FAKULTI TEKNOLOGI MAKLUMAT DAN KOMUNIKASI (FTMK)

BITI 3143 EVOLUTIONARY COMPUTING

PROJECT REPORT

PROJECT TITLE: DECRYPTION WORD

PREPARED BY:

NAME	MATRIC NUMBER
MOHAMAD ZAIRI BIN ABD GHANI	B031910029
JEYA JASSVINE A/P JEYABALA SUNDRAM	B031910136
FAIRUZ DIANA BINTI HAMZAH	B031910307
ANISAH BINTI KAMSIN	B031910156

PREPARED FOR:

TS. DR. ZERATUL IZZAH MOHD YUSOH

Introduction

In this project, we have understood and applied the knowledge we have learned in this course, BITI 3143 EVOLUTIONARY COMPUTING.

From our understanding, Genetic Algorithm is widely used as an adaptive technique to solve day to day routines and problems. Genetic Algorithms makes use of the operators such as selection,crossover and mutation to effectively manage the searching strategy. This algorithm is heavily based on natural selection concepts that are used in genetic studies. It is important to know why Genetic Algorithms are widely used, when it is based on a theory that was created many years ago. This is simply because this algorithm is robust. Robust being said that it can cope with incorrect or unexpected data. It also works well on continuous problems and it is based on an easy and understandable concept.

The concept behind Genetic Algorithm is easy to comprehend. First a random initial population is generated, Then we have to evaluate the fitness function and the reason why is because we only expect the fittest chromosomes to make it to the future generations. Once the fitness check has been done, the algorithm has to check if it has attained convergence. If yes, we are able to see our desired output and if not, the algorithm continues with the genetic operators function which is selection, crossing over and mutation. Once mutation is done, the fitness is evaluated again and the algorithm will check for convergence again. This cycle will repeat until the termination point when convergence is attained. The concept of the Genetic Algorithm can be summarised in the flowchart in Figure 1 as below.

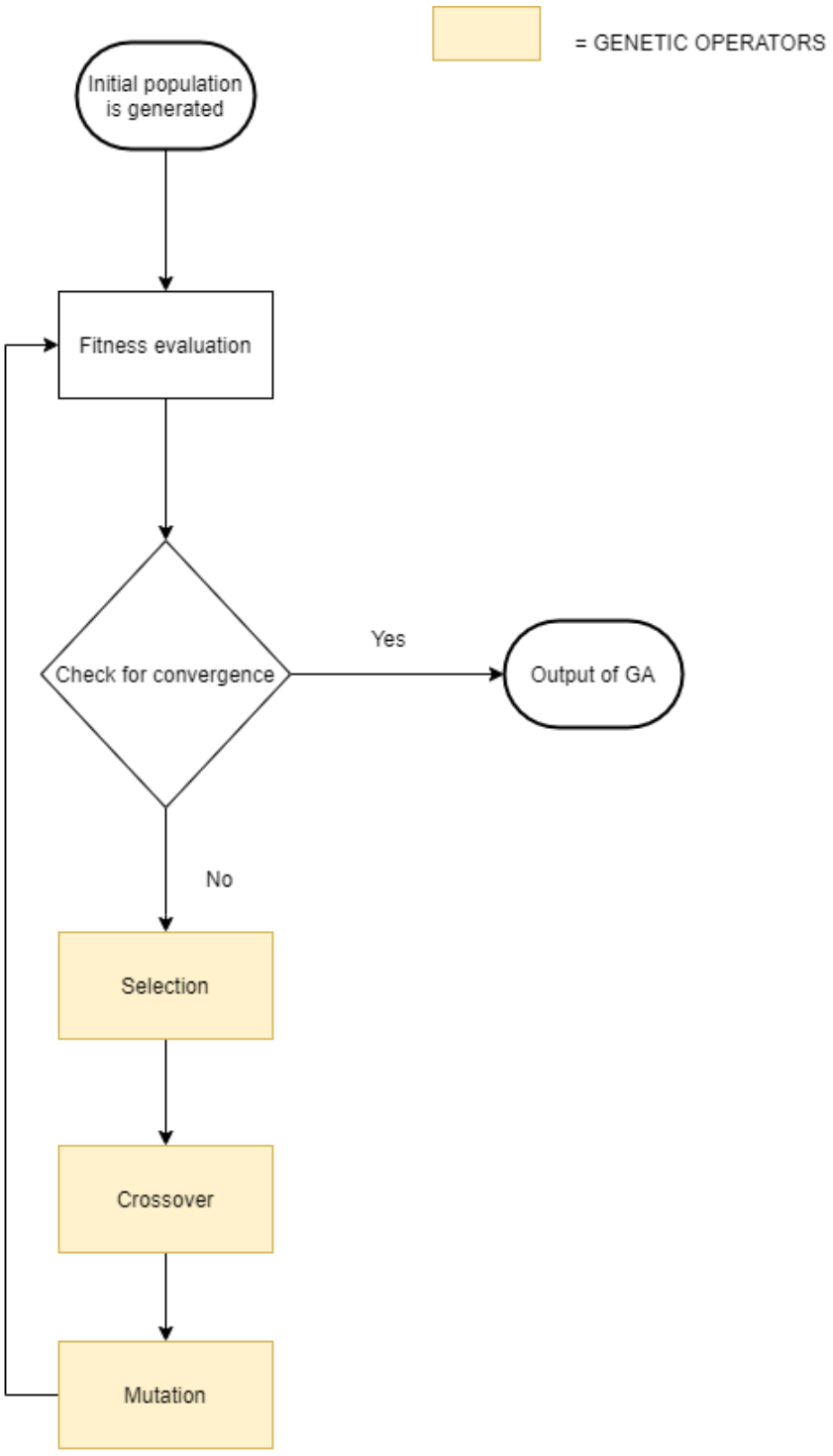

Figure 1 shows the general overview of the GA

Problem Description

This project has been developed using Genetic Algorithms to provide for a quality solution for optimization and search problems. In this case, we use the genetic algorithms to search for our target string. The genetic algorithm is a survival of the fittest. The fitness function is responsible to determine how fit the ability of the individual in the population is to compete with the other individuals.

Description of The Problem's Instances

In this project, we have set a target string which contains several letters. The target string we want to achieve is "I Love Evolutionary Computing 3000 !" from a search space, or in computational terms will be known as the population. The population ranges from uppercase letters, lowercase letters, numbers and symbols that are found in ASCII. In the case of this GA, a lower fitness score means the better it is, because it simply means the algorithm went through fewer conflicts while searching for the target string and it is more fitter.

```
// Target string to be generated
const string TARGET = "I Love Evolutionary Computing 3000 !";
```

Figure 2 shows the target string we want to achieve

The Algorithm Design

The algorithm design has to reach the target string to be considered as successful. This algorithm design is to simulate the survival of the fittest. Firstly it starts with the population that is maintained in the search space. We create the search space with each point representing a possible solution of the search place to obtain the target string "I Love Evolutionary Computing 3000 !" , so the search space we created here involves the ASCII letters.

Next, the fitness of the population is determined. Then the selection or parents from the population will occur, cross over and generate new population, mutation on new population, calculate fitness on new population and this whole cycle is repeated until

convergence. Convergence is said to have been reached when there is no significant difference in the offsprings produced in the preceding generation and the succeeding generation, this means that the algorithm has reached its converged set of problems. This also means we have attained our targeted string, hence it can terminate.

```
START
1) Initialize population
2) Calculate fitness
3) Check if converge
REPEAT
    a) Selection
    b) Crossover
    c) Mutate
    d) Calculate fitness
UNTIL convergence
STOP
```

Figure 3 shows the pseudocode for this algorithm design

Initialisation

Initialisation of the population. There has to be a population that is maintained in the search space. In this project, it is the ASCII characters. The characters on their own are considered as the gene. The characters that make up a string are considered as the chromosomes. The population will be created at random so it creates more alternatives and diversity. The reason why we need diversity is to avoid premature convergence. The population is first created randomly. Then when the iterations begin, the population generation is constituted. The characters are known as genes whereas a string of characters produced is known as the chromosomes.

```
// create initial population
for(int i = 0;i<POPULATION_SIZE;i++)
{
    string gnome = create_gnome();
    population.push_back(Individual(gnome));
}
```

Figure 3.1 shows the code for the population initialization

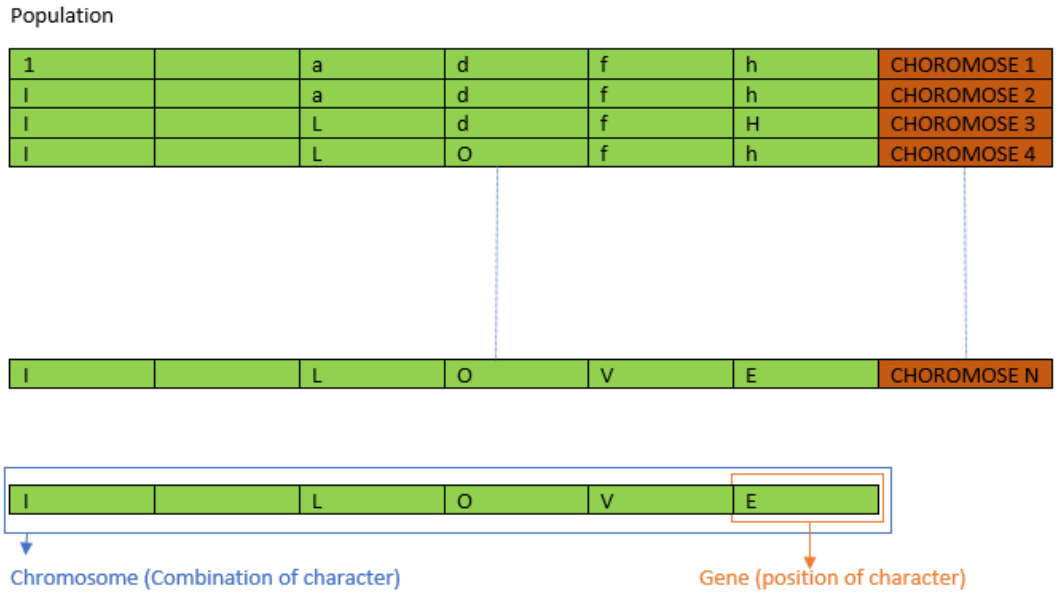

Figure 3.2 shows the chromosome representation

Fitness Function

The fitness function determines the fittest individual/character in the population. Chromosomes with better fitness scores produce better offsprings. Theoretically, in a Genetic Algorithm, the solutions are represented in the form of chromosomes. So to find the best solution, we have to test all the solutions (Chromosomes) to find the best. Each solution therefore needed to be given a value or better known as a score to indicate how close it is to being the best and this whole process is the fitness function calculation.

In our project, since we are finding a target string from a population of ASCII characters, the fitness score that is calculated is the number of

characters that are different from the characters in the target string itself. So this means that if a chromosome has a higher number of fitness scores, it has more different characters in that chromosome string compared to the target string, hence it is not the best fit. In conclusion, the lower the fitness score, the more preference is given as we want to minimise the difference between the characters in comparison to the target string.

```
Individual::Individual(string chromosome)
{
    this->chromosome = chromosome;
    fitness = cal_fitness();
};

int Individual::cal_fitness()
{
    int len = TARGET.size();
    int fitness = 0;
    for(int i = 0;i<len;i++)
    {
        if(chromosome[i] != TARGET[i])
            fitness++;
    }
    return fitness;
};
```

Figure 3.3 shows the code to calculate fitness function

Strategy of Parent Selection, Crossover, Mutation and Survival Selection

Selection strategy basically picks which are the fittest chromosomes to reproduce. Parent selection is carried out using the binary tournament selection. Two players are selected randomly to compete based on their fitness function. The winner is selected as one of the parents. Repeat the process until both parents are selected, and parent 1 is not equal to parent 2. Binary tournament selection is chosen because it is relatively simpler to compute and it gives a fairer chance of being selected as parent to chromosomes of lower fitness value. In a tournament of bigger selection size, more chromosomes of higher fitness are selected.

```
s = (90*POPULATION_SIZE)/100;
for(int i = 0;i<s;i++)
{
    int len = population.size();
    int r = random_num(0, 50);
    Individual parent1 = population[r];
    r = random_num(0, 50);
    Individual parent2 = population[r];
    Individual offspring = parent1.mate(parent2);
    new_generation.push_back(offspring);
}
```

Figure 3.4 shows the code of Parent Selection

Crossing over is the swapping of parts of the chromosome with one and another to provide more solutions for the convergence. When the crossover happens, a new offspring is produced. In this algorithm, one-point crossover is carried out. This means that one point is chosen on PARENT 1 and PARENT 2 and all the data beyond that chosen point is swapped between the two parents. The point of crossing over is chosen randomly in the range of 0 to the maximum gene length. This will result in CHILDREN 1 and CHILDREN 2 that carry mixed genetic information from their parents each.

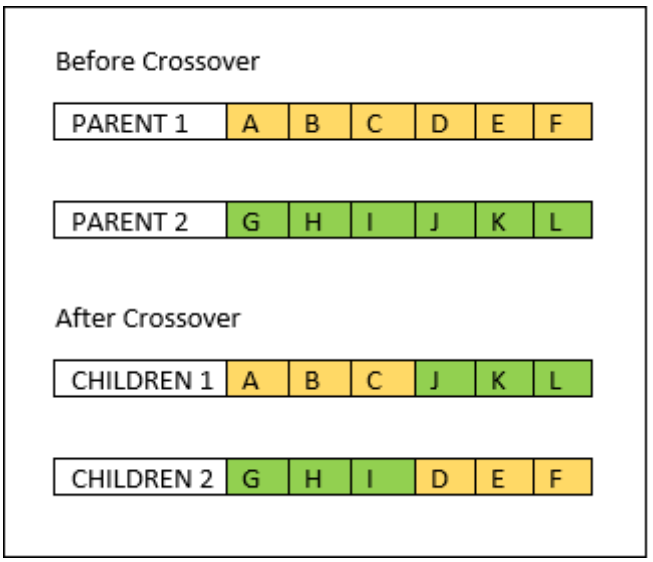

Figure 3.5 shows one-point crossover

```
Individual Individual::mate(Individual par2)
{
    // chromosome for offspring
    string child_chromosome = "";

    int len = chromosome.size();
    for(int i = 0;i<len;i++)
    {
        // random probability
        float p = random_num(0, 100)/100;



        // if prob is less than 0.45, insert gene from parent 1
        if(p < 0.45)
            child_chromosome += chromosome[i];


        // if prob is between 0.45 and 0.90, insert gene from parent 2
        else if(p < 0.90)
            child_chromosome += par2.chromosome[i];

        // otherwise insert random gene(mutate), for maintaining diversity
        else
            child_chromosome += mutated_genes();
    }

    // create new Individual(offspring) using generated chromosome for offspring
    return Individual(child_chromosome);
};

Individual::Individual(string chromosome)
```

Figure 3.6 shows the code of crossing over

Mutation is defined as tweaking the chromosome to create randomness that will cause higher probability in getting a new solution that is closer to the target solution. Mutation is essential for the convergence of Genetic Algorithms. There are several types of mutation, but based on this scenario, the insertion mutation is carried out. During the insertion mutation, a random gene gets selected and will be randomly inserted at a position. This preserves most of the order and the adjacency information. The key idea of mutation is diversity maintenance to prevent premature convergence.

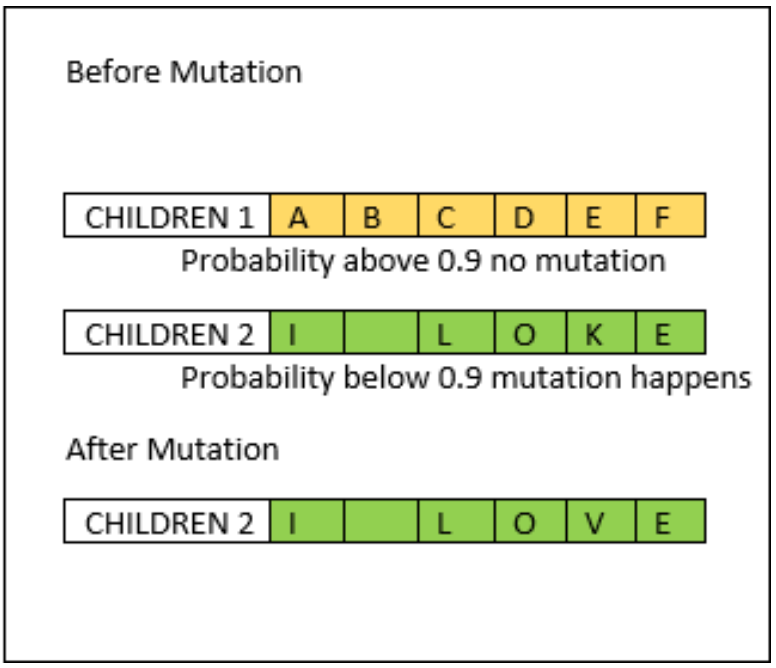

Figure 3.7 shows the code of insertion mutation

```
// Function to generate random numbers in given range
int random_num(int start, int end)
{
    int range = (end-start)+1; //space(+1)
    int random_int = start+(rand()%range);
    return random_int;
}

// Create random genes for mutation
char mutated_genes()
{
    int len = GENES.size();
    int r = random_num(0, len-1);
    return GENES[r];
}
```

Figure 3.8 shows the code of mutation

Selection is the idea to give preference to the individuals with good fitness scores and allow them to pass their genes to the successive generations. Survivor selection determines which individuals are about to be kicked out and which will be kept in the next generation. This is important as we do not want to get rid of the fit solutions as well as while maintaining the population diversity. Here we are using elitism, which means that the algorithm should always keep the copy of the fittest so far. The current fittest member of the

population will always be carried forward to the next generation. Therefore, no way that the current fittest member will be deleted, replaced or will not make it to the next generation.

```
int s = (10*POPULATION_SIZE)/100;
for(int i = 0;i<s;i++)
    new_generation.push_back(population[i]);
```

Figure 3.9 shows the code of Survivor Selection

Here we perform Elitism, that means 10% of the fittest population goes to the next generation.

Termination Strategy

The termination criteria of the algorithm are upon reaching a set maximum number of characters in string. The programme will be automatically terminated once the system gets the output string with lowest fitness.

```
// if the individual having lowest fitness score ie. 0 then
//we know that we have reached to the target and break the loop
if(population[0].fitness <= 0)
{
    found = true;
    break;
}
```

Figure 3.10 shows the code of Termination

```
String: I Love Evolut?onary Computing 3000 !   Fitness: 1
String: I Love EvolutMonary Computing 3000 !   Fitness: 1
String: I Love EvolutMonary Computing 3000 !   Fitness: 1
String: I Love EvolutMonary Computing 3000 !   Fitness: 1
String: I Love Evolutionary Computing 3000 !   Fitness: 0
```

Figure 3.11 shows the Termination of the algorithm

Measurement Indices

The measurement indices refers to the number of generations and fitness. The number of generations and fitness are the only measurement indices in this project. We have used these two measurement indices to produce target strings from a random character of string but have the same length.

Result Analysis

Population100		
Run	**Fitness Value**	**Generation**
1	34	651
2	33	539
3	33	501
4	33	786
5	33	574
	Average	**610.2**

Population 200		
Run	**Fitness Value**	**Generation**
1	32	380
2	33	462
3	33	376
4	34	409
5	33	263
	Average	**378**

Table 4.1 shows the average generation of 100 Population and 200 Population

Tables above show the results of Average of Generation for 100 and 200 Population with 5 test runs. As we can see, when we increase the population (100 to 200), the number of generations will decrease which is good for our results. The fitness value is also decreasing. So, we extend the population to 300, 400 and to 500 as shown below to see the further results.

Population 300		
Run	**Fitness Value**	**Generation**
1	33	180
2	33	272
3	33	217
4	33	219
5	32	191
	Average	**215.8**

Population 400		
Run	**Fitness Value**	**Generation**
1	33	211
2	34	210
3	33	159
4	32	207
5	32	193
	Average	**196**

Population 500		
Run	**Fitness Value**	**Generation**
1	31	174
2	32	139
3	32	146
4	33	211
5	32	207
	Average	**175.4**

Table 4.2 shows the average generation of 300 Population, 400 Population and 500 Population

Below are the visualisation of Generation by population and Fitness by Population that we gain from the tables.

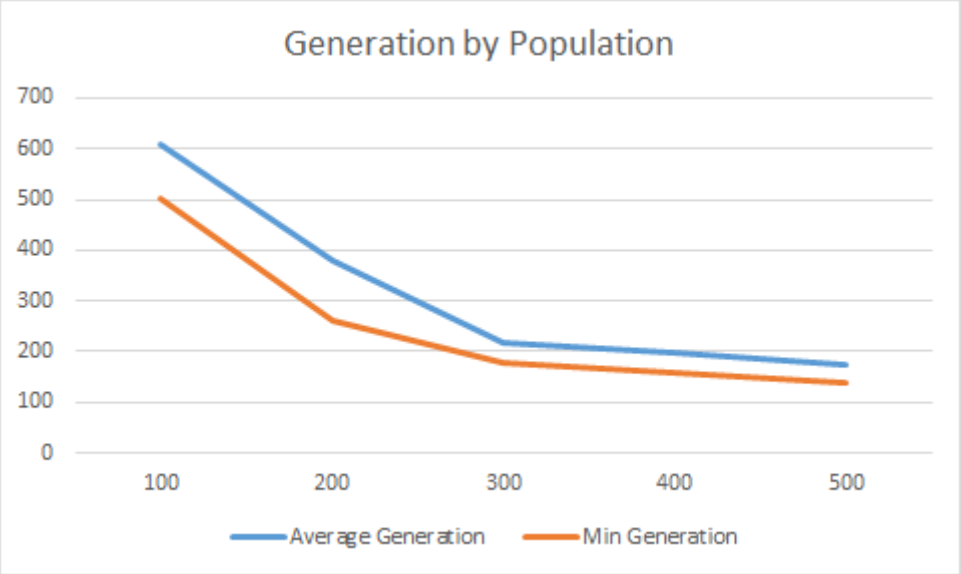

Figure 4.1 shows the Generation by Population

As we can see from 100 population to 300 population, the average generation and min generation have the drastic change, which are from 610.2 to 215.8

and 501 to 180 respectively. There is no very significant difference for the average generation between 300 population and 500 population.

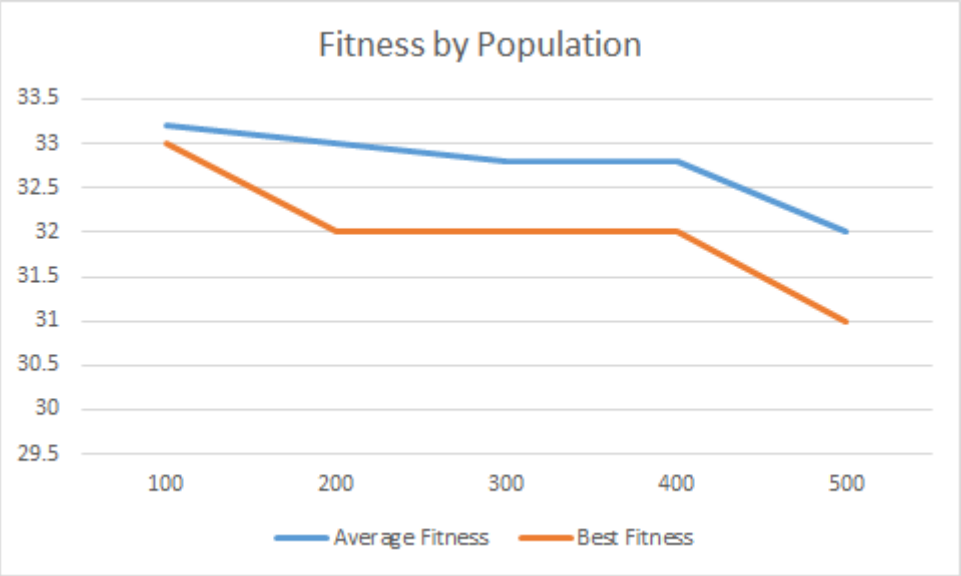

Figure 4.2 shows the Generation by Population

For the fitness graph, both average fitness and best fitness have big changes from 100 population to 500 population, which are 1.2 and 2 respectively. Even though there was no change between 300 population and 400 population for average fitness, it changed drastically from 400 population to 500 population. The best fitness value indicates the best individual in the current population while the average fitness is the mean of the entire fitness values of the population. The best fitness value can be smaller or greater than the average fitness value depending on the fitness function but they hardly conclude at the same point.

So, here we can conclude that 500 population is the best population for our project to produce target strings from a random character of string with the least number of average generation and least number of average fitness. Based on the report we generated with different parameters, it shows that the fitness value is decreasing in every population, so this shows the evolution is taking place.

References

1. *Genetic Algorithms - Survivor Selection*. Tutorialspoint. (n.d.). https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_survivor_selection.htm.
2. Ogunyale, K. (2019, December 1). *Understanding the Genetic Algorithm*. Medium. https://blog.usejournal.com/understanding-the-genetic-algorithm-4eac04a07a59.
3. Boronczyk, T. (2021, March 5). *An Introduction to Genetic Algorithms - SitePoint*. Sitepoint.com; SitePoint. https://www.sitepoint.com/genetic-algorithms-introduction/
4. Analytics Vidhya. (2017, July 31). *Genetic Algorithm | Application Of Genetic Algorithm*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2017/07/introduction-to-genetic-algorithm/
5. Brad Miller & David Goldberg, Genetic Algorithms, Tournament Selection, and the Effects of Noise, IlliGAL Report No. 95006
6. Devi, P. (n.d.). *A Symmetric Key Encryption Technique Using Genetic Algorithm*. Retrieved June 27, 2021, from http://ijcsit.com/docs/Volume%205/vol5issue01/ijcsit2014050186.pdf
7. Encryption and Decryption of a Message Involving Genetic Algorithm. (2019). *International Journal of Engineering and Advanced Technology*, *9*(2), 3920–3923. https://doi.org/10.35940/ijeat.b2379.129219
8. https://www.facebook.com/kdnuggets. (2018). *Genetic Algorithm Key Terms, Explained - KDnuggets*. KDnuggets. https://www.kdnuggets.com/2018/04/genetic-algorithm-key-terms-explained.html