COSC 3P71 Artificial Intelligence: Assignment 2 Genetic Algorithms

1st Rouvin Rebello
Department of Mathematics and Science
Brock University
St Catharine's, Canada
rr20jk@brocku.ca

***Abstract*—Genetic Algorithms (GA) are algorithms that use evolution as a search heuristic to provide good solutions to existing problems. It has its applications in crypt-analysis wherein an encrypted message can be decrypted using generated keys The keys created require various parameters that help create new solutions based on a fitness function.**

***Index Terms*—Genetic algorithm, mutation, crossover, inversion**

I. Introduction

The objective of this assignment is to implement a genetic algorithm that creates randomized keys based on a range of characters and subject it to mutation, crossovers and, elitism which helps generate new solutions. The new solutions are then compared using a fitness function which determines the best possible key to decrypt the encrypted text message.
The solutions discussed in this paper are important as they have a variety of real-world applications. For instance, cyber security, code-breaking, cryptocurrencies, cryptography studies, etc. [2]

II. Background

A. Subprograms/ Algorithms used

The main algorithms used in this paper include Tournament selection, Uniform Crossover, One-Point Crossover, Inversion Mutation and Elitism.

1) Tournament Selection: it is a concept wherein a population, k individuals are compared based on fitness, and the fittest individuals are carried forward to the next generation such that they are allowed to reproduce. This concept ensures that the right individuals are picked for a better result of the successive generation [1]. **Refer to algorithm 1**.

2) Uniform Crossover: this algorithm selects characteristics between two parents using a random 2-bit mask such that the characteristic at a point is exchanged whenever a bit is present. Crossover is determined by the rate that predicts how many parents reproduce **Refer to algorithm 2**.

3) One-Point Crossover: This algorithm selects a random substring of two parents and swaps them such that the child inherits characteristics from both parents. Crossover is determined by the rate that predicts how many parents reproduce**Refer to algorithm 3**

```
Data: Initial Population $P$, Tournament size $k$
Result: New Population $P_{\text{new}}$
for each element in $P$ do
    Randomly select $k$ chromosomes from $P$
    Let $candidate$ be the chromosome with the best
      fitness among the selected chromosomes
    if fitness(chromosome$_1$) is better than
      fitness(candidate) then
        $P_{\text{new}}$.append(chromosome$_1$)
    end
end
```

Algorithm 1: Tournament Selection

```
Data: Population $P$
Result: Offspring Population $P_{\text{offspring}}$
for every pair of parents $(Parent_1, Parent_2)$ in $P$ do
    Generate a binary mask of length equal to the
      chromosome size with random 0s and 1s
    for each gene in the chromosome do
        if mask[counter] = 1 then
            Swap genes between Parent$_1$ and Parent$_2$
        end
    end
end
```

Algorithm 2: Uniform Crossover

```
Data: Parents Parent$_1$ and Parent$_2$
Result: Offspring Child$_1$ and Child$_2$
Generate 2 random indices, $index_1$ and $index_2$
Select substring $sub_1$ from Parent$_1$ starting from
  $index_1$
Select substring $sub_2$ from Parent$_2$ starting from
  $index_2$
Child$_1$ = Parent$_1$ + $sub_2$
Child$_2$ = Parent$_2$ + $sub_1$
```

Algorithm 3: One-Point Crossover

4) Inversion Mutation: Mutation is used to introduce some degree of randomness to the solution. Similar to one point crossover, this algorithm selects two random indices which creates a substring in a child chromosome after which the substring is reversed and is placed back into the child chromosome. The mutation can be controlled using a rate that determines how many chromosomes are mutated. **Refer to algorithm 4**

```
Data: Child $child$
Result: Mutated Child
Generate 2 random indices, $index_1$ and $index_2$
Select substring $sub$ from $child$ starting from $index_1$
  to $index_2$
Reverse the characters in the substring $sub$
$child = child + sub$
```

Algorithm 4: Inversion Mutation

5) Elitism: keeps track of the best two individuals in the population and reserves a palace for them in the next generations. This is useful because as the population evolves, the good solutions are preserved and not subjected to major changes. It ensures the stability of the population while also avoiding premature convergence.

III. Experimental Setup

Parameters

- Population size: indicates the number of individuals present in the population
- Chromosomes length: length of the chromosome (randomly created string of alphabets (a..z) and hyphens (-))
- Crossover rate: probability of two parents performing a crossover (uniform or one-point) to create two children chromosomes.
- Mutation Rate: probability of child chromosome undergoing mutation (inversion) – introduces randomness into the search space.
- Number of generations: the total number of evolutions/generations that the initial population is subjected to.
- Seed: used as a parameter for random objects such that inputting a specific seed will always yield one result – good for duplication of the experiments.
- Data file: the file containing encrypted text, to be decrypted

Data Collected

- Parameters
- Average of each generation
- Best of each generation
- Best individual
- Decrypted text

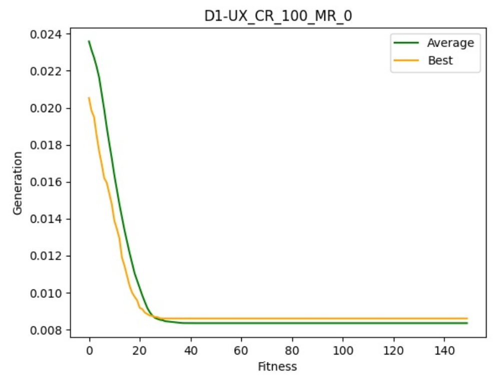


Fig. 1. Mutation = 0, Generation vs Fitness for best and average of each generation

Data

Stored in arrays - Includes changes in fitness values as a result of the different parameters. Which is then transferred to text files. Data stored in text files can be pulled into Python where graphs for average and best of a population can be calculated and visualized.

Duplication

Experiments can be duplicated in Java using the parameters provided within a saved text file. Running the Genetic_algorithm.java file will prompt the user with the parameters after which the output is printed to the terminal.

IV. Results

1. Data collected: such as the average and best fitness of each generation, is stored in text files and imported into Python, where the two metrics are visualized. For instance, below we can see the fitness value versus the number of generations for the average and best fitness of all the generations. **Refer to Fig 1**. In this output, we can see the improvement in the best fitness. the parameters used include label=–

- Data file 1
- Crossover = Uniform
- Crossover Rate = 100
- Mutation Rate = 0

Similarly, **fig 2** uses the same parameters with the mutation rate set to 10 yields a staggered curve with a better output.

2. Statistical analysis: a Mann-Whitney U test is conducted on the best individuals from each generation using the data2.txt file. The U test is a non-parametric test that aims to compare two data sets. In this case, we aim to compare the difference between two data sets (one with Uniform Crossover and the other with One-Point) thus, our null hypothesis is that there is no significant difference between the 2 cases however, our test on Python using the Scipy library [3] indicates a p value of 1.6830343488933836e-51 and a statistic of 708 which indicates that there is in fact a significant difference between the two sets. Hence, we reject the null hypothesis.

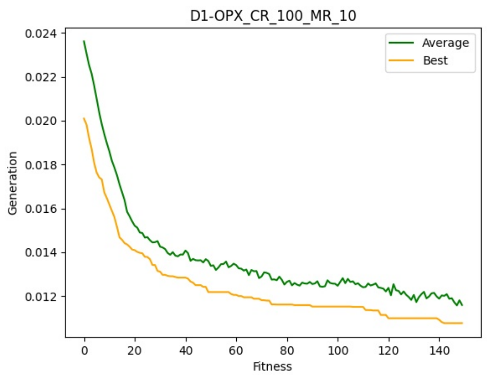
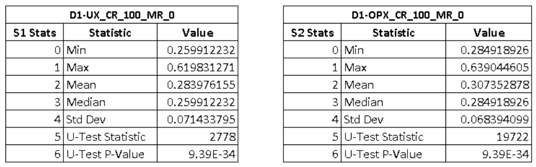


Fig. 2. Mutation = 10, Generation vs Fitness for best and average of each generation



D1-UX_CR_100_MR_0		
S1 Stats	**Statistic**	**Value**
0	Min	0.259912232
1	Max	0.619831271
2	Mean	0.283976155
3	Median	0.259912232
4	Std Dev	0.071433795
5	U-Test Statistic	2778
6	U-Test P-Value	9.39E-34

D1-OPX_CR_100_MR_0		
S2 Stats	**Statistic**	**Value**
0	Min	0.284918926
1	Max	0.639044605
2	Mean	0.307352878
3	Median	0.284918926
4	Std Dev	0.068394099
5	U-Test Statistic	19722
6	U-Test P-Value	9.39E-34

Fig. 3. Summary Tables for uniform crossover and one point crossover

3. Summary table: **Fig 3** This table stores the calculated values the average best individual of each generation using data file 1 using the parameters listed below **(Table 1)**. The summary table gives us important statistics like the minimum, maximum, mean, median, standard deviation and U test values. min and max give us an idea about the range of data we are using, the mean and median tells us if the data tends to be skewed or not, The standard deviation tells us about the diversity of the population (if standard deviation is less then it suggests that there is less diversity in the population) and finally the U test which tell us if there is a significant difference between two data sets that are somewhat identical.

Folder	**D1-UX_CR_100_MR_0**	**Folder**	**D1-OPX_CR_100_MR_0**
Name	**S1**	**Name**	**S2**
Crossover method	Uniform crossover	Crossover method	One-Point crossover
Crossover rate	100	Crossover rate	100
Mutation rate	0	Mutation rate	0

TABLE I
DATA FILE 1

V. DISCUSSIONS AND CONCLUSIONS

Introduction: This assignment mainly focuses on finding keys to decrypt text that is encrypted with a Vigenere Cipher. Keys are to be found using the implementation of a genetic algorithm. However, it is more focused on the testing aspect of the algorithm and how alteration of certain parameters can provide varying outputs.

Algorithm:

The GA consists of an initial population initializer which creates individuals that are then carried forward to the next generation using the concepts of tournament selection, crossover methods, mutation and elitism. The combination of these elements give us multiple keys the last of which is the best for the decryption.

Experimental setup: A series of experiments namely plotting of best and average generations, normality distribution and Mann-Whitney U test. For each of these tests, the results were calculated using Python files each time changing minor parameters along with the introduction of seeding which aims to allow experiments to be duplicated.

Results: For each parameter change, we can see varying results. Upon experimentation, the parameters Crossover rate = 90 and mutation rate = 15 using uniform crossover seemed to provide the best outputs. The graphical representations show how the data sets converge to give us good solutions. However, the main observation to take into consideration is the impact of different crossover and mutation rates applied to the population.

1) **Effect of Crossover:**
    a) Upon comparing the uniform order crossover with the one-point crossover, we can see how the different swaps of genetic material affect the population. In most cases, the UX tended to perform better than the one-point crossover.
2) **Influence of Mutation Rates:**
    a) The experiments show us that when the chromosomes are set to reproduce without a mutation rate, the graphs tend to be much smoother than when a mutation rate is applied. This suggests that mutation rates often introduce a minimum amount of randomness therefore resulting in better solutions.

Conclusion: In essence, the application of genetic algorithms in cryptanalysis are important as they provide different approaches to a problem. The adaptability of genetic algorithms is important as it allows the user to make breakthroughs in problems that often result in tunnel vision. Hence it is a great application to many real-world situations.

REFERENCES

[1] Berman, B. O. (n.d.). Genetic Algorithm Slides. Brightspace - COSC 3P71. https://brightspace.brocku.ca/d2l/le/lessons/77495/topics/2358526
[2] Chatgpt. ChatGPT. (n.d.). https://openai.com/chatgpt
[3] W3schools. (n.d.). Introduction to SciPy library in python. Scipy tutorial. https://www.w3schools.com/python/scipy/index.php