# A Heuristic Study Of Genetic Algorithm

## For Evolution

Hiran Harilal
Department of Computer Science
University of Hertfordshire
Hatfield, United Kingdom
hh16aca@herts.ac.uk

*Abstract*— **'Genetic algorithm' refers to a specific algorithm executed in a manner such that some specific sorts of problems are solved. Although most of the examples we create in this are built on the foundation of the formal Genetic Algorithm, we concentrate on the study of evolutionary theories rather than implementing the algorithm with great accuracy. Genetic Algorithms (GAs) was developed as a concept mostly by John Holland, a professor as the University of Michigan. He authored the book on Adaption in Natural and Artificial Systems and pioneered GA research. Today, GAs are served as an integral part of a wider field of research called 'Evolutionary Computing'. Randomized search heuristics like evolutionary algorithms are mostly applied to problems whose structure is not completely known but also to combinatorial optimization problems.**

**Keywords—Genetic-Algorithm; Evolutionary-Computing; Darwinian-Evolution;**

## I. INTRODUCTION

*"The fact that life evolved out of nearly nothing, some 10 billion years after the universe evolved out of literally nothing, is a fact so staggering that I would be mad to attempt words to do it justice."— Richard Dawkins*[1]

The initial study of evolutionary systems by computer scientists' dates to the 1950's and 60's. They introduced the concept that evolution could be used as an optimization tool for engineering problems. These systems were built on the idea of evolving a population of solutions to a given problem using operators based on the basic conception of natural genetic variation and natural selection. [2]

Genetic Algorithms are constructed based on a model of natural and biological evolution which was formulated initially by Charles Darwin. [3] Darwin put forward the Theory of Evolution which elucidates the process of adaption of a species by the Principle of Natural Selection, which favors the species for survival and further evolution based on the ability of adapting best to their environmental conditions.

The prevailing study on Molecular Genetics presents to us a relatively detailed understanding on the basic building block of every living being, which is encoded in the Deoxyribonucleic Acid (DNA) – a double stranded macromolecule of helical structure. There have been traditional methods used in information fusion, however with the introduction of the evolutionary approach, due to the inherent parallel nature, the ability to deal with difficult problems has been enhanced. However, the application of this algorithm is more of an art than science. Caution must be taken while choosing the right model and the parameters in order the gain an in-depth knowledge on the morphological development of the algorithm.

This paper aims to give a brief overview of basic and advanced concepts, models and variants of Evolutionary algorithms implementations and applications especially those in information fusion.

Here we use the concept of Randomization which a very integral part of problem specific algorithms. [4] Through this paper, we attempt to study the factors and the effect of each of them on the performance of the process of evolution. The performance may be evaluated based on the time taken for the completion of the evolution and the number of generations taken to reach the target sentence. This is done by altering few factors such as the mutation rate and the population, the details of which will be explained further down.

## II. DARWINIAN NATURAL SELECTION

Before we begin with the concept of Genetic Algorithm and the implementation of our simulation, it is necessary to have some pre-requisite knowledge on the three core principles of Darwinian Evolution. The presence of these three elements is mandatory for the process of natural selection to occur.

### *1. Heredity.*

This can be defined as the process by which the properties of parents are passed on to their children. If the creatures live long enough to reproduce, then the children of the next generations to come, inherit these traits.

### *2. Variation.*

In order to introduce variation in a population, there must be a variety of traits present in a population. Let's take an example of a population of beetles with no variation, that is they are the same. Without variation in this population, the children would be a carbon copy of their parents and of each other. There cannot be any creation of new traits and the process of evolution doesn't occur.

*3. Selection.*

In a population, there are some members that have the opportunity to be parents and pass down their genetic information while other do not. This mechanism is normally referred to as 'survival of the fittest'. Taking an example of a population of gazelles which are being chased by lions every day. The faster the gazelles are, the more likely they are to live longer as they can easily escape from the lions. Hence, they have a chance to reproduce and pass their genes down to their children. However, the term fittest can be misleading. Being fit in a population doesn't mean being bigger, faster or stronger. This might be the case in most instances, but natural selection works on the basis that some traits are better adapted for the creature's environment and therefore produce a greater likelihood of surviving and reproducing. This has nothing got to do with the creature being better or more physically fit.

## III. OBJECTIVE

Through this project, we aim to use random phrases to find the solution through simulated evolution. Now, it's worth noting that this problem (arrive at the phrase "to be or not to be that is the question") is a ridiculous one. Since we know the answer, all we need to do is type it. The steps followed in order to run a GA is given in the form of a sketch under implementation.
The point here is that there is more facilitation to testing our code when solving a problem with a known answer. Once we arrive at the solution to our problem we feel more confident in taking GA to the next level: solving problems with unknown answers. Hence, after performing our experiment with GA, if we successfully obtain our target phrase, 'to be or not to be', then we have accomplished the right GA. While checking on the accuracy of our GA, we aim to find out the factors affecting the performance of evolution and their effect on it and this form our main objective of this paper.

## IV. IMPLEMENTATION

Unity is a cross-platform game engine developed by Unity Technologies, which is primarily used to develop video games and simulations for computers, consoles and mobile devices.

*figure 1: Unity logo*

First announced only for OS X, at Apple's Worldwide Developers Conference in 2005, it has since been extended to target 27 platforms. Unity is an all-purpose game engine that supports 2D and 3D graphics, drag and drops functionality and scripting through C#.

*Creating a population:*

This process is done with virtual DNA, which can be described as a set of properties (we call them as genes), that determine the appearance or conduct of an element.
In our example, DNA is taken as a string of characters. In the discipline of genetics, there is a notable distinction between the terms genotype and phenotype. The genotype of an element can be defined as the actual genetic code, in our case, it is the digital information of the element itself. This gets disseminated from generation to generation. The phenotype is different from the genotype in the sense that it is defined as the expression of the data. This distinction gives us the solution to how to use genetic algorithms in one's work.
However, in our example, is that there is no distinction between these two. The DNA data itself is a string of characters and that very string is the expression of that data.
Hence at this stage, we create a population of N elements, each element having randomly generated DNA.

*Selection:*

The Darwinian Principle of *selection* is applied here. We need to gage the population and establish which members are fit to be selected as parents for the following generations. There are two steps to the process of selection.

- **Evaluate fitness:**

For the proper functioning of the Genetic Algorithm, it is necessary that we introduce the fitness function. This function describes the fitness of a given member of the population by generating a numeric score. However, this is not how the real-world works. Creatures are not given a score; either they survive, or they don't. However, in the case of traditional genetic algorithm, we try to obtain a optimal solution to a problem through evolution. For that, we should be able to numerically evaluate any given possible solution.

***Fitness = no. of correct characters***.

- **Create a mating pool.**

A better solution for the mating pool is to make use of the Probabilistic Method, which can be labelled as the 'wheel of fortune' (also known as the 'Roulette Wheel'). To exemplify this method, we consider an example where we have a

population of five elements, each with a fitness score.
Firstly, we normalize all the scores. Normalizing a set of fitness scores involves standardizing their range such that we take up any value lying between 0 and 1. This is taken as a percentage of total fitness. Adding up all fitness scores in the example given below, we get,

3 + 4 + 0.5 + 1.5 + 1 = 10

| Element | Fitness |
|---|---|
| A | 3 |
| B | 4 |
| C | 0.5 |
| D | 1.5 |
| E | 1 |

*Table 1: Element and Fitness.*

Dividing each score by the total fitness, we obtain the normalized fitness.

| Element | Fitness | Normalized Fitness | Expressed as a % |
|---|---|---|---|
| A | 3 | 0.3 | 30% |
| B | 4 | 0.4 | 40% |
| C | 0.5 | 0.05 | 5% |
| D | 1.5 | 0.15 | 15% |
| E | 1 | 0.1 | 10% |

*Table 2: Normalized fitness.*

Now, for the wheel of fortune.

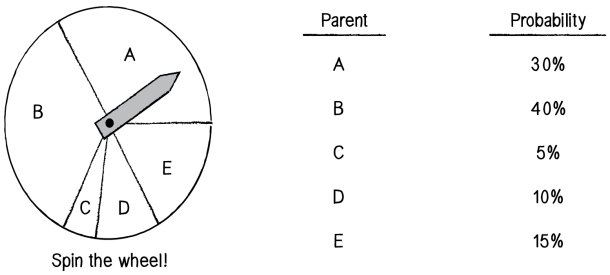



*Figure 1: Wheel of fortune.*

After spinning the wheel, we observe that the highest chance of being selected lies with Element B followed by A, D, E and finally C. This probability based selection based on fitness is an excellent approach. One, there is guarantee that the elements tagged with a high score will by most likely to reproduce. Two, it doesn't entirely eradicate any variation from the population. Unlike with the elitist method, the element scoring the lowest (C) has a chance of passing its information down to the future generation. There are possibilities that even the lowest scoring element have a tiny wad of genetic code that can prove to be truly beneficial and hence these should not be eliminated from the population.
For example, in the case of evolving 'to be or not be', we may have the following elements.

***A: to be or not to go***
***B: to be or not to pi***
***C: xxxxxxxxxxxxxxxxbe***

We can observe from this that, elements A and B have the highest score and hence are the most fit. But neither of the two contain the apt characters for the target phrase. Element C, though it has a very low score, it contains the genetic data for the target. Therefore, while we would want A and B to be selected to generate the majority of the next generation, we would want C also to have a small possibility to contribute to the reproductive process.
The next step after the selection of parents is the reproduction process to make the population's next generation, using the Darwinian Principle of heredity. There are variety of techniques that can be employed here. One such technique is asexual reproduction, where we pick just one parent and a child is created as an exact copy of the parent. However, the typical approach with Genetic Algorithm is to select two parents and creating a child using the following steps.

***1) Crossover.***
The process of generating a child out of the genetic code of two parents is called Crossover. In the case of our example, we pick two phrases from the mating pool.

**Parent A: FORK** : **Parent B: PLAY**

We need to make a child phrase from the two parents, A and B. The obvious technique would be to take the first two characters from A and the second two from B, as below,

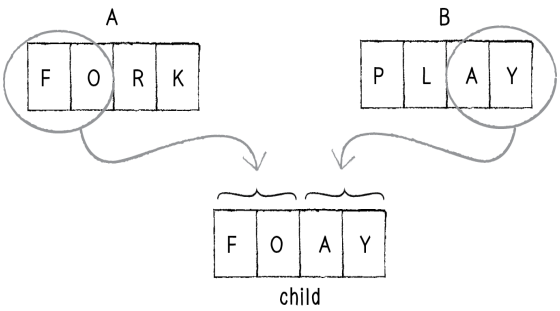



*Figure 2*

We can make a variation to this technique by picking out a random midpoint. We can either end up with FLAY or FORY. This is better than the 50/50 approach, as this increases the variation of possibilities for the following generation.

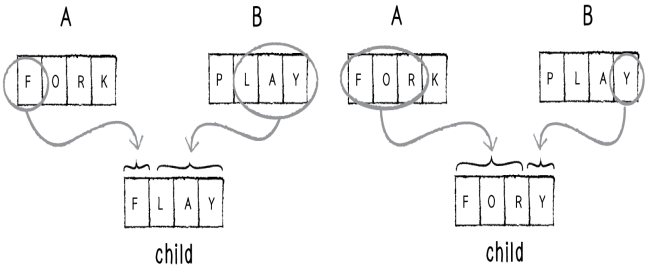



*Figure 3.*

Another approach is to select a parent at random for each character in the child string. It's like flipping a coin four times: heads from parent A, tails from parent B. Hence, we end up with different results like: PLRY, FLRK, FLRY, FORY etc.

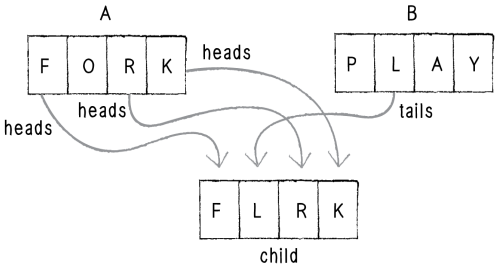
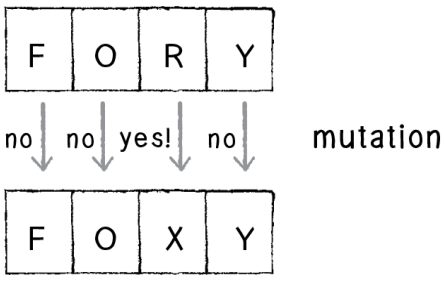



*Figure 4.*

Both of the strategies eventually do give the same results, however, if the order of the genetic information plays some role in expressing the phenotype, hence one may prefer one solution over the other.

***2) Mutation.***

The final process before adding the child to the next generation once the crossover is done is mutation. However, this is an optional step, there have been cases where it is unnecessary. This exists because of the Darwinian Principle of Variation. We create an initial population randomly, making sure that we start with a variety of elements. However, there is a limit to the amount of variety when seeding the first generation and mutation allows us to add more variety throughout the evolutionary process itself.




Mutation is represented in terms of a rate. For a given genetic algorithm, we can give the mutation rate as 5%,1%,0.1% etc. Assuming that we just finished with the crossover and ended up with the child FORY. If the mutation rate is 1%, it means that, there is 1% chance for each character generated from the crossover to mutate. However, a 1% probability is fairly low and most of the time mutation doesn't occur at all in a four-character string. Whenever it does, the mutation character is replaced with a randomly generate one.

As seen in some of the examples, the rate of mutation affects the behaviour of the system. A high mutation rate would negate the evolutionary process itself. If most of the child's genes are randomly generated, there is no guarantee that the more 'fit gene' would occur with greater frequency with the succession of generations.

The process of selection and reproduction is repeated over and over again, N times till we have the new population of N elements. Here, the new population of children constitutes the current population and we loop back to evaluate fitness and carry out selection and reproduction again.

Translating all of the above-mentioned steps of the Genetic Algorithm into Processing code, we get the following:

***SETUP:***

Step 1: ***Initialize***. Create a population of N elements, each with randomly generated DNA.

***LOOP:***

Step 2: ***Selection***. Evaluate the fitness of each element of the population and build a mating pool.

Step 3: ***Reproduction***. Repeat N times:

a) Pick two parents with probability according to relative fitness.

b) Crossover—create a "child" by combining the DNA of these two parents.

c) Mutation—mutate the child's DNA based on a given probability.

d) Add the new child to a new population.

Step 4. Replace the old population with the new population and return to Step 2.

## V. EXPERIMENTAL SETUP

**Experiment:**

The experiments are conducted in the Unity software and the time taken is manually recorded.

The observed values are then taken for further calculations and analysis on Excel.

**Aim:**

To study the factors affecting the performance of evolution.

**Objective:**

Since there are two main factors affecting the performance, we have split the experiment into two cases, giving us two main objectives:

1. To check the dependence of population with number of generation and time keeping mutation rate as constant.
2. To check the dependence of mutation rate with number of generation and time keeping

population as constant.

**Procedure:**

1. To check the dependence of population with number of generation and time [Mutation rate = K]

In this case we take the mutation rate as constant and vary the population. We take mutation rate as 1% and 5 different values of populations.
The target phrase to be obtained was "to be or not to be". The values that were obtained as output were, Number of generations until target phrase and the time taken. These determine the performance of evolution.
The values are obtained and recorded on Excel in the form of a table against each population

2. To check the dependence of mutation rate with number of generation and time [Population = K]

In this case we take the population as constant and vary the mutation rate. We take the population as 1000 and 5 different values of mutation rates.
The target phrase remains the same, "to be or not to be"
The values that were obtained as output were the same in this case as well, Number of generations until target phrase and the time taken. These determine the performance of evolution.
The values are obtained and recorded on Excel in the form of a table against each population

## VI. Observations and Findings

As the experiment was divided into 2 parts, there are 2 sets of observations:

1. When mutation rate is kept constant;

| Total Population | Mutation Rate | No of Generations until phrase solved | Total time (in sec) |
|---|---|---|---|
| 150 | 1% | 1002 | 18.8 |
| 300 | 1% | 648 | 10.1 |
| 600 | 1% | 256 | 4.9 |
| 1000 | 1% | 71 | 1.8 |
| 50,000 | 1% | 27 | 4.3 |

*Table 3: Obtained Data (Case I)*

Notice how increasing the population size drastically reduces the number of generations needed to solve for the phrase. However, it doesn't necessarily reduce the amount of time. Once our population balloons to fifty thousand elements, the sketch runs slowly, given the amount of time required to process fitness and build a mating pool out of so many elements. (There are, of course, optimizations that could be made should you require such a large population.) The above observation would be clearer with the help of a graph:

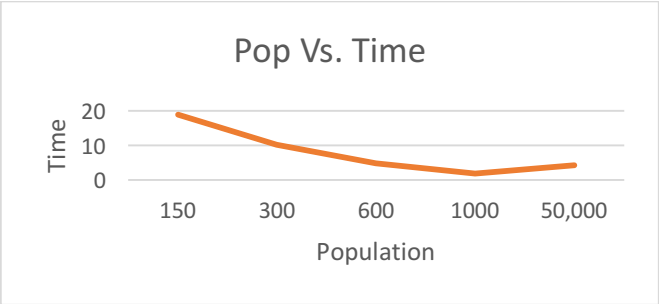



*Figure 5: Graph obtained by plotting Population v/s Time*

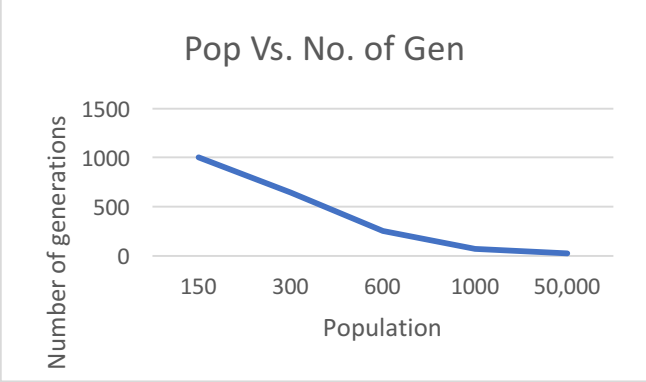



*Figure 6: Graph obtained by plotting Population v/s No. of Generation*

With varying population, we generally see a decreasing trend of time and number of generation.

That is population and performance is inversely proportional with the exception when population is 50,000.

Hence, we can conclude that there is a dependency between population and the performance.

2. When population is kept constant;

| Total Population | Mutation Rate | No of Generations until phrase solved | Total time (in sec) |
|---|---|---|---|
| 1000 | 0% | 37 | 1.2 |
| 1000 | 0.01% | 49 | 1.8 |
| 1000 | 1% | 71 | 1.6 |
| 1000 | 2% | 60 | 1.4 |
| 1000 | 10% | Very large value | Very large value |

*Table 4: Obtained Data (Case II)*

Without any mutation at all (0%), you just have to get lucky. If all the correct characters are present somewhere in some member of the initial population, you'll evolve the phrase very quickly. If not, there is no way for the

sketch to ever reach the exact phrase. Run it a few times and you'll see both instances. In addition, once the mutation rate gets high enough (10%, for example), there is so much randomness involved (1 out of every 10 letters is random in each new child) that the simulation is pretty much back to a random typing monkey. In theory, it will eventually solve the phrase, but you may be waiting much, much longer than is reasonable. With the above data, we can obtain the following graph:

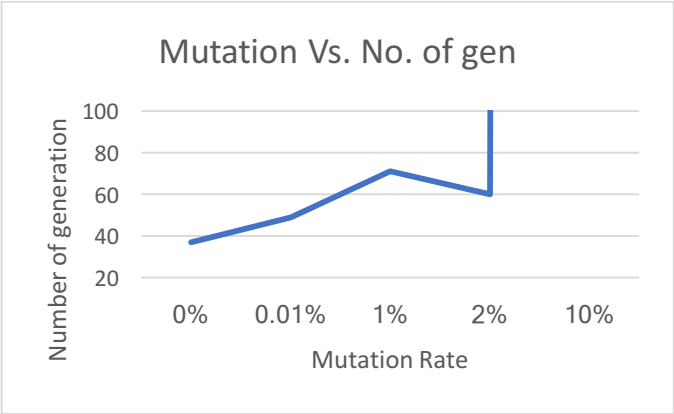



*Figure 7: Graph obtained by plotting Mutation v/s No. of Generation*

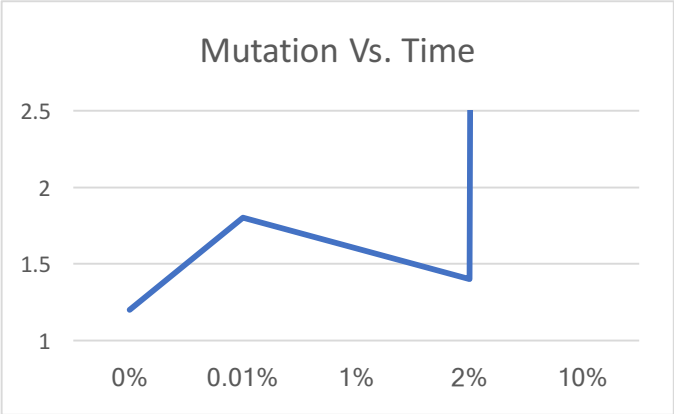



*Figure 8: Graph obtained by plotting Mutation v/s Time*

Here, we do not observe a particular trend in the mutation, however we can see that there is a random dependency of mutation with time and no. of generations. (Graph is not constant).
Hence, we can conclude that there is a dependency between mutation and performance.

## VII. CONCLUSION

After conducting this experiment, we can conclude that the GA used in the experiment is right as we have obtained our target sentence.
Another important thing we can conclude from our experimentation is that the performance of evolution is affected by factors and a variation in any of the factors would make a significant difference in the path to acquiring the target. Two of the important factors we use here are the mutation rate and the population size. In this experiment, we present the number of generations and time taken to account for the performance of the evolutionary process.

We find that there is, in general an inverse relation between the two, factor and performance.

## VIII. FUTURE WORKS

Varying the values of the mutation rate or population total is pretty simple and involves little more than typing numbers in your code. The real hard work of a developing a genetic algorithm is in writing a fitness function. If you cannot define your problem's goals and evaluate numerically how well those goals have been achieved, then you will not have successful evolution in your simulation. Improving the fitness function can improve the performance of the simulation much better. In the end, if you do not have a fitness function that effectively evaluates the performance of the individual elements of your population, you will not have any evolution. And the fitness function from one example will likely not apply to a totally different project. The same evolution idea can be improvised in many different areas like let's consider a racing simulation in which a vehicle is evolving a design optimized for speed.

***fitness = total number of frames required for vehicle to reach target***

How about a cannon that is evolving the optimal way to shoot a target?

***fitness = cannonball distance to target***

The design of computer-controlled players in a game is also a common scenario. Let's say you are programming a soccer game in which the user is the goalie. The rest of the players are controlled by your program and have a set of parameters that determine how they kick a ball towards the goal. What would the fitness score for any given player be?

***fitness = total goals scored***

## REFERENCES


[1] Fromm, H. (2005). Back to Bacteria: Richard Dawkins' Fabulous Bestiary.

[2] Maslov, I. V., & Gertner, I. (2006). *Multi-sensor fusion: an evolutionary algorithm approach*. Information Fusion, 7(3), 304-330.

[3] Fogel, D. B. (1997). *An introduction to genetic algorithms*: Melanie Mitchell. MIT Press, Cambridge MA, 1996. $30.00 (cloth), 270 pp.

[4] Giel, O., & Wegener, I. (2003, February*). Evolutionary algorithms and the maximum matching problem*. In Annual Symposium on Theoretical Aspects of Computer Science (pp. 415-426). Springer, Berlin, Heidelberg

[5] Shiffman, D. (2012). The Nature of Code: Simulating Natural Systems with Processing. Daniel Shiffman.

[6] Shiffman, D. (2012). The Nature of Code: The evolution of code. Daniel Shiffman.

[7] S. Micali and V. V. Vazirani. Analgorithm for finding maximum matching in general graphs. In *Proc. 21st Annual Symp. on Foundations of Computer Science (FOCS)*, pages 17–27, 1980.