COSC 3P71- Assignment-2

David Martin
Brock University
St.Catherines, Canada
dm20zo@brocku.ca 6995948

***Abstract*—The Report explores the implementation of a Genetic Algorithm for decrypting text using randomly generated keys. Additionally, each component of a Genetic Algorithm will be explored like tournament selection, Uniform and Two Point Crossover,mutations and elitism. The report will also shed light on how changing parameters like mutation rate and Crossover rate could ultimately impact the performance of the Genetic Algorithm. Additionally, statistical tests will be performed on the collected data to compare the performance of the two crossover methods.**

I. Introduction

Cryptography plays a significant role when it comes to securing information and encrypting text with a cipher. The aim of this report is to implement a Genetic Algorithm to decrypt text with generated keys from implementing a Genetic Algorithm. In the context of this assignment, a potential solution is a generated key which is used to attempt to decrypt the text in the file. In the context of Genetic Algorithms, each randomly generated key is a chromosome representation. As the algorithm performs the steps of the Genetic algorithm, the generated keys will slowly start to reassemble English and as does the encrypted message in the text file. The importance of the Genetic Algorithm for this task is vital as each generation derives better fitted keys to decrypt the text to reassemble English. This report aims to take what was learned in class in the context of Genetic Algorithms, and apply them to this problem. This report will also explain each procedure of Genetic Algorithms such as, randomly generating a population, tournament selection, crossovers and mutations each were implemented in the context of the cryptography problem given in the assignment.

II. Background

A. Components of a Genetic Algorithm

To explain how the Genetic Algorithm was implemented for this problem, it is vital to explain each step of a Genetic Algorithm to understand how it was applied to this specific problem.

B. Generating A population

The first process involved in a Genetic Algorithm is to randomly generate a population while guessing the size empirically. Each chromosome in will represent a possible candidate solution. In the case of this problem, the chromosomes are strings of randomly generated characters from A-Z and including the character ”-.” The population size in this case is estimated to produce the most effective results. In the case of this problem, a populations size of 200 was provided. Meaning that there were 200 randomly generated strings of random characters each acting as a candidate solution. The generation of the initial population can be represented as pseudo code:

Algorithm 1 Generate Population

```
for $i = 1$ to Population Size do
    Select a subset of size $k$  Find the chromosome with the best fitness in the subset  Add the chromosome with the best fitness to the parents array
```

Then after the population is randomly generated, it is vital to evaluate the fitness level for each chromosome in the initial population. The fitness level corresponds to how suitable the chromosome is for providing a potential solution. In the case of this problem, the fitness level would correspond to how close the randomly generated string which is acting as a key. The fitness level in this case is how well the key is able to decrypt the text such that the text slowly starts to represent English. Initially, the fitness level will be high meaning that non of the keys perform all that well acting as key.

C. Tournament Selection

Another Vital process of a Genetic Algorithm, is known as ”Tournament Selection.” Tournament Selection essentially randomly selects k chromosomes from the population.K could be anywhere between the number 2 and 5. Out of k selected chromosomes, the one with the best fitness level is best suitable to act as a parent to the next generation.Thus, being preserved to be a parent. This process continues until the number of parents selected from each tournament matches the initial population size. The chromosomes with a bad fitness level will not be suitable for re population. The number of parents selected from the tournament selection will match the population size. Tournament selection can represented with pseudo code below:

D. Crossover

Crossover methods are a mandatory process in creating Genetic Algorithms. Crossovers in Genetic Algorithms are

Algorithm 2 Tournament Selection

```
Data: Population, Tournament Size $t$
Result: Parents array
for $i = 1$ to Population Size do
    Select $t$ individuals randomly from the population
    Find the individual with the best fitness in the tournament
    Add the selected individual to the parents array
```

essentially combining two chromosomes that are suitable for reproduction that were chosen via the previous function tournament selection. In the case of this problem, they keys with the highest fitness level will be able to crossover with other keys with high fitness levels to bring to the next generation. The way these chromosomes ”reproduce” is with each other is with crossovers. The two crossover methods implemented were uniform crossover and two point crossover. For the purpose of understanding, both crossover implementation methods will be explained.

- Uniform Crossover
  With Uniform Crossover, two parents are selected from the new population. Then, a mask is generated containing a series of randomly generated 0s and 1s. Following this, there is a probability generated to determine if the crossover will occur or not. If the crossover does not occur, then the two parents selected for the crossover will simply be within the next generation. However, if the crossover does occur, then we create the two offspring with genes from the parents. This happens according to the mask such that if there is a 1 in the mask with the corresponding gene, then that genetic trait will remain in the first child. The also holds true for the second parent and the second child. However, if a 0 is present at that corresponding index, that child one inherits that trait from the second parent. The same rule is applied. The following algorithm will be presented in pseudo code to provide a better understanding of this crossover method:

Algorithm 3 Uniform Crossover

```
Data: Parent 1 ($P1$), Parent 2 ($P2$), Crossover Rate
Result: Child
for gene in range($P1$.length) do
    $rand\_value \leftarrow$ random(0, 1)
    if $rand\_value <$ Crossover Rate then
        $child$.gene[gene] $\leftarrow$ (randomInt(0, 1) == 1) ? $P1$.gene[gene] : $P2$.gene[gene]
    else
        $child$.gene[gene] $\leftarrow$ $P1$.gene[gene]
```

- Two Point Crossover:
  With Two Point Crossover, the process is the same as any other crossover method initially. Choosing two parents from the population to produce offspring. With Two Point Crossover, two crossover points are randomly selected. Essentially, between those two crossover points, child one would inherent genes from parent two and child two would inherent genes from parent one.Outside of that range between the two crossover points, Child one would inherent from Parent One and Child Two would inherent from parent two. The following algorithm will be presented in pseudo code below:

Algorithm 4 Two-Point Crossover

```
Data: Crossover Rate, Chromosome Array arr
Result: Children Chromosome Array children
chromosome[] children $\leftarrow$ new chromosome[POPULATION_SIZE]
for $i \leftarrow 0$ to arr.length $- 1$ 2 do
    chromosome parentOne $\leftarrow$ arr[i]
    chromosome parentTwo $\leftarrow$ arr[$i + 1$]
    int pointOne $\leftarrow$ rand.nextInt(CHROMOSOME_LENGTH)
    int pointTwo $\leftarrow$ rand.nextInt(CHROMOSOME_LENGTH)
    if pointOne ¿ pointTwo then
        pointOne, pointTwo $\leftarrow$ pointTwo, pointOne
    char[] c1 $\leftarrow$ parentOne.getChromosome().clone()
    char[] c2 $\leftarrow$ parentTwo.getChromosome().clone()
    for j $\leftarrow$ pointOne to pointTwo do
        c1[j], c2[j] $\leftarrow$ c2[j], c1[j]
    chromosome childOne $\leftarrow$ new chromosome(c1)
    chromosome childTwo $\leftarrow$ new chromosome(c2)
    children[i] $\leftarrow$ childOne
    children[$i + 1$] $\leftarrow$ childTwo
return children
```

Crossovers for the case of this problem are vital for finding better encryption keys with better fitness values by inheriting characters from their parents. Implementing crossovers allow for stronger potential solutions.

E. Mutation

Another component to genetic Algorithms are procedures known as mutations. Mutations essentially re write components within a gene. In the case of this specific problem with encryption keys , mutations would alter characters within the string. Mutations ultimately provide more genetic variation when performing a genetic algorithm. While mutations can improve the fitness while running the GA, it is possible that mutations could also negatively effect. The mutation operator that was utilized for this problem is known as the ”inversion” operator. Essentially, the inversion mutation works by picking two points within in a chromosome and inverting each gene within those two points. In the context of this assignment, the two points would act as indexes for the character array. Then, each character between these two indexes would become inverted.Thus, adding more genetic variety to the population. The pseudo Code for inversion will be provided below:

Algorithm 5 Mutation Operation

```
Data: Mutation Rate, Random Number Generator ($rand$)
Result: Mutated Chromosome
$rand\_val \leftarrow$ rand.nextDouble()
if $rand\_val \leq$ Mutation Rate then
    $start\_index \leftarrow$ rand.nextInt(chromosome_length)
    $end\_index \leftarrow$ rand.nextInt(chromosome_length)
    if $start\_index > end\_index$ then
        $temp \leftarrow start\_index$
        $start\_index \leftarrow end\_index$
        $end\_index \leftarrow temp$
    while $start\_index < end\_index$ do
        $temp \leftarrow$ chromosome[$start\_index$]
        chromosome[$start\_index$] $\leftarrow$ chromosome[$end\_index$]
        chromosome[$end\_index$] $\leftarrow$ temp
        $start\_index \leftarrow start\_index + 1$
        $end\_index \leftarrow end\_index - 1$
return Mutated Chromosome
```

F. Eliteism

Elitism was also a prevalent part of this Genetic Algorithm. Elitism essentially preserves the chromosomes with the highest fitness level and ensures that those chromosomes are participating in the next generation. In doing this, it ensures that the Genetic Algorithm produces better results by preserving candidate solutions that perform well for each generation.

G. Evaluate/Continue Process

After the next generation has been produces with crossovers and mutations, the algorithm will evaluate the fitness of the new generation. This entire process will continue until the maximum number of generations have been produced. The entire genetic algorithm will be written in pseudo code below:

Algorithm 6 Genetic Algorithm Main Function

```
Data: Crossover Rate, Mutation Rate, Tournament Size ($k$), Seed
Initialize Number generation with seeding
for gen = 1 to MAXGEN do
    Evaluate fitness of each individual in the population
    Perform Tournament Selection with size $k$
    Perform Uniform Crossover with mutation rate
    Mutate with mutation rate
    Evaluate fitness of the new population
```

III. Experimental Setup

Before running the experiments, there were a number of parameters that were required to be tested. These parameters included: mutation rate, Crossover rate,the value of k for tournament selection as well as the Crossover method that was being used during that specific execution of the Genetic Algorithm. The following parameters that were applied to the main Genetic Algorithm function were the following:

- A: Crossover rate = 100%, Mutation Rate = 0%
- B: Crossover rate = 100%, Mutation Rate = 10%
- C: Crossover rate = 90%, Mutation Rate = 0%
- D: Crossover rate = 90%, Mutation Rate = 10%
- E: Implementing our own set of Parameters

Throughout each of the experiments, a population size of 200 chromosomes were used. Additionally, each of this programs were run with 5 different number seeds to ensure that the random numbers generated in the program would continue to be produced. Thus, there was each of the 5 parameters being run with 5 different number seeds assuming it was implemented correctly. Additionally, each of the parameters above were tested with both Uniform Crossover as well as Two point Crossover. All of these runs were tested on both Data1.txt and Data2.txt. Both of which were provided.

IV. Results

The experiments were plotted in such a way that it plots the average of the averages and the average of the best chromosomes amongst each of five seeds for each of the five different parameters. For each of the plots, A through E will be labeled in the legend in each graph to display that run A-D represent what corresponding parameters were the genetic algorithm was run with.

- RUN A: Crossover rate = 100%, Mutation Rate = 0%
- RUN B: Crossover rate = 100%, Mutation Rate = 10%
- RUN C: Crossover rate = 90%, Mutation Rate = 0%
- RUN D: Crossover rate = 90%, Mutation Rate = 10%
- RUN E: Crossover rate = 100%, Mutation Rate = 20%

UNIFORM CROSSOVER FOR DATA 1

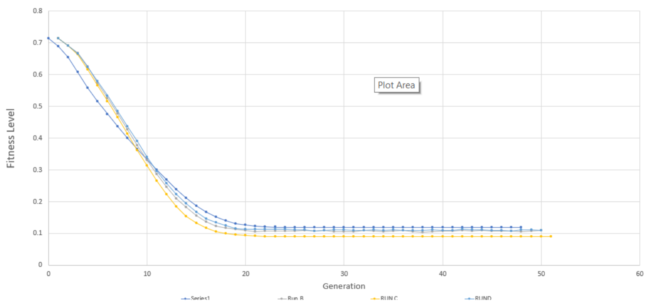


Fig. 1. Uniform Cross Over Average Data 1

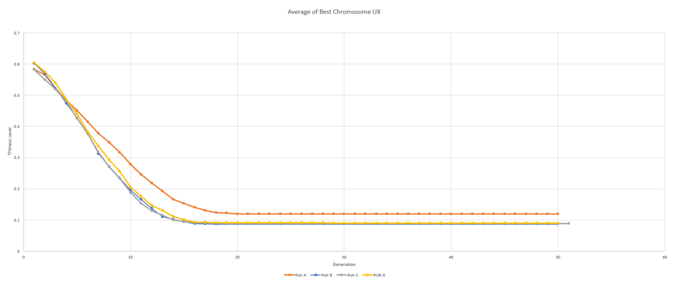


Fig. 2. Uniform Cross Over Best Chromosome Average

- Above displays the results of the Genetic Algorithm for the average of the averages and the average best chromosomes for each generation amongst all five seeds with all the specified parameters A through D using Data 1 over 50 generations. Looking at the plots, each run for the best and the average around 0.1. Either slightly below or slightly above 0.1. Looking closely at the graph, RUN C seems to come to the best results. Run C had a crossover rate of % 90 and a mutation rate of % 10. Otherwise all other runs seem to converge in the 0.10 range of fitness by the 20 generation mark.

Two Point Crossover FOR DATA 1

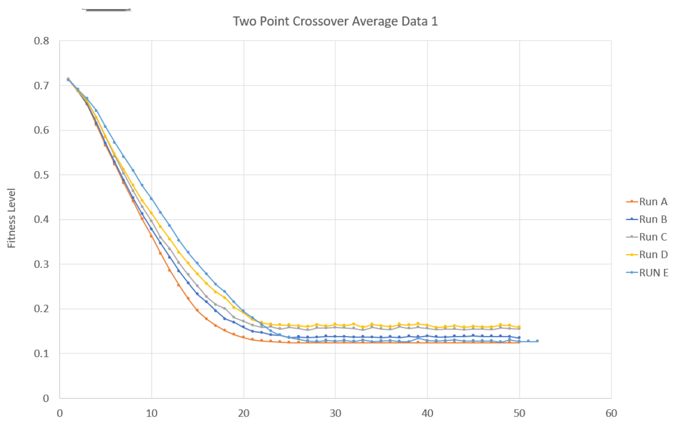


Fig. 3. Two Point Crossover Average for Data 1

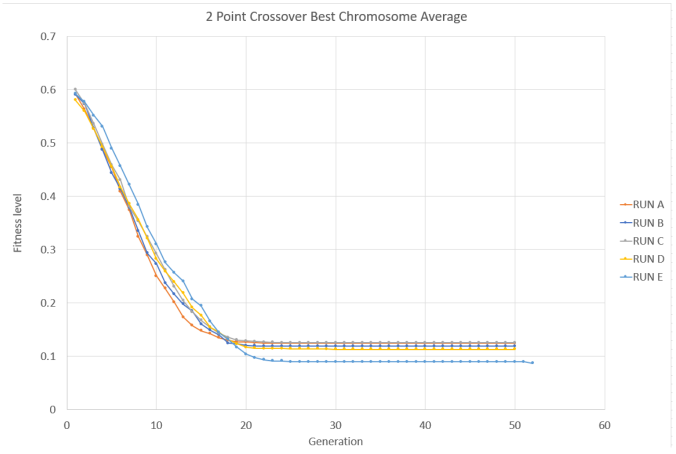


Fig. 4. Two Point Crossover Best Chromosome Average

- Above displays the results of the Genetic Algorithm for the average of averages and the average for the best chromosomes for each generation amongst all five seeds and each of the specified parameters A through E using Data 1 over 50 generations. When looking at the plot, it seems that the average for Run A has the best average fitness level. Different from uniform crossover, Two Point Crossover seems to converge after more generations. However, both converge in the 0.10 range of fitness as no more genetic variation is provided for the genetic Algorithm. Though, looking at the Average for the Best Chromosomes, run E seems to have the best results with a chromosome best average of below 0.10. Run E had the highest mutation rate. Thus, it is possible that a higher mutation rate has allowed for more genetic information that performed better for this specific run.

Uniform Crossover for DATA 2

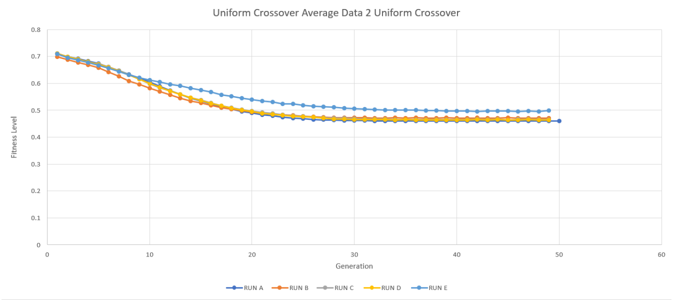


Fig. 5. Uniform Cross Over Average DATA 2

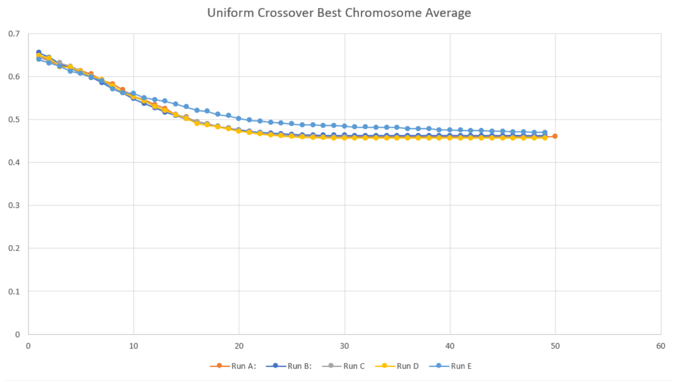


Fig. 6. Uniform Cross Over Best Chromosome Average for DATA 2

- Above displays the results of the Genetic Algorithm for the the average of averages and the average for the best chromosomes for each generation amongst all five seeds and each of the specified parameters A through E using Data 2 over 50 generations. When looking at the plot,it seems that it converges around the 0.50 fitness level. Run E parameters seemed to have produced the best results with the fitness level being the lowest of each of the other runs. Run E had a mutation rate of % 20.

Two Point Crossover for DATA 2

- Above displays the results of the Genetic Algorithm for the the average of averages and the average for the best chromosomes for each generation amongst all five seeds and each of the specified parameters A through

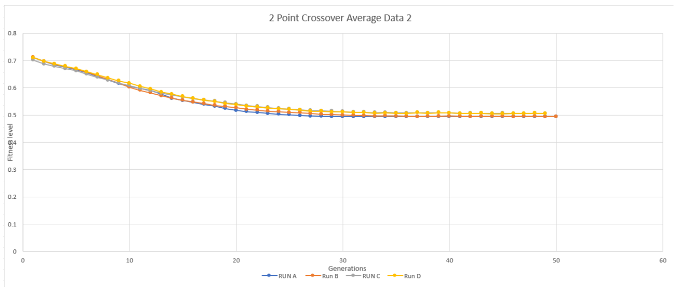


Fig. 7. Two Point Crossover Chromosome Average for DATA 2

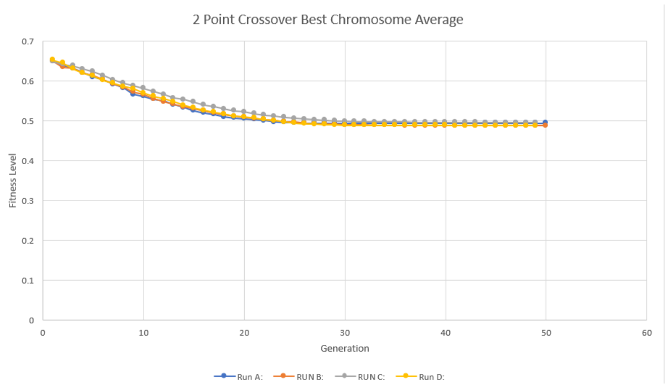
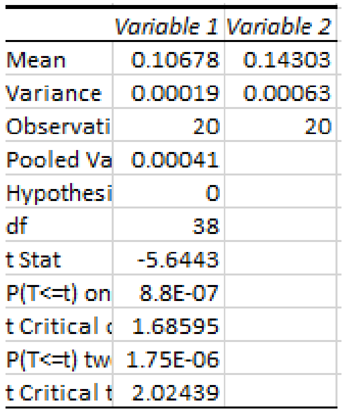


Fig. 8. Two Point Crossover Best Chromosome Average for DATA 2

E using Data 2 over 50 generations using the Two Point Crossover method. Just by comparing the two plots between Uniform Crossover and Two point crossover for data set 2, it seems that Uniform Crossover gets better results for all Runs compared to Two Point Crossover. The two point Crossover for all runs seems to roughly converging around the 0.50 fitness level mark and never going below 0.50 compared to uniform crossover which would perform into the high 0.40s fitness level.

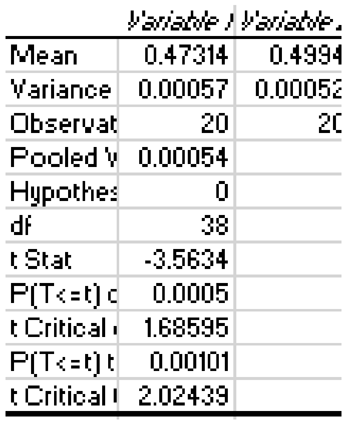
T-TEST/ Hypothesis

- **Null Hypothesis**: When comparing the population averages between uniform Crossover and Two point Crossover for data set 1, A T-Test will be performed to show that there is no significant difference between averages produced from the two different crossover methods. **NOTE:** The tables are in figure 9 and figure 10



	Variable 1	Variable 2
Mean	0.10678	0.14303
Variance	0.00019	0.00063
Observati	20	20
Pooled Va	0.00041	
Hypothesi	0	
df	38	
t Stat	-5.6443	
P(T<=t) on	8.8E-07	
t Critical c	1.68595	
P(T<=t) tw	1.75E-06	
t Critical t	2.02439	

Fig. 9. T test for data One P-value: 1.75E-06



	Variable 1	Variable
Mean	0.47314	0.4994
Variance	0.00057	0.00052
Observat	20	20
Pooled V	0.00054	
Hypothes	0	
df	38	
t Stat	-3.5634	
P(T<=t) c	0.0005	
t Critical	1.68595	
P(T<=t) t	0.00101	
t Critical	2.02439	

Fig. 10. T test for data Two P-value: 0.001006488

- Thus, looking at the P-values for both sets of data, it is sufficient to reject the null hypothesis and come to the conclusion there is a significant statistical difference between the uniform crossover population average and two point crossover. This is due to the fact that both P-Values are less than 0.05.

V. Conclusion

Summarizing this report, the following experiments were conducted with the following conditions:

A. Setting up the Experiments

- Created a population size of 200 chromosomes
- Both uniform and Two Point Crossover were employed for the experiments
- The entirety of the Genetic Algorithm was tested on both Data1.txt and Data2.txt
- Each with five different number seeds.
- Multiple Runs with varying percentages of mutation rates and Crossover Rates
- Inversion Mutation was the operator used for each experiment.

B. Findings

Following the execution of each of these experiments, the following results can be observed about the Genetic Algorithm that was implemented:

- When conducting the T-Test, there was evidence to suggest that Uniform Crossover performed better than the Two Point Crossover.
- The genetic Algorithm was converging early on Data 2 compared to Data 1.

- Using a Higher mutation rate provides more genetic variety to the population that could potentially improve the average fitness of each generation.
- Each Run with different parameters produced different results.

C. Opinion/ Summary

In conclusion, Data 1 performed significantly better than Data 2. Also, Uniform Crossover converged on a better fitness level than Two Point Crossover. Another point worth considering is that eliteism greatly improved the performance of the Genetic Algorithm and not just the crossover methods. Eliteism is important to the Genetic Algorithm to enhance performance and converge on a better solution.

References

[1] Dr. Ombuki-Berman, B 2023."GA_Assignment_slides" [Powerpoint], COSC 3P71: Introduction to Artificial Intelligence. Brock University. 10, November.