelling Salesman Problem and train and simulate an Autonomous Self Driving Car. We have also aimed to understand the basic functioning of the simple Genetic Algorithm used and have expected to find a correlation of various parameters in the Genetic Algorithm used such as the Length of the input string, Size of the population, Mutation Rate, Elitism and Scoring pattern with the associated efficiency of the algorithm used for the Infinite Monkey Problem in arriving at a solution by varying these parameters in a suitable range.

ACKNOWLEDGEMENT

It gives me immense pleasure to express my deepest sense of gratitude and sincere thanks to my highly respected and esteemed guide Ms. Deepti Gupta, Assistant Professor, CSE , MAIT Delhi, for their valuable guidance, encouragement and help for completing this work. Their useful suggestions for this whole work and co-operative behaviour are sincerely acknowledged.

I am also grateful to all my department teachers for their constant support and guidance.

I also wish to express my indebtedness to my parents as well as my family member whose blessings and support always helped me to face the challenges ahead.

Place: Delhi	Isha Negi(0374802715)
Date:	Mayur Garg(05214802715)

APPENDIX I

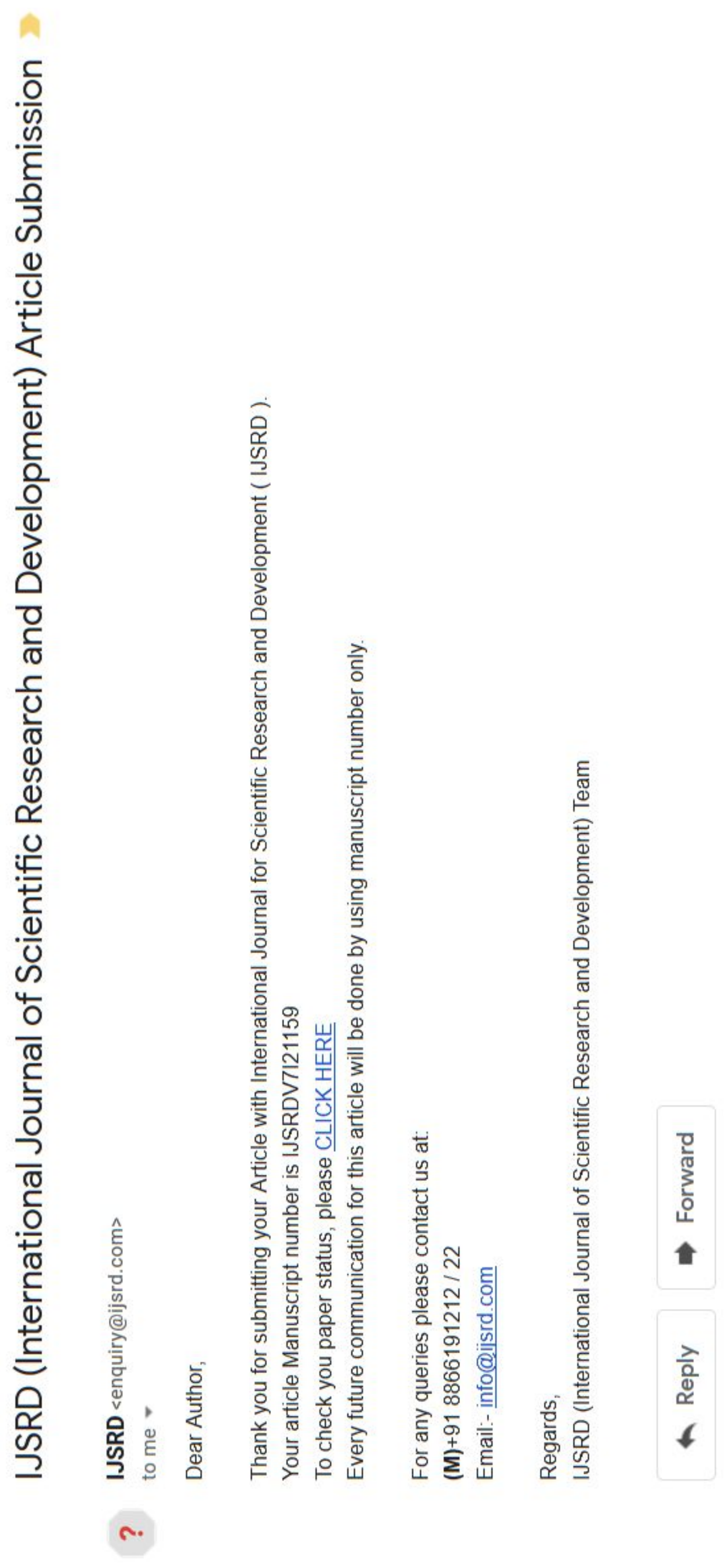
TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS

- GPS - Global Positioning System
- AI - Artificial Intelligence
- GA - Genetic Algorithm(s)
- BBH - Building Block Hypothesis
- ESA-LOGA - Enhanced Selection and Log Scaled Mutation GA
- GTSP - Generalized Travelling Salesman Problem
- AVG - Automatic Vehicle Guidance
- API - Application Programming Interface
- IDE - Integrated Development Environment
- IDLE - Integrated Development Learning Environment

PROOF OF RESEARCH PAPER PUBLICATION


IJSRD (International Journal of Scientific Research and Development) Article Submission

IJSRD <enquiry@ijsrd.com>
to me

Dear Author,

Thank you for submitting your Article with International Journal for Scientific Research and Development (IJSRD).
Your article Manuscript number is IJSRDV7I21159
To check you paper status, please CLICK HERE
Every future communication for this article will be done by using manuscript number only.

For any queries please contact us at:
(M)+91 8866191212 / 22
Email:- info@ijsrd.com

Regards,
IJSRD (International Journal of Scientific Research and Development) Team

CHAPTER 1
INTRODUCTION

1.1 Overview

The genetic algorithm is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution. The genetic algorithm repeatedly modifies a population of individual solutions. At each step, the genetic algorithm selects individuals at random from the current population to be parents and uses them to produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution. Genetic algorithms can find fit solutions in a very less time which are considerably good as per the need. The random mutation guarantees to some extent that we see a wide range of solutions.

1.1.1 Infinite Monkey Problem

The infinite monkey theorem states that a monkey hitting keys at random on a typewriter keyboard for an infinite amount of time will almost surely type any given text, such as the complete works of William Shakespeare. However, the probability that a monkey would type a complete work such as of Shakespeare is so tiny that the chance of it occurring during a period of time hundreds of thousands of orders of magnitude longer than the age of the universe is extremely low (but technically not zero).

However, this time can be greatly reduced by allowing the initial random results to learn over time. This can be achieved by using genetic algorithms wherein random text can be led to generate the desired one by simulating a process of natural selection.

1.1.2 Travelling Salesman Problem

This problem consists of finding a closed route which is of minimum length that traverses through a given set of points. Such a similar problem is often encountered by people in delivery business where it is optimal for them to traverse through various destinations and

return back to the starting point using the shortest net route.
It is an NP-hard problem in combinatorial optimization. Using methods such as brute force to calculate this minimal route can take an impractical amount of time if the size of the set of the points is large enough. However, using genetic algorithms a considerably good (although it may not be the optimal one) can be found in realistic time.

1.1.3 Autonomous Car Simulation

An autonomous car, also known as a self driving car car, or a driverless car, is a vehicle that is capable of sensing its environment and moving with little or no human input. Autonomous cars combine a variety of sensors to perceive their surroundings, such as radar, computer vision, Lidar, sonar, GPS, odometry and inertial measurement units. Advanced control systems interpret sensory information to identify appropriate navigation paths, as well as obstacles and relevant signage.
One of the control systems that can interpret information from the surroundings to help make decisions to drive an autonomous car is genetic algorithms. Such an algorithm (alongside other AI based techniques) can be deployed to train an autonomous car system to identify obstacles on the path and make intelligent decisions to avoid them.

1.1.4 Genetic Algorithms

A genetic algorithm solves optimization problems by creating a population or group of possible solutions to the problem. The individuals in this population will carry chromosomes that are the values of variables of the problem.
The genetic algorithm solves the problem by allowing the less fit individuals in the population to die and selectively breeding the most fit individuals (the ones that solve the problem best). This process is called selection, as in selection of the fittest. The genetic algorithm will take two fit individuals and mate them (a process called crossover). The offspring of the mated pair will receive some of the characteristics of the mother, and some of the father.

In nature, offspring often have some slight abnormalities, called mutations. Usually these mutations are disabling and inhibit the ability of the children to survive, but once in a while they improve the fitness of the individual. The genetic algorithm similarly occasionally causes mutations in its populations by randomly changing the value of a variable.

After the genetic algorithm mates fit individuals and mutates some, the population undergoes a generation change. The population will then consist of offspring plus a few of the older individuals, which the genetic algorithm allows to survive to the next generation because they are the most fit in the population, and we will want to keep them breeding. These most fit individuals are called elite individuals.

After dozens or even hundreds of "generations", a population eventually emerges wherein the individuals will solve the problem very well. In fact, the most fit (elite) individual will be an optimum or close to optimum solution.

The processes of selection, crossover, and mutation are called genetic operators.

1.2 Need of Study

There are many opportunities for optimization in the business world today. Some of the best known are optimizing schedules and work flow to minimize cost and time, while maximizing output.

Some optimizers use simple "exhaustive search", meaning that every possible combination is tried to see what is the best one. This is an accurate approach, since we are bound to find the best combination of variables - eventually. However, it is a very inefficient approach, because whenever there are more than a few combinations, it takes too long to try them all. That is why users of exhaustive search optimizers tend to limit the number of variables they use, or tend to limit the number of values these variables can take.

The genetic algorithm, by contrast, does not try every possible combination. It attempts instead to intelligently get closer and closer to the best solution. Therefore, far more variables can be utilized, and you can allow all values of a variable. Optimization can still take a good deal of time if we give a GA a fair number of variables, but it will be doing much more work in that amount of time.

Genetic algorithms are searching dozens or hundreds of parts of the search space simultaneously. This means they are less likely to become stuck in "local minima" as the others quite often do. (Local minima are decent solutions that the optimizer can never get out of in order to find better solutions.)
We can apply the genetic algorithm to solve a variety of optimization problems that are not well suited for standard optimization algorithms, including problems in which the objective function is discontinuous, nondifferentiable, stochastic, or highly nonlinear.

1.3 Objectives

The main objectives of this project are as following:

1. Using a genetic algorithm to generate a string which is the exact match to the user input string starting from a set of randomly generated strings.
2. To deploy genetic algorithm to tackle route optimisation problem and find a close to minimal route connecting points on a 2D plane.
3. To train and simulate a car model to prevent obstacles by learning over various generations and cover its path.
4. To analyse the fitness of successive generations and accuracy and speed of the genetic algorithm to solve the above mentioned problems by varying various parameters such as size of each generation, elitism, mutation level, fitness function etc.

CHAPTER 2
LITERATURE REVIEW

Genetic algorithms (GAs) are adaptive methods which may be used to solve search and optimisation problems.[13] However, finding the optimal solution to complex high dimensional, multimodal problems often requires very expensive fitness function evaluations.[2]

But GAs are very common and efficient, many researchers have given an overview regarding their implementation of the GAs:

L. Haldurai et al.[1] presented an overview of the structure and functioning of the GAs and genetic operators which are Selection operations (Roulette wheel selection, Rank selection, Elitism selection), Crossover operations (Single point crossover, Two point crossover, Uniform crossover) and Mutation operations. It also proposed FGKA and IGKA algorithms which are modified versions of genetic algorithms to solve clustering problems.
Pushpendra Kumar Yadav et al.[2] emphasised the Building Block Hypothesis (BBH) which consists of a description of a heuristic that performs adaptation by identifying and recombining "building blocks" and a hypothesis that a genetic algorithm performs adaptation by implicitly and efficiently implementing this heuristic.
Kenneth De Jong[4] visualised the method of using genetic algorithms for adaptive search and proposes the creation of performance oriented learning system using GAs. It also proposed the method to use GAs to change parameters in performance systems by highlighting a key set of parameters that control the system's behaviour. GAs can also be used to modify data structures or executable code. However, for GAs, a large amount of samples may be taken from the search space before high quality solutions are found which is not possible for all domains.

David Beasley et al.[13] provided a statistical analysis of a typical GA run and its comparison with other methods such as gradient methods (which fail for multimodal functions), iterated search (where no overall picture of the shape of the domain is observed) and simulated annealing (which does not build an overall picture of the search space).

2.1 Techniques in Genetic Algorithms

David E. Goldberg et al.[8] tested a population-sizing equation to permit accurate statistical decision making among competing building blocks in population oriented search schemes such as genetic algorithms and also examined the number of function evaluation required to solve problems accurately.

Neeraj Gupta et al.[3] proposed ESA-LOGA (Enhanced Selection and Log Scaled Mutation GA) technique. In this implementation, after crossover, best genomes were selected from a pool of parents and children. Mutation rate was also varied logarithmically by mapping maximum and minimum values of the fitness function to the minimum and maximum value of permissible mutation rates respectively and then calculates mutation rate for each individual using a defined equation.

2.2 Travelling Salesman Problem Techniques

Jia Xu et al.[12] proposed a dynamic mutation technique on this problem of combinatorial optimisation wherein mutation operation is produced dynamically as its size varies with the change of population stability. Combining this with random cross mapping method, RCDM-GA approach was generated which provided significantly better results than T-GA (Traditional Genetic Algorithm).

Pooja Vaishnav et al.[11] surveyed various modifications of GAs to tackle GTSP (Generalized Travelling Salesman Problem) and concluded that performance of GAs can be improved by using hybridization techniques.

2.3 Development of Autonomous Cars

Semi-autonomous features such as lane keeping and automatic braking gave rise to vision guided autonomous driving.[9] Keshav Bimbraw[9] also gave an account of historical antecedents, contemporary progress and prospective predictions in the world of autonomous vehicles.

Yago Saez et al.[10] proposed using GAs for AVG (Automatic Vehicle Guidance). The GAs were tested on 3 different tracks and compared with other bots. However, one of the problems with this approach is that the trajectory obtained in the evolving process depends highly on the initial state of the car.

CHAPTER 3
METHODOLOGIES USED

3.1 Proposed Algorithm

Genetic algorithms in general use 3 major operators that define the structure and functioning of the algorithm. These are - Selection operator, Crossover operator and Mutation operator.

The selection operator is based on the idea to give preference to the individuals with good fitness scores and allow them to pass their genes to the successive generations. It includes definition of a fitness function which evaluates how well a particular individual performs in the given problem.

The crossover operator represents mating between individuals. Two individuals are selected using selection operator and crossover sites are chosen randomly. Then the genes at these crossover sites are exchanged thus creating a completely new individual (offspring). Crossover allows for the possibility of combination of good traits from each of the parents and hence an improved solution.

The key idea of the mutation operator is to insert random genes in offspring to maintain the diversity in population to avoid the premature convergence. Mutation allows the algorithm to search for the solution in a wider search space and prevents it from being stuck on a local maxima.

For each of the problems (Infinite Monkey Problem, Travelling Salesman Problem and Autonomous Car Simulation), unique functionalities of each of the operators were defined.

3.1.1 Flowchart

For the creation of genetic algorithms, the following flowchart was used. It was modified to suit the needs of each of the problems tackled in this project.

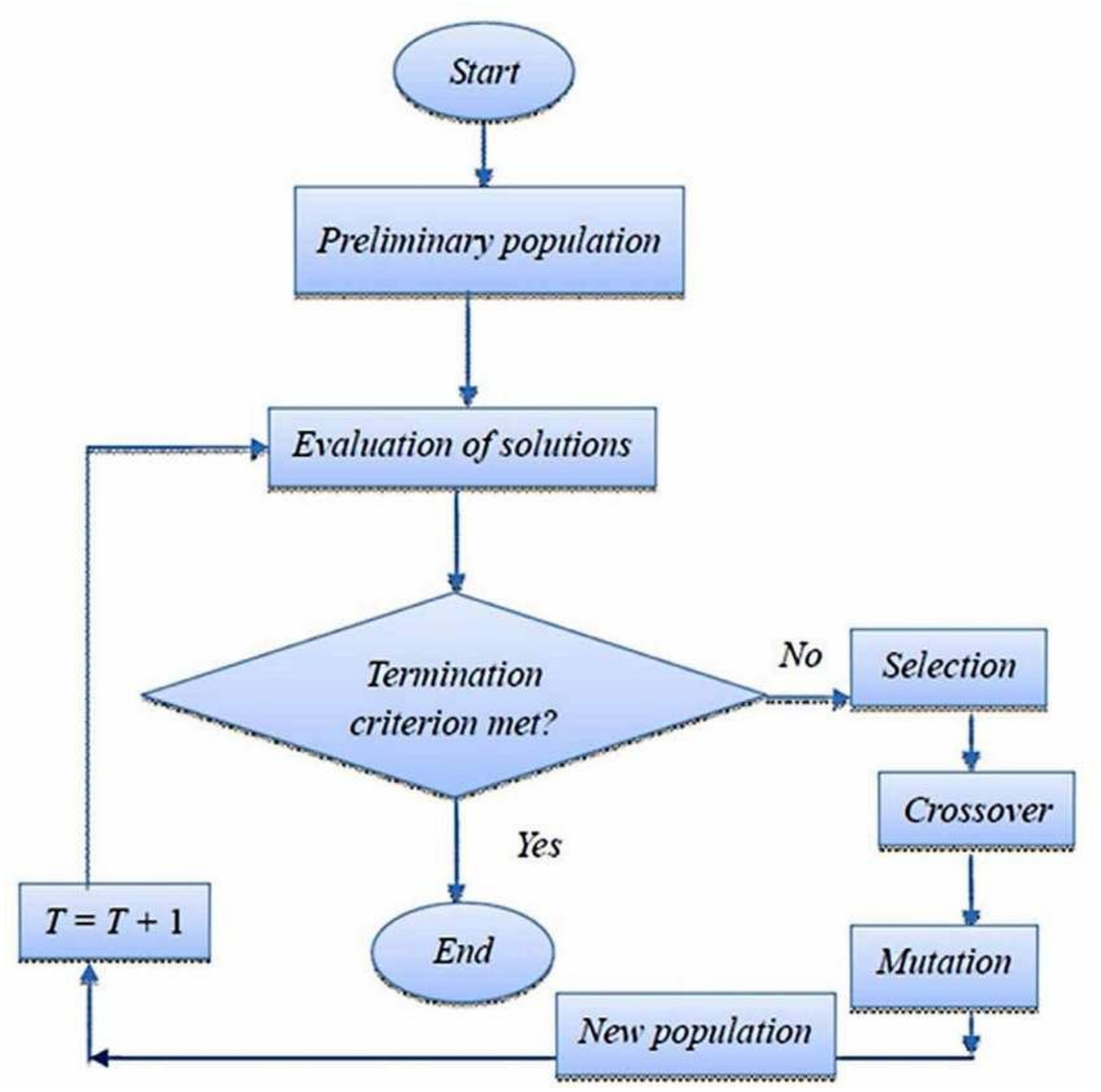


Fig 1. Flowchart of a Genetic Algorithm

3.1.2 Pseudocode

For the creation of genetic algorithms, following algorithm was used.
It was modified to fit each of the problems.

```
1. Begin.
2. Define DNA sequence characteristics for the given problem.
3. Generate initial population randomly of given size 'N' and set generation count =
   1.
4. Calculate fitness for each genome of the initial population.
5. If termination criteria is met, goto step 12 else goto step 6.
6. Sort the population by the value of the fitness of its genomes.
7. Create an empty set for new population.
8. From the old population, select few elite members (genomes with highest value of
   the fitness function) and add them to the new population.
9. From the remaining members of the old generation, fill the new generation upto
   size 'N' by crossing over selected genomes from the old population.
10. To these new crossed over genomes, apply mutation function based on the
    mutation rate.
11. Increment generation count and goto step 4.
12. End.
```

3.2 Technologies Used

3.2.1 Unity 3D 2018

Unity is a cross-platform game engine developed by Unity Technologies and was used to build this application interface. As of 2018, the engine has been extended to support upto 27 platforms. The engine can be used to create both 3D and 2D games as well as simulations and applications for desktops and laptops, home consoles, smart TVs, and mobile devices. Unity gives users the ability to create games in both 2D and 3D, and the engine offers a primary scripting API in C#, for both the Unity editor in the form of plugins, and games themselves, as well as drag and drop functionality.

Unity can be also be used to create simple applications for Desktop, Android and iOS as it comes with a powerful Physics engine and GUI design tools and projects can be easily exported as a Unity package, Android apk, Windows exe, WebGL file, etc from within the editor.

3.2.2 Microsoft Visual Studio 2017

Microsoft Visual Studio 2017 was used as the primary code editor environment while building this project. Coupled with Unity API, Visual Studio serves as a medium of seamless integration of code and its effect in the game and comes with powerful code editing, debugging and refactoring tools, specially for scripting for Unity game engine. Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps. It includes a code editor supporting IntelliSense as well as code refactoring. Visual Studio supports 36 different programming languages and allows the editor and debugger to support nearly any programming language. Built-in languages include C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML, and CSS.

3.2.3 C#

The primary programming language used for scripting in this project was C#. C# is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed around 2000 by Microsoft within its .NET initiative. C# is a modern, general-purpose, object-oriented programming language.

3.2.4 Python 3.6

Python is an interpreted high-level programming language for general-purpose programming. Python has a design philosophy that emphasizes code readability by using significant whitespace.

It is a multi-paradigm programming language wherein object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming.
Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution, which binds method and variable names during program execution.
Python was used in this project to compile and analyse the date generated and evaluate the basic insights generated from it.

3.2.5 Python IDLE

Python IDLE (Integrated Development and Learning Environment) is an integrated development environment for Python, which comes bundled with the default implementation of the language Python. IDLE is intended to be a simple IDE and suitable for beginners, especially in an educational environment. It was utilised to create and edit Python scripts for data analysis.

3.2.6 Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+.
Matplotlib library was used in Python scripts to generate line and bar graph from the raw data to gain insights.

CHAPTER 4
IMPLEMENTATION

4.1 Infinite Monkey Problem

4.1.1 Overview

In this project, we have used a genetic algorithms to solve the Infinite Monkey Problem such that instead of always randomly typing to generate the required string of characters, the system can learn from its previous attempts and get closer and closer to the required text. Therefore, this genetic algorithms slowly moves the system away from its purely random nature and helps it learn so that it can steer itself in the direction of the correct solution.

To set up the algorithm, following input parameters need to be provided :

- Target String
- Population Size
- Elitism
- Mutation Rate
- Scoring Pattern or Fitness Function

These parameters can be provided via a Android application user interface that has been created for this project.

4.1.2 Working

The algorithm functions in the following way :

1. Generation count is set to 1 and a random population of the given size is generated wherein each string is of the same length as that of the target string and is generated by taking characters randomly from the set of valid characters.
2. Fitness value is calculated for each string based on the scoring pattern defined and the population is sorted based on the fitness value of each string.

3. If fitness value of any string is 1, it denotes that the target string has been reached as all characters of that string matches to the corresponding characters of the target string. The algorithm stops and displays the current count of the generation.
4. Otherwise, an empty set of new population is initialised and the fittest members equal in count to the elitism value of the current generation is copied to the new population.
5. The rest of the set of the new population is populated by taking strings from the current population based on the weighted fitness sum method and applying Crossover operation on them to yield a new child string.
6. On such newly generated strings, Mutation operator is also applied.
7. Generation count is incremented and the system goes back to step 2.

4.1.3 Parameters Used

Size of Target String

Target string is that piece of text that is required to be typed in the Infinite Monkey Problem. In the original version of the problem, the target string equals the entire works of William Shakespeare. However in this genetic algorithm, one can provide a custom string that will be predicted over various generations by the system. Because of computational limitation and efficiency, we have limited the size of the target string to be in the range 5 to 30.

Since each character has an equal chance of being chosen i.e. the algorithm has no preference for any character position in the string and also no preference for any specific character from the valid characters set, the actual contents of the string has no effect on the effectiveness of the algorithm. This means that from the target string defined, only length of that string dictates how efficiently the algorithm can learn to predict it. This is because the size of the DNA defined for the genetic algorithm will be equal to the size of the target string and hence all strings predicted by the algorithm will have the same size to that of the target string.

The average no. of generations required by the genetic algorithm to predict the target strings of various sizes is displayed below for each of the scoring patterns -

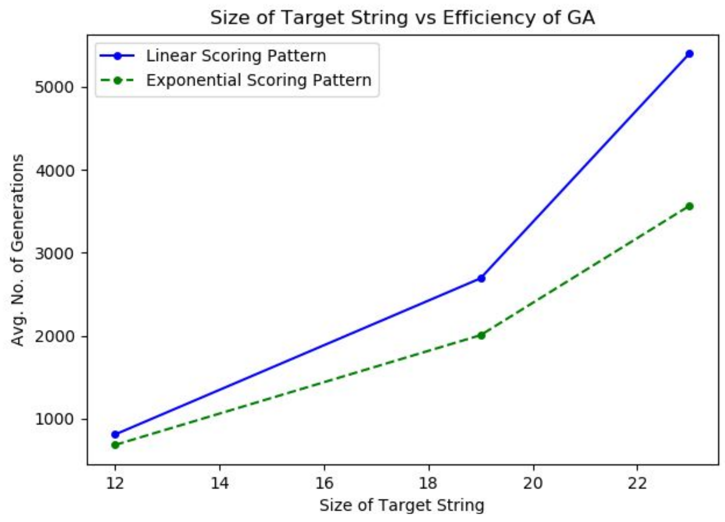


Fig 2. Effect of Size of Target String on Efficiency of GA

Population Size

Population size in a genetic algorithm detects the no. of child genes which are generated every generation. A higher population size gives the algorithm a more variety of genes to choose from for crossover but increase computation in the process per generation and hence slows down the rate of creation of new generations. For the analysis of this genetic algorithm, we have used 3 different population sizes - 15, 20 and 25.

The average no. of generations required by the genetic algorithm for each population size is displayed below for each of the scoring patterns -

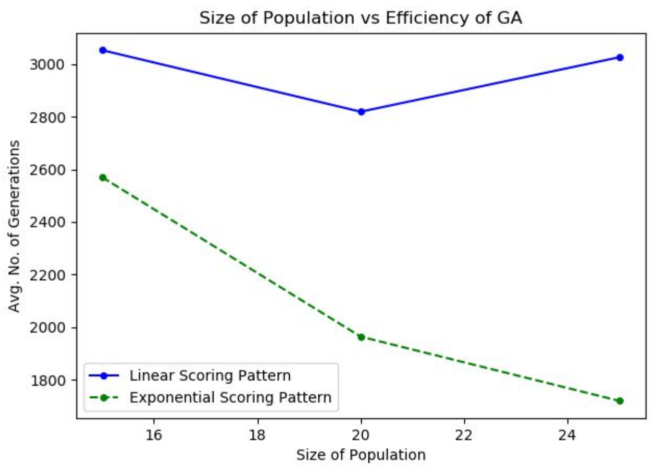


Fig 3. Effect of Population Size on Efficiency of GA

Elitism

Elitism is the number that denotes how many of the most fittest individuals would survive till the next generation without any crossover and mutation. For instance, an elitism value of 2 denotes that top 2 most fittest genes of the current generation would be copied directly to the next generation without the application of Crossover and Mutation operator on them.

Elitism is used to make sure the algorithm doesn't divert away from the target string. By making sure that fittest individuals survive, it is assured that continuous application of the Mutation operator doesn't steer the solution away from the global maxima. When elitism is set to 0, the average fitness of each next generation seems to fluctuate very rapidly instead of almost always rising upward. This greatly hinders the ability of the system to learn from its previous outputs.

For the analysis of this genetic algorithm, we have used 5 different elitism values - 1, 2, 3, 4 and 5.

The average no. of generations required by the genetic algorithm for each elitism value is displayed below for each of the population sizes -

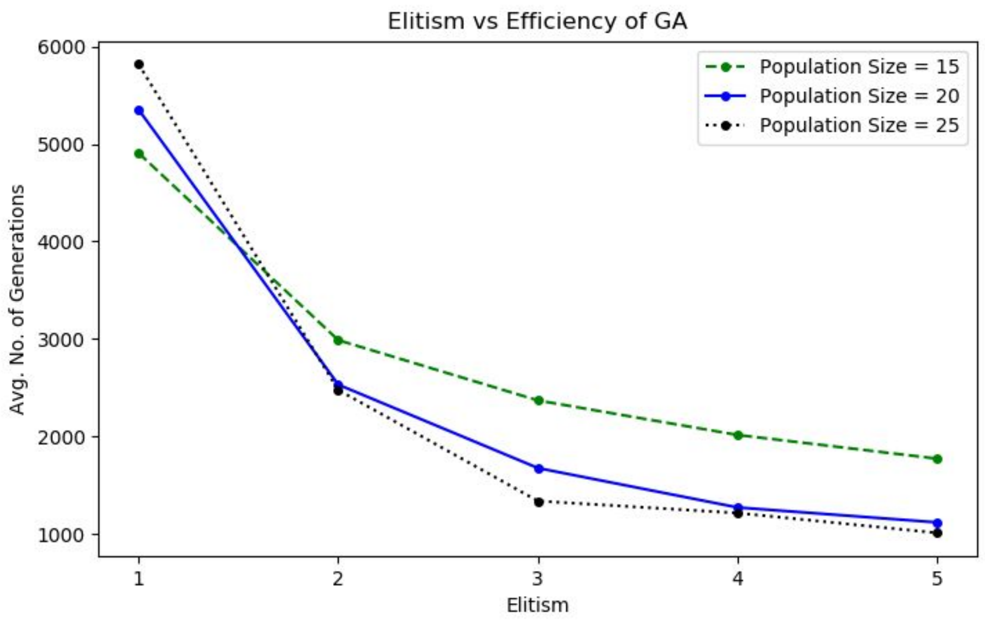


Fig 4. Effect of Elitism on Efficiency of GA

Mutation Rate

Mutation rate defines the no. of characters which are randomised in each child genome. Higher mutation rate allows the algorithm to search for the solution in a wider search space but also hinders the ability of the algorithm to steer towards the global maxima. Lower mutation rate on the other hand doesn't lead to major fluctuations in the average fitness of the generation but also doesn't allows us to look for solutions in the wider space which is very desirable in many cases.

When mutation is set to 0, it is quite possible that the algorithm never reaches the solution because all the child genes now being generated in the next generation are identical or similar to the genes from the previous generation and hence there is no new information for the algorithm to learn from and improve on its solution.

For the analysis of this genetic algorithm, we have used 4 different mutation rates - 0.01, 0.04, 0.07 and 0.1.

The average no. of generations required by the genetic algorithm for each mutation rate is displayed for each of the population sizes -

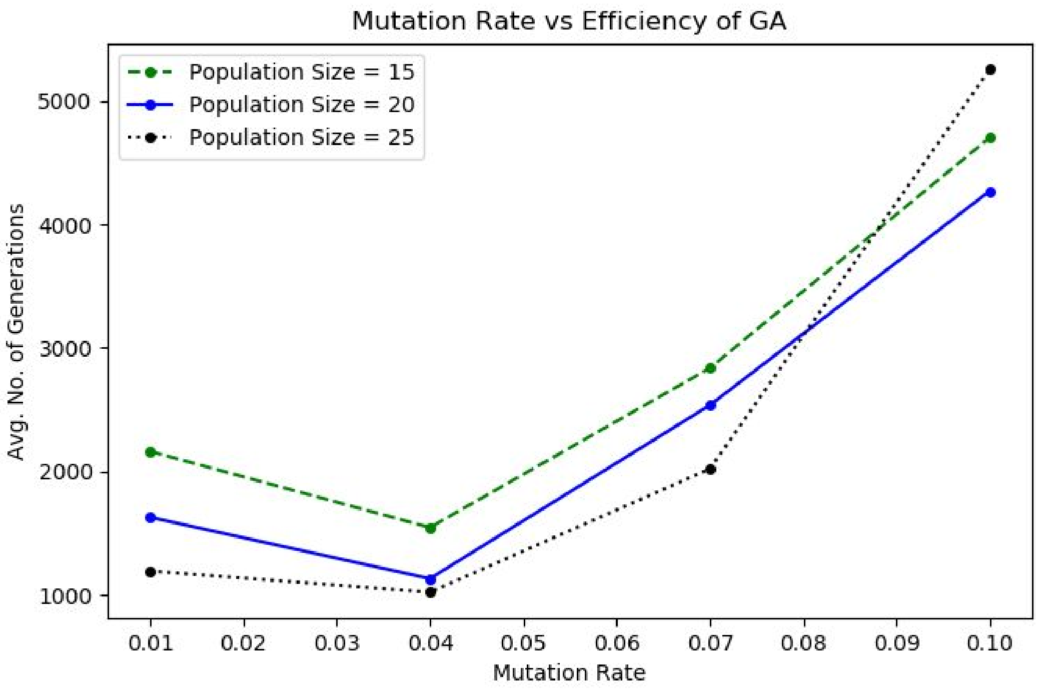


Fig 5. Effect of Mutation Rate on Efficiency of GA

Scoring Pattern

Scoring pattern dictates how the fitness value is calculated based on the no. of matching characters out of the total number of characters. For this algorithm, two different scoring patterns have been defined - Linear and Exponential.

Linear Scoring Pattern

No. of matching characters/Total number of characters.

Exponential Scoring Pattern

(5^(No. of matching characters/Total number of characters) - 1)/(5-1)

The average no. of generations required by the genetic algorithm for each of the scoring pattern is displayed below for each of the length of the strings -

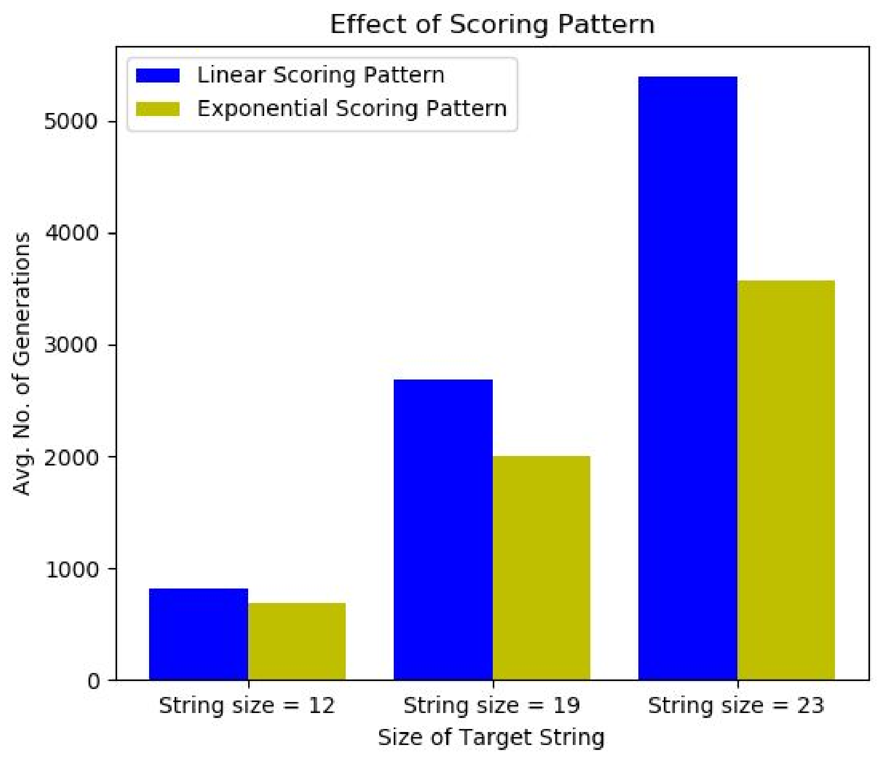


Fig 6. Effect of Scoring Patterns on Efficiency of GA

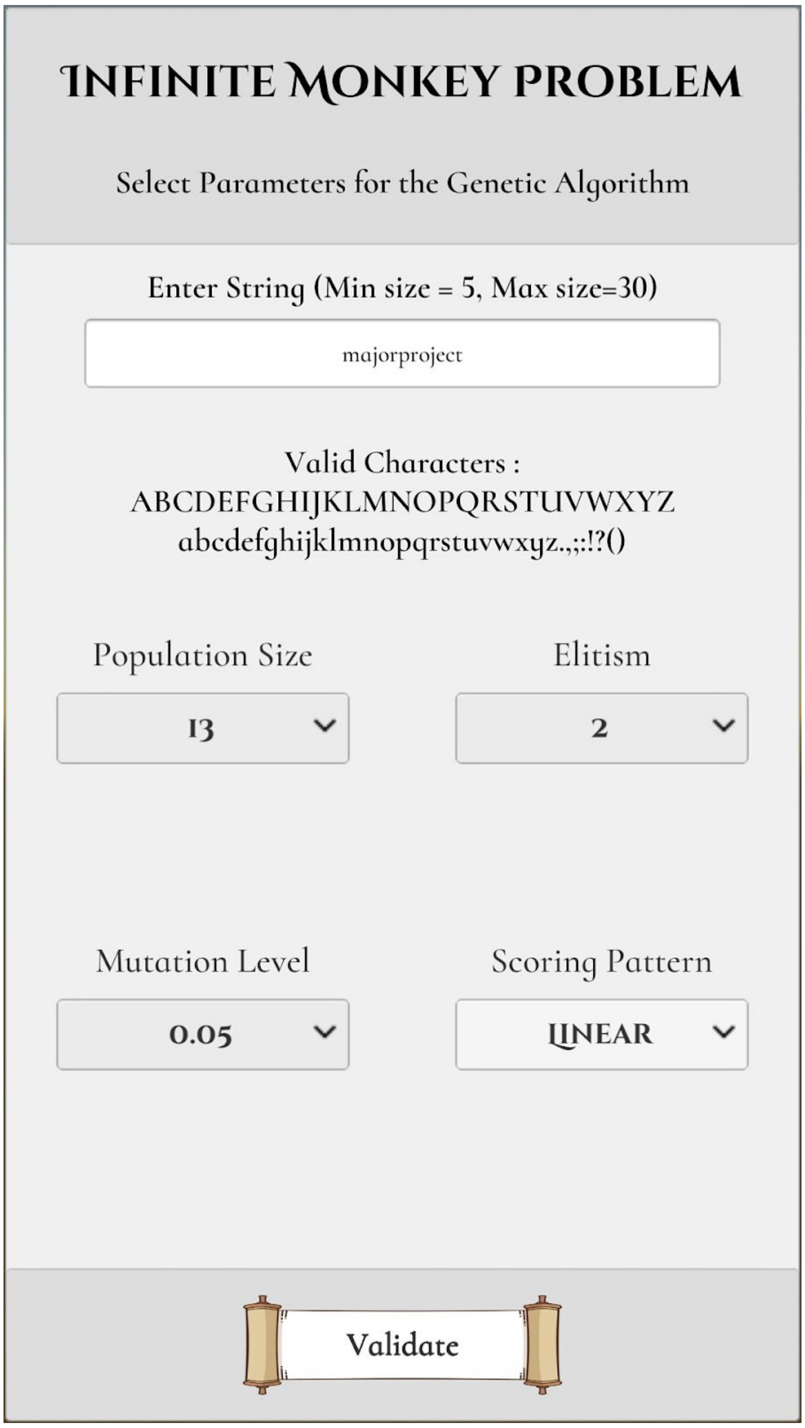
4.1.4 Observations Made

Based on the analysis of the data regarding the effect of various parameters on the efficiency of the genetic algorithm, following observations were made -

- Fig 2. indicates that the ability of the algorithm to arrive at a solution slows down with the increase in the length of the input string. This is solely because with increase in size of the target string there are more characters in the string of each individual all of which must be correctly predicted by the algorithm to reach the desirable output. However while the rise in no. of generations required to reach the target string is almost linear for the Exponential scoring pattern, it is way more steep for Linear scoring pattern.
- The effect of increasing population size is visualised in Fig 3. With rise in the size of the population, there are more individuals per generation to choose from for crossover. However, no significant improvement is noticed with the Linear scoring pattern. But with the Exponential scoring pattern, the algorithm arrives at a solution much more quickly with increase in the size of the population.
- Elitism denotes how many of the fittest individuals are copied to the next generation without any crossover or mutation. With increase in elitism, significant improvement is observed as seen from Fig 4. A slightly even better improvement is noticed when population size is high.
- The effect of low level of mutation as well as high level of mutation can be seen from Fig 5. At high level of mutation such as 0.1 i.e. 10% the algorithm struggles to arrive at a solution because it diverges repeatedly from its best solution. On the other hand, low level of mutation doesn't offer enough options in the search space any given time to arrive at the solution quickly. The mutation value of 0.04 i.e. 4% was observed to be the best for the Infinite Monkey Problem.
- The effect of scoring pattern chosen is clearly evident from Fig 6. The Exponential scoring pattern outperforms Linear scoring pattern in all cases. This is because exponential scoring pattern gives a lot more weightage to any new improvement in the solution.

4.1.5 Screenshots

Following screenshots of the Android application interface for this project display the working of the genetic algorithm for the Infinite Monkey Problem -


INFINITE MONKEY PROBLEM

Select Parameters for the Genetic Algorithm

Enter String (Min size = 5, Max size=30)

majorproject

Valid Characters :
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz.,;:!?()

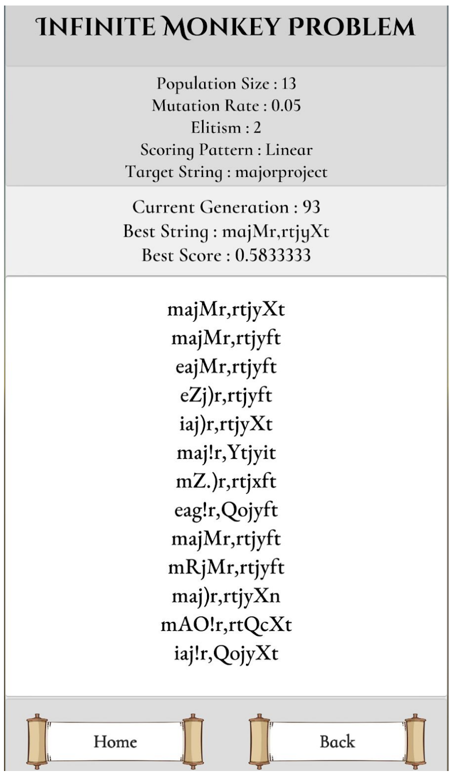
Population Size: 13

Elitism: 2

Mutation Level: 0.05

Scoring Pattern: LINEAR

Fig 7. Setting up parameters for the Infinite Monkey Problem



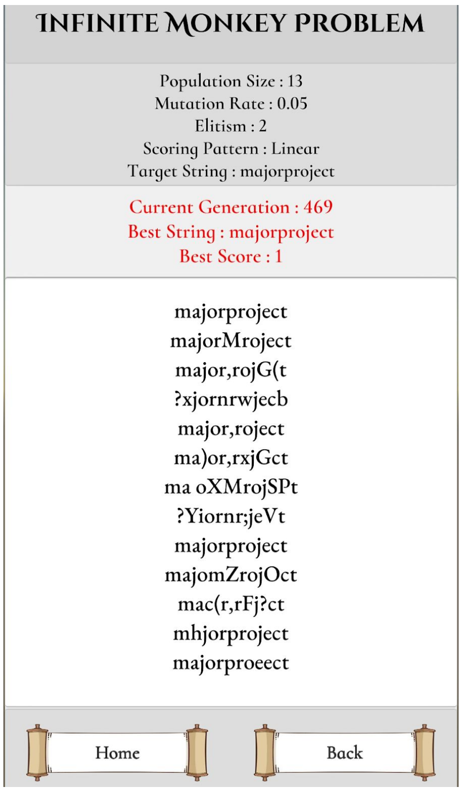
INFINITE MONKEY PROBLEM

Population Size : 13
Mutation Rate : 0.05
Elitism : 2
Scoring Pattern : Linear
Target String : majorproject

Current Generation : 93
Best String : majMr,rtjyXt
Best Score : 0.5833333

```
majMr,rtjyXt
majMr,rtjyft
eajMr,rtjyft
eZj)r,rtjyft
iaj)r,rtjyXt
maj!r,Ytjyit
mZ.)r,rtjxft
eag!r,Qojyft
majMr,rtjyft
mRjMr,rtjyft
maj)r,rtjyXn
mAO!r,rtQcXt
iaj!r,QojyXt
```

Fig 8. Algorithm for Infinite Monkey Problem in Progress



INFINITE MONKEY PROBLEM

Population Size : 13
Mutation Rate : 0.05
Elitism : 2
Scoring Pattern : Linear
Target String : majorproject

Current Generation : 469
Best String : majorproject
Best Score : 1

```
majorproject
majorMroject
major,rojG(t
?xjornrwjecb
major,roject
ma)or,rxjGct
ma oXMrojSPt
?Yiornr;jeVt
majorproject
majomZrojOct
mac(r,rFj?ct
mhjorproject
majorproeect
```

Fig 9. Final result for the Infinite Monkey Problem

4.2 Travelling Salesman Problem

4.2.1 Overview

In this project, we have used a genetic algorithm to tackle the Travelling Salesman problem. In this problem, a salesman has to visit ‘n’ nodes in a cyclic manner in such a way that the total distance of the loop is minimal. This is an NP-hard problem and its solution using merely brute force becomes computationally infeasible as the no. of nodes increases.

Genetic algorithms do not promise to solve this problem but can provide a near optimal solution quickly than conventional brute force methods. For most real life applications of this problem, such a solution is efficient. Genetic algorithm works by initially predicting random sequences of nodes which indicate a cyclic path from one node to the next node in the sequence and from the last node to the first. It then calculates the fitness value of each such sequence and identifies the best ones and recombines them to create a new generation of such nodes.

To set up the algorithm, following parameters need to be provided in the application interface -

- Mutation Rate
- Elitism
- Nodes on a 2D grid

For simplicity, the population size has been hard coded to 20.

4.2.2 Working

The algorithm functions in the following way -

1. Generation count is set to 1. For ‘n’ nodes, ‘n’ integer indexes from ‘0’ to ‘n-1’ are used to denote and refer to them. They are put in an array and shuffled 20 times to create the first population of random cyclic sequences of these nodes.
2. Fitness value of each sequence is calculated based on the total length of the loop formed by such a cyclic sequence. The entire population is sorted based on this fitness value.

3. The best sequence so far is updated.
4. An empty set for a new population is created. Fittest sequences from the current generation equal in count to the elitism value are copied to this new population.
5. Rest of the set of the new population is populated by selecting sequences from the current generation based on the weighted fitness sum method and applying the crossover function on them.
6. Each new sequence in the new generation then subjected to mutation via the Mutation operator and the defined mutation rate.
7. The algorithm goes back to step 2.

Since it is not possible to say whether a particular sequence is the best possible combination (without comparing all possible combinations as in brute force method), the algorithm never terminates by itself. The user can see the current best sequence and decide whether it meets his/her desired level of quality and choose to terminate the application.

4.2.3 Parameters Used

Elitism

Elitism value denotes how many sequences would be copied directly to the next generation. Elitism doesn't allow the system to diverge from its best solution so far and makes that the best solution of the next generation is at least as fit as the best solution of the current generation.
Elitism much be considerably lower than the population size so as to leave enough members in the current generation for effective crossover.
The application interface allows the user to choose between 6 elitism values - 0, 1, 2, 3, 4 and 5.

Mutation Rate

Mutation prevents premature convergence to an inefficient solution. Random mutation allows the system to explore a wider range of solutions which is necessary to reach the global maxima.

Lower level of mutation is slow in exploring more varied solutions whereas significantly higher level of mutation causes the system to continuously diverge in large amounts. It is necessary to have the appropriate level of mutation for any genetic algorithm.
This application allows for 11 possible mutation rates for this problem ranging from 0.00 to 0.10 in increments of 0.01. A mutation value of 0.01 denotes 1% chance of mutation wherein in a particular sequence there is 1% probability that any two random nodes in that sequence would be switched.

Nodes on a 2D Grid

The application allows for selection of nodes on a 2D grid which represent positions on a 2D plane. The no. of nodes selected denote the size of the genome of each individual in the algorithm i.e. the length of all the sequences generated. The nodes can be selected by the user by merely tapping on them. To give a clear idea of the working of the genetic algorithm, a minimum of 10 nodes need to be selected. The maximum no. of nodes that can be selected is capped to 30 for sheer simplicity.

Fitness Function

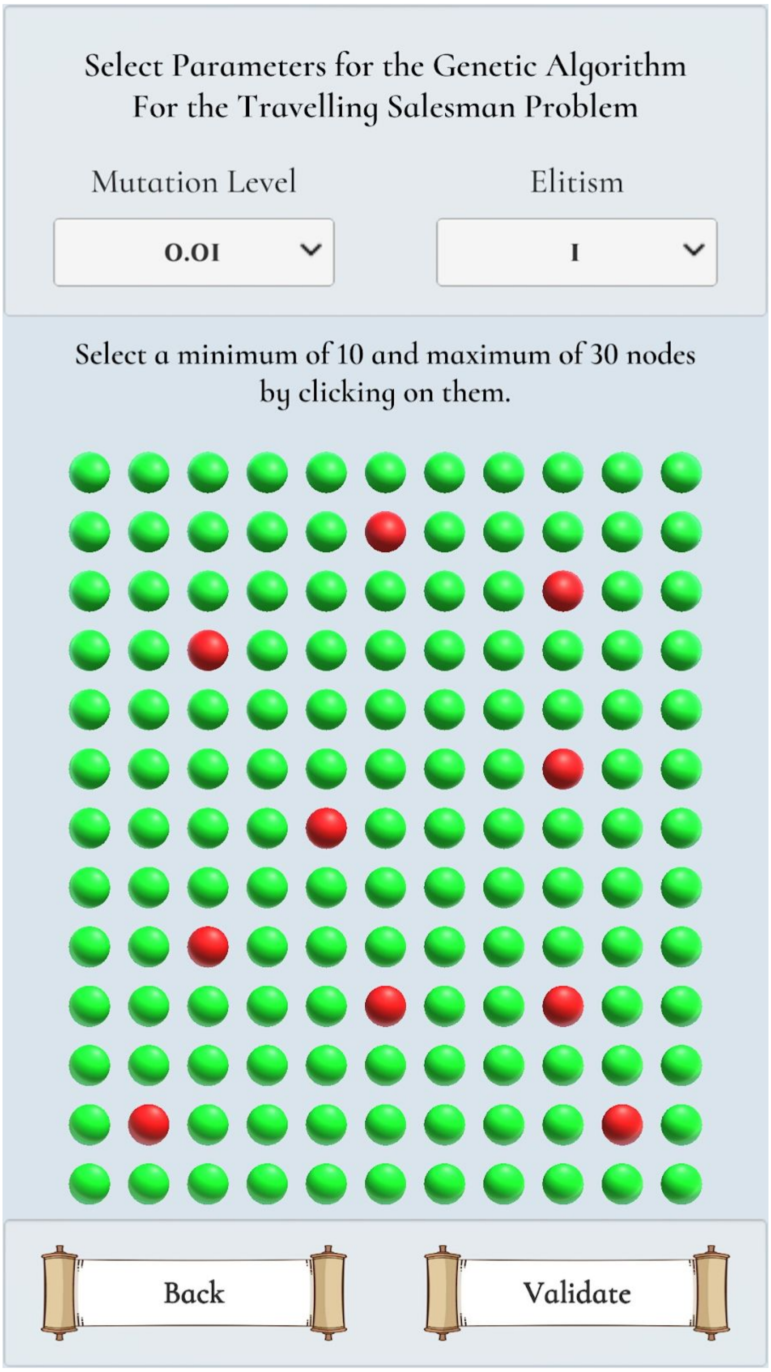
Since fitness of each sequence is inversely proportional to the absolute length of the cyclic route that is formed by that sequence (i.e. shorter routes are preferred over longer ones), the following fitness function is used to calculate the fitness value -

100/Total absolute distance of the route

Herein, smaller distances have larger fitness values and longer routes have smaller fitness values.

4.2.4 Screenshots

Following screenshots of the Android application interface for this project display the working of the genetic algorithm for the Travelling Salesman Problem -



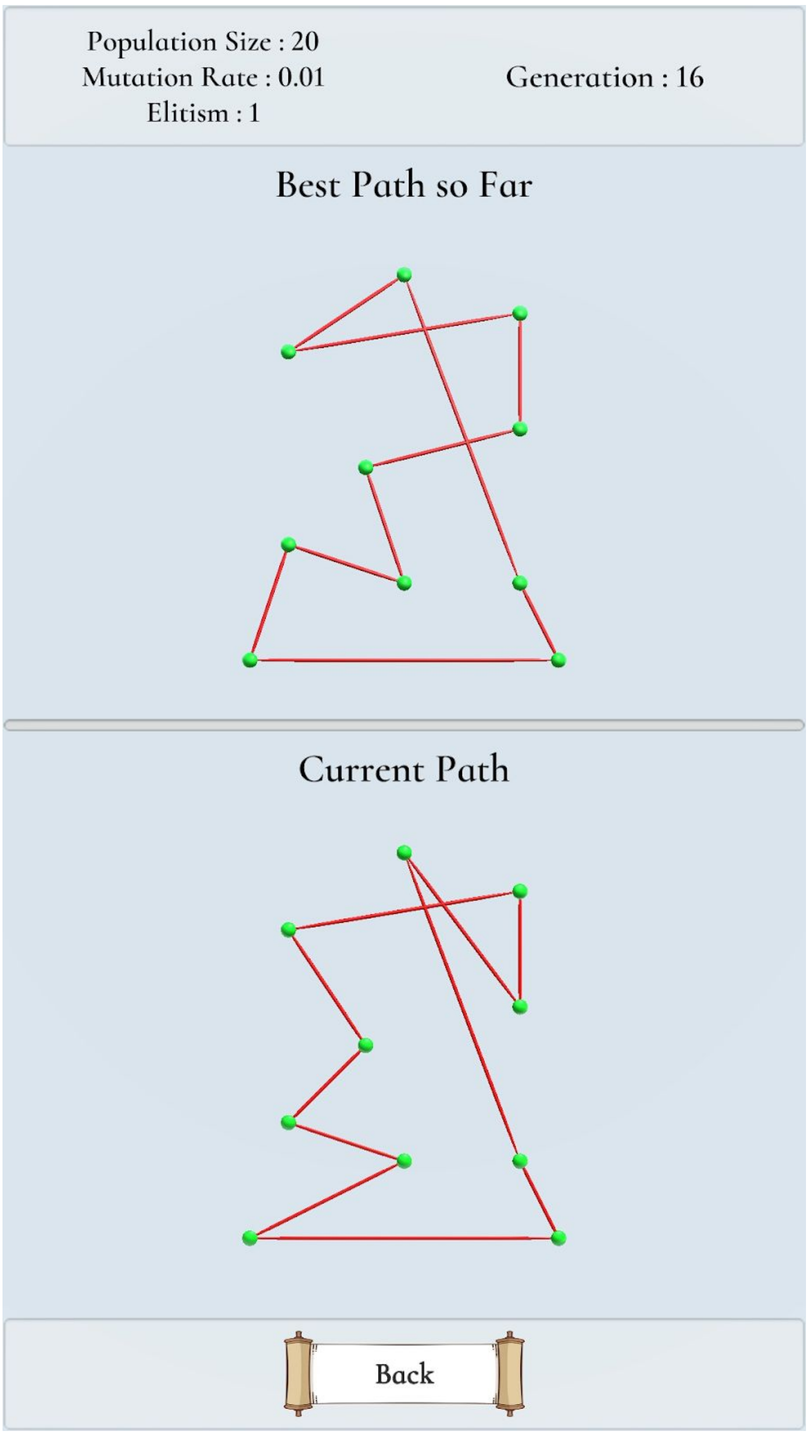
Select Parameters for the Genetic Algorithm
For the Travelling Salesman Problem

Mutation Level: 0.01

Elitism: 1

Select a minimum of 10 and maximum of 30 nodes by clicking on them.

Fig 10. Selection of parameters for the Travelling Salesman Problem



Population Size : 20
Mutation Rate : 0.01
Elitism : 1

Generation : 16

Fig 11. Working of the genetic algorithm for the Travelling Salesman Problem

4.3 Autonomous Car Simulation

4.3.1 Overview

Genetic algorithms can be used to train the various attributes of autonomous movement over various generations. This functionality is often used in robotics and for various characters in games to train them for autonomous locomotion. The genetic algorithm however by itself doesn't move the object. It only finely tunes the variables responsible for movement and the actual movement is controlled by some other system.

In this project, we have utilised genetic algorithms to train a simulation of an autonomous car. The actual movement in this simulation is carried out by a feedforward neural network. The genetic algorithm over time sets up the weights and biases of this neural network in such a way that the neural network can now correctly predict where the car should move so as to avoid all obstacles coming its way.

For simplicity, following values of the given parameters have been used in the genetic algorithm-

- Population Size = 20
- Mutation Rate = 0.05 i.e. 5%
- Elitism = 5

The following values have been used for the car simulation -

- Max Velocity = 7 units
- Acceleration/Deceleration = 1.2 units
- Angular Velocity = 30 units
- Sensor Angle = 70 degrees
- Sensor Length = 10 units
- No. of Sensors = 5

4.3.2 Working

The algorithm functions in the following way:

1. Generation count is set to 1. A sequence of values is initialised for each individual in the initial population. This sequence denotes the value of weights and biases for that particular individual which will be used to calculate movement via a neural network.
2. Fitness value of each individual i.e. car in the current population is calculated and they are sorted based on that. Fitness value is calculated by testing how well this car behaves in the simulated environment.
3. An empty set of new population is created wherein sequence of weights and biases of the fittest cars equal in count to the elitism value are copied from the current generation to the new generation.
4. Rest of the new population is populated by selecting parent cars from the current generation using the weighted fitness sum method. They are used to generate new child cars using the Crossover operator.
5. The child cars are also subjected to mutation via the Mutation operator.
6. The algorithm goes back to step 2 after incrementing the generation count.

4.3.3 Parameters Used

Neural Network

The neural network used in this simulation is a convolutional feedforward neural network. Herein its weights and biases are unique for each car and are stored as its attributes which are then applied to the neural network when that car needs to be simulated for autonomous driving.

The neural network used consists of 3 layers -

- Input Layer - The size of the input layer is one greater than the no. of sensors. The input from the sensors as well the current velocity are used as inputs in this input layer.
- Hidden Layer - It is used to allow the neural network to understand and predict complex behaviour. The size of the hidden layer used in this network is 10.

- Output Layer - This gives the output data for movement. In this simulation, the output layer gives out 2 real values - the first denotes the amount of forward acceleration/deceleration and the second denotes the direction and amount of turn.

This neural network doesn't have any backpropagation algorithm as its training is controlled by the genetic algorithm.
The activation function used in this neural network is the sigmoid function.

Mutation

The mutation value in this algorithm denotes how many weights and biases on average would be replaced by random values for a particular car. Here, mutation value of 5% i.e. 0.05 has been used

Elitism

The elitism value denotes how many top fit cars would be copied to the next generation without modification. This allows that the best performing cars of the current generation aren't lost in the future. Here elitism value of 5 has been chosen for a population size of 20.

Fitness Function

The fitness function in this algorithm outputs a positive value denoting the fitness value of a particular car. A higher fitness value denotes that a particular car is more fit than the other. The function used is defined below and depends on the net distance covered by the car -

Total distance covered by the car ^ 2

The square nature of the fitness function causes any new improvement to have more weightage than any previous similar improvement. This allows the cars to improve in performance quickly.

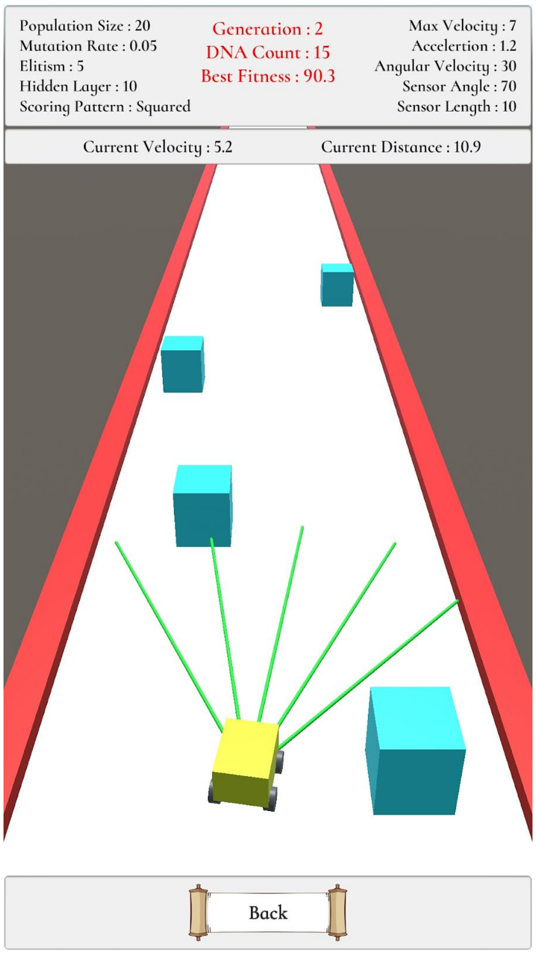
Sensors

A total of 5 sensors have been used which alongside the current velocity serves as input for the neural network that defines the movement variables for any car. The sensors project out of the front of the car spanning an angle equal to the Sensor angle (Here 70 degrees) and shoot out as far as the Sensor length (Here 10 units).

The value from the sensors ranges from 0 to the value of the sensor length. If the sensor hits any obstacle, the value from that sensor is the distance between the car and the point where the sensor hits the obstacle. If it doesn't hit any obstacle, the value from the sensor which serves as an input to the neural network is equal to the sensor length.

4.2.4 Screenshots

Following screenshots of the Android application interface for this project display the working of the genetic algorithm for the Autonomous Car Simulation -



Population Size : 20
Mutation Rate : 0.05
Elitism : 5
Hidden Layer : 10
Scoring Pattern : Squared

Generation : 2
DNA Count : 15
Best Fitness : 90.3

Max Velocity : 7
Accelertion : 1.2
Angular Velocity : 30
Sensor Angle : 70
Sensor Length : 10

Current Velocity : 5.2
Current Distance : 10.9

Fig 12. Working of Autonomous Car Simulation

CONCLUSION

Summary

Genetic Algorithms served us in multiple ways from simple string generation, route optimization to training a 3D model of a self driving car.

- The program solved the ‘Infinite Monkey theorem” by finding the string matching exactly to the user input in efficient time by learning over generations.
- The program found a close to optimal shortest closed route between a given set of points in a small amount of time.
- The program trained a self driving car and simulated it to drive along a path while avoiding obstacles.

Future Scope

Genetic Algorithms are an advanced way to solve problems in a more accurate method and hence have a significant future scope in various applied fields.
Genetic Algorithms can further be used to train other robotic models in locomotion and eventually help them in progressing to more complicated paths.
It can be used to find efficient routes through even the most complicated set of points and find solutions to problems that cannot be directly tackled by humans.
Genetic Algorithms use the fail-proof method of testing every possible solution however in a more refined manner and hence prove to be among the most efficient.

Genetic algorithm is a probabilistic solving optimization problem which is modeled on a genetic evaluations process in biology and is focused as an effective algorithm to find a global optimum solution for many types of problem. This algorithm is extremely applicable in different artificial intelligence approaches as well as different basics approaches like object oriented, robotics and others. In future we shall concentrate on the development of hybrid approaches using genetic algorithm and object oriented technology.

REFERENCES

1. L. Haldurai, T. Madhubala and R. Rajalakshmi, “A Study on Genetic Algorithm and its Applications,” IJCSE, Vol. 4, Issue 10, Oct. 2016.
2. Pushpendra Kumar Yadav and Dr.N.L.Prajapati, “An Overview of Genetic Algorithm and Modeling,” IJSRP, Vol. 2, Issue 9, Sept. 2012.
3. Neeraj Gupta, Nilesh Patel, Bhupendra Nath Tiwari, and Mahdi Khosravy, “Genetic Algorithm based on Enhanced Selection and Log-scaled Mutation Technique” in Proceedings of the Future Technologies Conference (FTC) 2018, pp.730-748.
4. Kenneth De Jong, “Learning with genetic algorithms: An overview,” Machine Learning, Vol. 3, Issue 2–3, pp 121–138, Oct. 1988.
5. Goldberg D. E., & Lingle R, “Alleles, loci, and the traveling salesman problem” in Proceedings of the First International Conference on Genetic Algorithms and Their Applications, (pp. 154–159).
6. Grefenstette J., Gopal R., Rosmaita B., and VanGucht D., “Genetic algorithms for the traveling salesman problem” in Proceedings of the First International Conference on Genetic Algorithms and their Applications, pp. 160–168.
7. D. E. Goldberg and K. Deb and J. H. Clark, "Genetic Algorithms, Noise, and the Sizing of Populations", Complex Systems, 1992.
8. David E. Goldberg and John H. Holland, “Genetic Algorithms and Machine Learning,” Machine Learning, Vol. 3, Issue 2–3, pp 95–99, Oct. 1988.
9. Keshav Bimbraw, “Autonomous Cars: Past, Present and Future” in Proceedings of the 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO-2015), pp 191-198.
10. Yago Saez, Diego Perez, Oscar Sanjuan and Pedro Isasi, “Driving Cars by Means of Genetic Algorithms” in Parallel Problem Solving from Nature - PPSN X, 10th International Conference Dortmund, Germany, September 13-17, 2008.

11. Pooja Vaishnav, Dr. Naveen Choudhary and Kalpana Jain, “Traveling Salesman Problem Using Genetic Algorithm: A Survey,” IJSRCSEIT, Vol. 2, Issue 3, 2017.
12. Jia Xu, Lang Pei and Rong-zhao Zhu, “Application of a Genetic Algorithm with Random Crossover and Dynamic Mutation on the Travelling Salesman Problem,” ICICT, 2018.
13. Beasley, David, Bull, David R. and Martin, Ralph Robert, “An overview of genetic algorithms: Part 1, fundamentals,” University Computing 15, pp 56-69.

Research Paper on Analysis of Genetic Algorithm in Infinite Monkey Problem

Ms. Deepti Gupta
Department of Computer Science and Engineering
Maharaja Agrasen Institute of Technology, Delhi

Ms. Isha Negi
Department of Computer Science and Engineering
Maharaja Agrasen Institute of Technology, Delhi

Mr. Mayur Garg
Department of Computer Science and Engineering
Maharaja Agrasen Institute of Technology, Delhi

ABSTRACT

Revolutionary enhancements have been made in the field of Artificial Intelligence and Machine Learning. One of the key aspects of the latter is to solve problems of which little is known in advance. Such problems encompasses a very broad search space and it is, more often than not, tedious to design a situation specific algorithm. In such cases, adaptive methods such as Evolutionary algorithms are preferred. One of the most widespread used evolutionary algorithms is Genetic Algorithms. Genetic algorithms mimic the processes involved in natural selection and evolution to repeatedly improve on its solution and arrive at an optimal or near optimal result. Genetic algorithms, or evolutionary algorithms in general, are extensively used to solve many search and optimisation problems specially NP-hard problems where it is computationally infeasible to arrive at an optimal solution but quite often a near optimal solution is sufficient. Other problems which can be solved by genetic algorithms include problems where it is unclear as to how an algorithm must be designed to solve it. Such problems such as the designing character movement in games can be tackled by Genetic algorithms as long as we can define a fitness function to evaluate the effectiveness of our result. In the suggested system, we have aimed to implement Genetic Algorithms to tackle the Infinite Monkey Problem. We have aimed to understand the basic functioning of the simple Genetic Algorithm used and have expected to find a correlation of various parameters in the Genetic Algorithm used such as the Length of the input string, Size of the population, Mutation Rate, Elitism and Scoring pattern with the associated efficiency of the algorithm in arriving at a solution by varying these parameters in a suitable range.

I. INTRODUCTION

Genetic algorithms are widely used to solve optimisation problems by an intelligent exploitation of the random search. They can be also used to solve problems where it is possible for us to improve over our previous solution by evaluating its effectiveness and use that very solution to arrive at an even better one.

In this case, we have used a simple Genetic Algorithm to tackle the Infinite Monkey Problem wherein we have defined the three operators of the genetic algorithm - Selection, Crossover and Mutation to train the system and arrive at a solution. We have also analysed the effect of various parameters in the algorithm used such as the Length of the input string, Size of the population, Mutation Rate, Elitism and Scoring pattern on the efficiency of the algorithm of arriving at a result.

Infinite Monkey Problem

Infinite Monkey Theorem or Infinite Monkey Problem is a classical example that illustrates the ineffectiveness of the brute force approach or the