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 idea that random inputs or instructions if given repeatedly would eventually generate the correct output. The Infinite Monkey Theorem states that given a monkey who is hitting keys randomly on a keyboard repeatedly and an infinite amount of time, it will almost surely type any piece of text such as Hamlet by Shakespeare. However, the odds of that happening within the timespan equal to the age of this universe even if each atom in the universe was such a monkey is unfathomably low but technically not zero. This problem gives a concrete idea of the magnitude of large numbers and states that in the operational stance, it impossible to generate an effective outcome from purely random sequences.

Genetic Algorithm
Genetic Algorithms are adaptive and evolutionary algorithms that mimic the notions of natural selection and evolution to produce results. Herein we start from purely random solutions and combine the best such solutions to generate the next generation of solutions which are in most cases better than the previous ones. This process is repeated over and over to arrive at a desirable output.
Three basic operator functions are defined in a genetic algorithm which are Selection, Crossover and Mutation. These operators are used to implement the algorithm which functions in the following sequence -

1. A set of population of solutions is initially randomly generated.
2. A fitness value which defines the effectiveness or correctness of each solution is evaluated.
3. Based on fitness values, some solutions are selected to mate via the Selection operator.
4. The attributes of the solutions are mixed via the Crossover operator and then the resulting solution is mutated via the Mutation operator.
5. Using this procedure, an entire new generation of solutions is generated and the procedure repeats until one or more solutions reaches the desired fitness value.

In this case, 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.

II. METHODOLOGIES

Functioning of the Genetic Algorithm
The algorithm used in this case is a simple genetic algorithm which takes in parameters such as -

- Target String
- Population Size
- Elitism
- Mutation Rate
- Scoring Pattern

These parameters defines the various attributes of the genetic algorithm and consequently affect its behaviour. The algorithm would then start from a population of purely randomly generated strings and learn over various generations to reach the target string. The number of generations required to reach the target string is then displayed. For the following case, these parameters were varied to take up a range of values so that the their effect on the effectiveness of the genetic algorithm could be studied i.e. the variation in the number of generations required to reach the target string with variation in these parameters.

For this genetic algorithm that tries to tackle the Infinite Monkey Problem, following operators or functions were defined -

Random Gene Generation
A set of valid characters was defined which contains all the characters that were allowed to be in the target string, initial random strings or any intermediate strings generated. A total of 61 unique characters were in that set (including a blank space) which are all listed below -

ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz.,;:!?()

For generating a random gene during string mutation, the system would randomly pick any random character from the set and assign it to the necessary gene in the string. For generating an initial random string, the system would repeatedly generate a random gene for all the gene positions in the string until the length of the random string was equal to the target string.

Selection Operator
To select parent genes for crossover, it is necessary to calculate the fitness of all the solutions in the population so that better fit solutions can be preferred for crossover. To do this a fitness function has been defined. Each character in a solution is compared to the corresponding character in the target string and the no. of matching characters are identified. Based on the number of such matching characters, a fitness value is calculated which ranges from 0 (i.e. no matching characters) to 1 (all characters matched and the target string has been reached). To calculate this fitness values, two Scoring Patterns have been identified out of which any one can be chosen - 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)

Both Scoring patterns yield a fitness value between 0 and 1 for any given string.
To select parents for crossover, weighted fitness sum method was used. Fitness sum value of all the strings in a particular generation is calculated. Based on this sum value, those parents are selected which have the highest fitness value but also leaves a small chance that a parent with a smaller fitness value can be selected. The latter case allows the system to not be stuck on the local maxima.

Crossover Operator
Once two parent genes have been selected, it is now necessary to combine their genes (here the characters in the strings) in such a way that the child element has some traits of each of its parents. This is done by initialising a string of size equal to the target string and randomly assigning to its corresponding gene one of the corresponding gene value from any one of its parents. For instance, for

a string of size ‘n’, the system allocates to its first gene the value of the first gene of any of its parents, to its second gene the second gene of any of its parents and so on.

Mutation Operator
Once a child gene has been created by the Crossover operator, Mutation operator is applied so that the system doesn’t get stuck on the local maxima and can explore a wider range of solutions. To apply the mutation operator, a mutation rate is supplied. The algorithm then iterates over the entire child string and applies a mutation by assigning to that character a random gene if a random value then generated is smaller than the mutation rate. Hence a mutation rate of 0.01 suggests that on average 1% of characters of the child string would be mutated.

Technologies Used

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 three-dimensional and two-dimensional 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.

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.

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.

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.

IDLE
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.

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.

III. EXPERIMENTAL RESULTS

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.

For the analysis of this genetic algorithm, we have used 3 different target strings which are listed below -

Hello World! (Length = 12)
Genetic Algorithms! (Length = 19)
Infinite Monkey Problem (Length = 23)

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 -

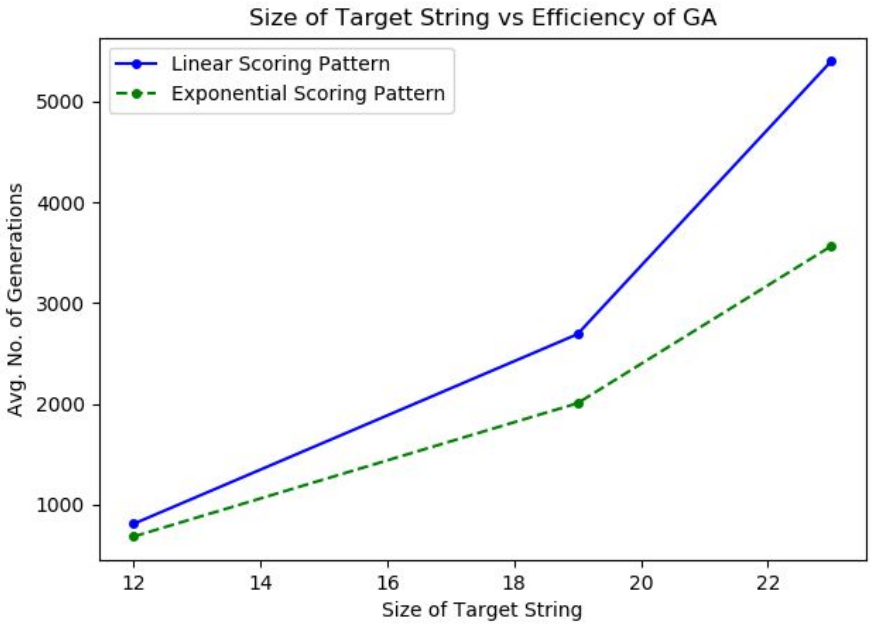

Fig 1. Effect of Size of Target String on Efficiency of GA

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. Population size must be significantly greater than the elitism value.

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 -

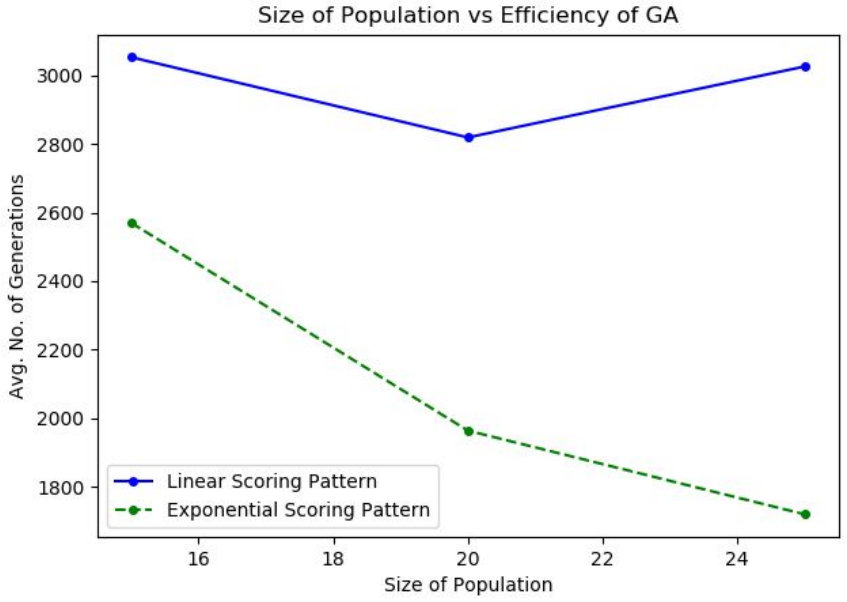

Fig 2. Effect of Population Size on Efficiency of GA

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 -

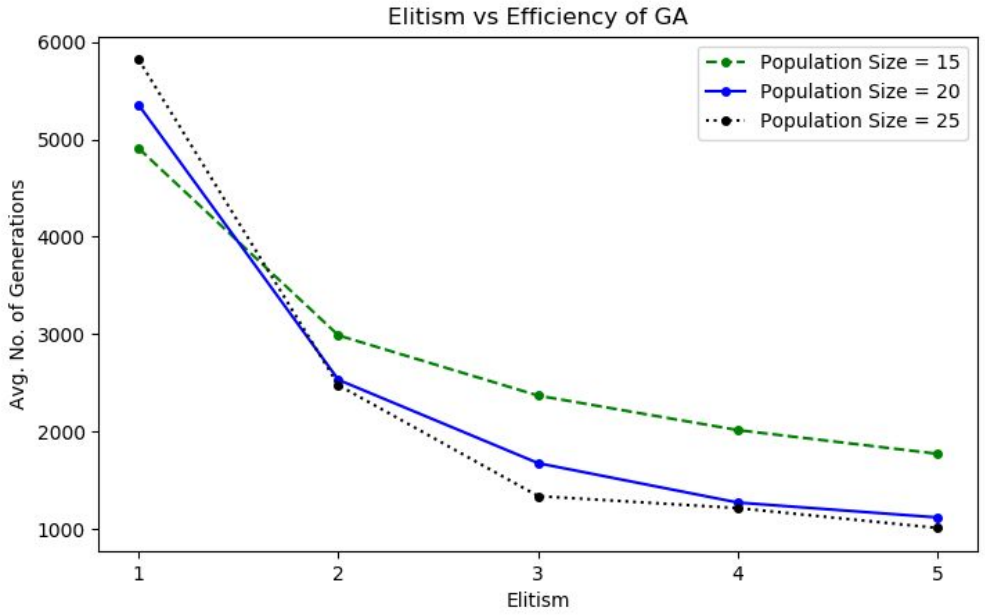

Fig 3. Effect of Elitism on Efficiency of GA

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.

Mutation makes sure that the algorithm doesn't get stuck on the local maxima instead of the global maxima by continuously diverting from the path. 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 -

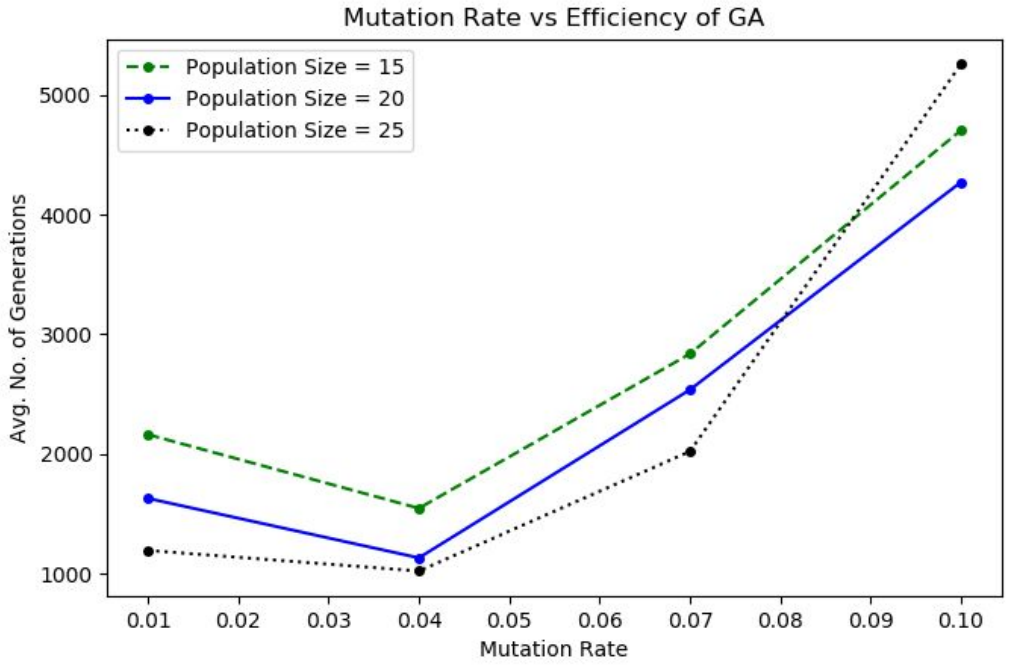

Fig 4. Effect of Mutation Rate on Efficiency of GA

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)

Variation in scoring patterns changes how much weightage in the fitness value the algorithm gives to each minor improvement in the solution. In case of linear scoring pattern, each new matching character has the same weight whereas in the case of exponential scoring pattern, each new matching character has significantly more weight in the fitness value than any previous matching character.

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 -

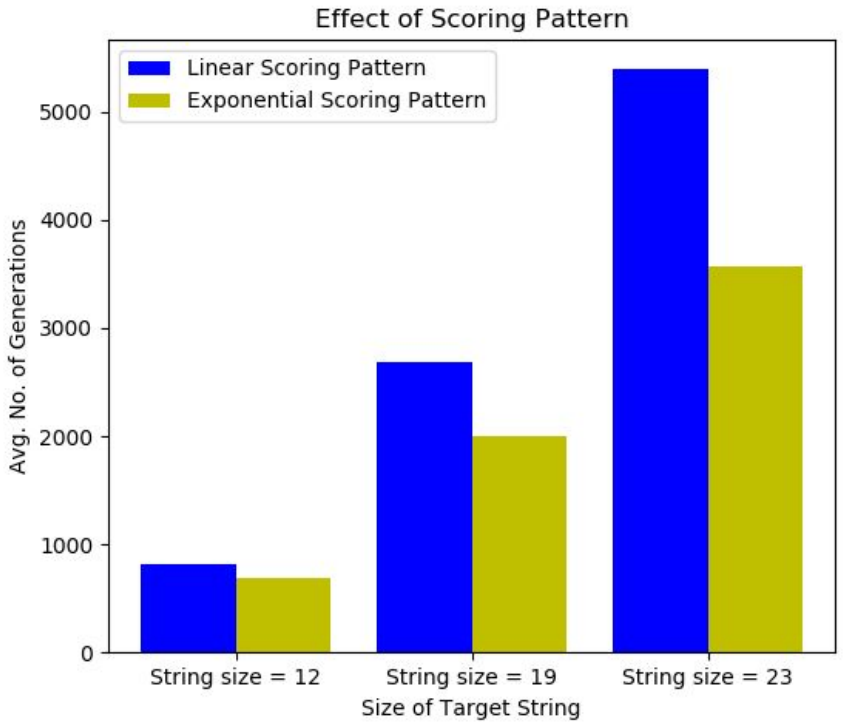

Fig 5. Effect of Scoring Patterns on Efficiency of GA

IV. 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 1. 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 attributes (here characters) in the genome of each individual all of which must be fine tuned 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 2. 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 3. Algorithms with any population size shows the similar level of improvement with increase in elitism. However a slightly 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 4. At high level of mutation such as 0.1 i.e. 10% the algorithm struggles to arrive at a solution because it deviates 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 this given problem.

- The effect of scoring pattern chosen is clearly evident from Fig 5. 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.

V. CONCLUSIONS

Summary

The genetic algorithm to tackle the Infinite Monkey Problem was successfully created and tested by tuning many parameters (Size of the target string, Size of the population, Elitism, Mutation rate and the scoring pattern) and their effect on the efficiency of the genetic algorithm was observed. It was seen that the algorithm performed better with a higher population size, higher elitism, mutation rate around 4% and Exponential scoring pattern. The effectiveness of the algorithm decreased significantly when mutation rate was very low or very high and when length of the target string was observed. When either of mutation and elitism were set to 0, the algorithm almost never arrived at the solution. 0 mutation lead to the algorithm being stuck wherein any new improvement couldn't be observed. 0 elitism lead to the system behaving quite like random search as it could no longer keep track of its best solution so far.

Future Scope

- Genetic Algorithms can be used to solve various kinds of optimisation and search problems. They are also being used to train movement of cars and robotics without explicit programming. The ability of the algorithm to arrive at solutions for problems by themselves which cannot be easily programmed by humans finds its uses in many real world applications.
- The knowledge of effect of various parameters on the genetic algorithm can be utilised to efficiently tune any genetic algorithm to get the best results.

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.