COSC 3P71 GA Assignment 1 Report

Name: Jay Shah Student - ID: 77244974 - Email: fn22gq@brocku.ca

I. Introduction

Genetic Algorithms(GA) are a type of stochastic optimization algorithm inspired by the process of evolution.

As described in [1], Cryptanalysis is an active and challenging area of research. The security of cryptographical systems is extremely important, given how much of our lives take place online. Most people are comfortable paying online by credit card, even though this information passes through tens or even hundreds of machines owned by complete strangers. This confidence largely stems from the widespread use of cryptographic systems, which we trust to keep our private information secure. To ensure that these systems remain robust and effective, researchers continually test and investigate potential vulnerabilities. One such approach involves using evolutionary algorithms, such as genetic algorithms, to break encryption or uncover weaknesses. These algorithms simulate natural selection to search for solutions efficiently.

In this report, we apply genetic algorithms to find the passwords used to encrypt text with a simple cryptographic system: the Vigenère cipher. This report explores the use of evolutionary algorithms in cryptanalysis.

II. Background

Genetic Algorithms simulate evolution as a search heuristic to find optimal or near-optimal solutions.

As mentioned in [2], According to Universal Darwinism, three core mechanisms are necessary for evolution:

- Selection of fittest individuals
- Reproduction with inheritance
- Variation in the form of Mutations and Crossovers.

Algorithm 1 from [3] outlines the basic structure of a genetic algorithm. It begins with a randomly initialized population of a predefined size. The fitness of each individual is evaluated, and based on this, a selection process is used to choose parents for reproduction. Mutation and crossover are applied to produce offspring ($c1$, $c2$) from parents ($p1$, $p2$). This process repeats until the maximum number of generations is reached or the desired fitness is achieved. Elitism is also employed by preserving the top 10% of the population in each generation.

A. Chromosome Structure

Each chromosome consists of a key (the candidate solution) and a fitness score. The key is a character array of a fixed maximum length which is interpreted as a String. Characters are selected from the lowercase alphabet ('a' to 'z') and the hyphen ('-'). Hyphens are treated as null characters and are not included in the effective key length. This allows the algorithm to explore variable-length solutions within a fixed-size representation.

Algorithm 1 Genetic Algorithm

Require: Key length L, Cipher text C, Population size N, Max generations G, Target fitness F

Initialize $population \leftarrow$ CreateRandomPopulation(N, L)
for $generation = 1$ to G **do**
  $fitnessScores \leftarrow$ evaluateFitness($population$)
  if $\min(fitnessScores) < F$ **then**
    break
  end if
  $newPopulation \leftarrow$ SelectElite($population, fitnessScores$)
  while $|newPopulation| < N$ **do**
    $p1, p2 \leftarrow$ TournamentSelection($population, fitnessScores$)
    $c1, c2 \leftarrow$ Crossover($p1, p2$)
    $c1 \leftarrow$ Mutate($child1$)
    $c2 \leftarrow$ Mutate($child2$)
    Add $c1$ and $c2$ to $newPopulation$
  end while
  $population \leftarrow newPopulation$
end for
$bestIndividual \leftarrow$ Individual in $population$ with lowest fitness
$bestFitness \leftarrow$ Fitness($bestIndividual, C$)
return $(bestIndividual, bestFitness)$

B. Evaluation

The fitness function provided as part of [1] will accept some encrypted text and a key. It returns a real number f, which indicating how much the decrypted result looks like English. f will be greater than or equal to 0 with smaller numbers indicating a better solution.

C. Selection Method

Fitter individuals reproduce more, while some individuals go extinct as described in [2]. To replicate this, the selection process is made stochastic. Tournament Selection, as represented by Algorithm 2, selects a random sample of k individuals. From this sample, we select the fittest individual. We keep selecting until we have a new population of required size.

D. Crossover and Mutation

The crossovers and mutation are performed between the chromosomes stochastically depending on the crossover and mutation rate respectively.

Algorithm 2 Tournament Selection

Require: Tournament size k, Population P
Initialize $newPopulation \leftarrow \emptyset$
while $|newPopulation| \neq$ requiredPopulationSize **do**
  Initialize $best \leftarrow$ dummy chromosome with worst fitness
  for $i = 1$ to k **do**
    $index \leftarrow$ random integer from 0 to $|P| - 1$
    $candidate \leftarrow P[index]$
    if $candidate.fitness < best.fitness$ **then**
      $best \leftarrow candidate$
    end if
  end for
  Add $best$ to $newPopulation$
end while
return $newPopulation$

Two types of crossover are used:

- **Uniform Crossover** (Algorithm 3) – swaps genes based on a randomly generated mask.
- **Two-Point Crossover** (Algorithm 4) – swaps gene segments between two random points.

Two mutation strategies are employed:

- **Inverse Mutation** (Algorithm 5) – reverses the sequence of a random substring.
- **ASCII Mutation** (Algorithm 6) – alters a character's ASCII value by ±2.

Algorithm 3 Uniform Crossover

Require: Parents $p1$, $p2$, Crossover rate r, Chromosome length L
$c1, c2 \leftarrow$ Copy($p1$), Copy($p2$)
if random() $< r$ **then**
  for $i = 1$ to L **do**
    if random() < 0.5 **then**
      Swap $c1[i]$ and $c2[i]$
    end if
  end for
end if
return $(c1, c2)$

Algorithm 6 ASCII Mutation

Require: Chromosome c, Mutation rate r, Chromosome length L
$c' \leftarrow$ Copy(c)
if random() $< r$ **then**
  repeat
    $i \leftarrow$ random integer in $[0, L-1]$
  until $c'[i] \neq$ –
  $c'[i] \leftarrow c'[i] \pm 2$
end if
return c'

Algorithm 4 Two-Point Crossover

Require: Parents $p1$, $p2$, Crossover rate r, Chromosome length L
$c1, c2 \leftarrow$ Copy($p1$), Copy($p2$)
if random() ¡ r **then**
  $start \leftarrow$ random integer in $[0, L)$
  $end \leftarrow$ random integer in $[start, L)$
  for $x = start$ to $end - 1$ **do**
    $c1, c2 \leftarrow c2, c1$
  end for
end if
return $(c1, c2)$

Algorithm 5 Inverse Mutation

Require: Chromosome c, Mutation rate r, Chromosome length L
$c' \leftarrow$ Copy(c)
if random() $> r$ **then**
  return c'
end if
$start \leftarrow$ random integer in $[0, L-1]$
$end \leftarrow$ random integer in $[start, L-1]$
Invert substring of c' from $start$ to end
return c'

III. Experimental Setup

We conduct 8 core experiments, each combining:

- 2 encrypted datasets (data1.txt and data2.txt)
- 2 mutation methods
- 2 crossover methods

Each configuration is tested with Six combinations of crossover and mutation rates, and each setting is repeated 5 times with different random seeds. Key length bounds are 26 for data1.txt and 40 for data2.txt.

Averages of Fitness values among the 5 runs are recorded every 10 generations for analysis.

A. Parameters

- Population size: 500
- Max generations: 100
- Tournament size: 2
- Target fitness: 0.01

Tested crossover rates: {0.9, 1.0} Tested mutation rates: {0.0, 0.1, 0.5}

IV. Results

A. Graph Analysis

The graphs below showcase the average fitness over the 5 runs obtained at every 10 generations. Overall, we can observe that configurations with no mutation give poor results.

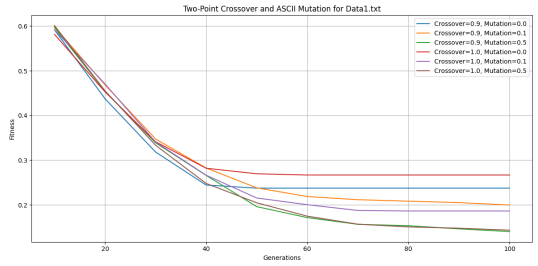

Fig. 1. Two-Point Crossover and ASCII Mutation for Data1.txt

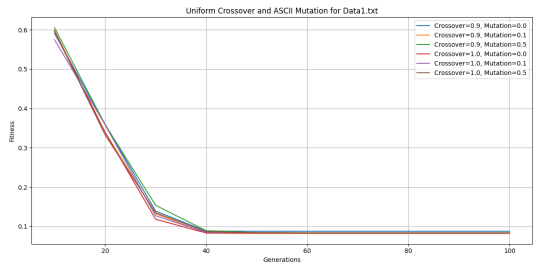

Fig. 2. Uniform Crossover and ASCII Mutation for Data1.txt

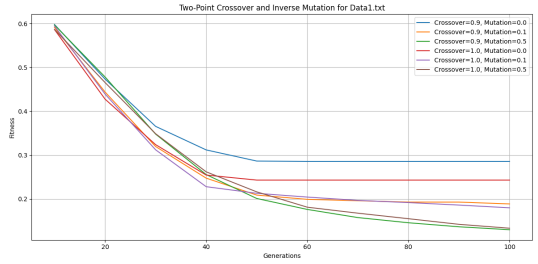

Fig. 3. Two-Point Crossover and Inverse Mutation for Data1.txt

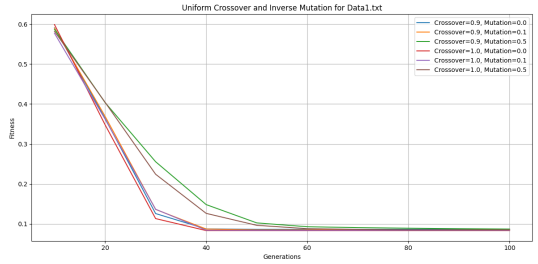

Fig. 4. Uniform Crossover and Inverse Mutation for Data1.txt

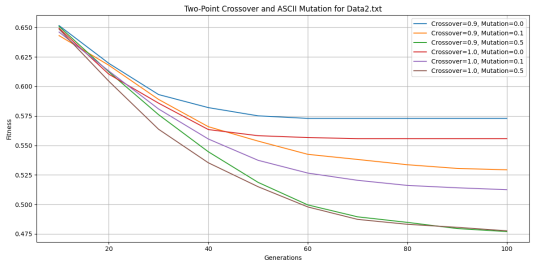

Fig. 5. Two-Point Crossover and ASCII Mutation for Data2.txt

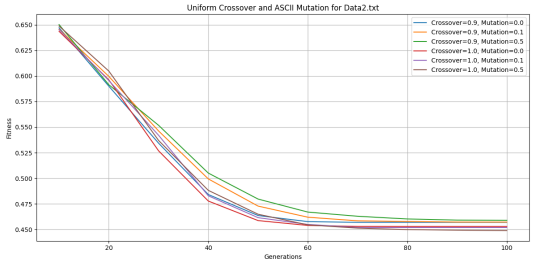

Fig. 6. Uniform Crossover and ASCII Mutation for Data2.txt

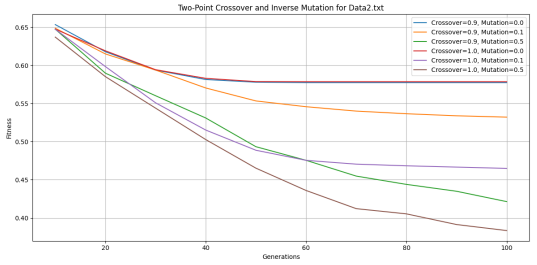

Fig. 7. Two-Point Crossover and Inverse Mutation for Data2.txt

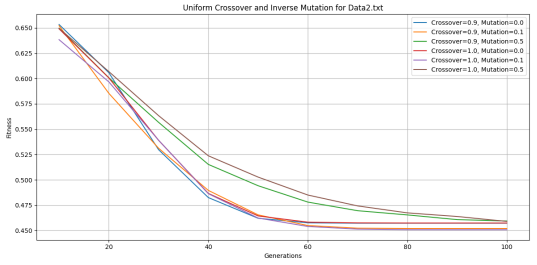

Fig. 8. Uniform Crossover and Inverse Mutation for Data2.txt

Also, Two-Point crossover seems to perform better with higher mutation as it consistently gives the best fitness, sometimes even without reaching convergence at the 100th generation. Further, $data1.txt$ gives better results with 0.9 crossover rate while $data2.txt$ performs better with a crossover rate of 1.0 for Two-Point Crossover. These trends are not as evident in Uniform Crossover.

B. Statistical Analysis and Student T-test

The previous observations are verified using Student T-Test as shown in Table I. The p-values for different crossover methods are all close to 0, indicating that the differences in their population means are statistically significant. The Student t-test uses aggregated averages from both types of mutations.

TABLE I
PAIRWISE T-TEST P-VALUES FOR CROSSOVER METHODS ON DATA1.TXT.

	Uniform Crossover	Two-point Crossover
Uniform Crossover	-	0.000000
Two-point Crossover	0.000000	-

The results shown in Table II and Table III are final averages for all the 6 combinations of crossover rates and mutation rates for 5 runs ($N = 5 * 6 = 30$) as mentioned in the Section III.

TABLE II
STATISTICAL SUMMARY FOR POPULATION AVERAGE FITNESS ON DATA1.TXT.

Method	Min	Max	Avg	Std
GA w/ Uniform Crossover & Inverse Mutation	0.082894559	0.086664154	0.084779096	0.001302799
GA w/ Uniform Crossover & ASCII Mutation	0.081902652	0.087927495	0.084091798	0.001854411
GA w/ Two-point Crossover & Inverse Mutation	0.1296725	0.285483587	0.193205691	0.056040256
GA w/ Two-point Crossover & ASCII Mutation	0.140039677	0.266650223	0.195460684	0.046056048

TABLE III
STATISTICAL SUMMARY FOR POPULATION AVERAGE FITNESS ON DATA2.TXT.

Method	Min	Max	Avg	Std
GA w/ Uniform Crossover & Inverse Mutation	0.450724409	0.459298362	0.455891048	0.003334905
GA w/ Uniform Crossover & ASCII Mutation	0.448895399	0.458926766	0.454456655	0.003508236
GA w/ Two-point Crossover & Inverse Mutation	0.383400733	0.578595345	0.492969971	0.075183883
GA w/ Two-point Crossover & ASCII Mutation	0.476975549	0.572889393	0.520833660	0.036196025

Uniform Crossover always outperforms Two-Point Crossover for Data1, irrespective of the Mutation type. For Data2, Two-Point Crossover performs better but with relatively higher standard deviation. This is due to the high performance we previously saw at 0.5 mutation rate in Section IV-A

Table IV and Table V compare the results using Student T-tests with a sample as of 30. They they show whether the differences in means are statistically significant. However, for a given crossover, there is not enough evidence that the mean for different types of mutation is different as the $p-values > 0.05$. This is true for both datasets.

TABLE IV
STUDENT T-TEST BETWEEN GA CONFIGURATIONS FOR DATA1.TXT (N = 30).

	Uniform + Inverse	Uniform + ASCII	Two-point + Inverse	Two-point + ASCII
Uniform + Inverse	–	0.102716	0.000000	0.000000
Uniform + ASCII	0.102716	–	0.000000	0.000000
Two-point + Inverse	0.000000	0.000000	–	0.865411
Two-point + ASCII	0.000000	0.000000	0.865411	–

TABLE V
STUDENT T-TEST BETWEEN GA CONFIGURATIONS FOR DATA2.TXT (N = 30).

	Uniform + Inverse	Uniform + ASCII	Two-point + Inverse	Two-point + ASCII
Uniform + Inverse	–	0.110002	0.011469	0.000000
Uniform + ASCII	0.110002	–	0.008921	0.000000
Two-point + Inverse	0.011469	0.008921	–	0.074551
Two-point + ASCII	0.000000	0.000000	0.074551	–

V. DISCUSSIONS AND CONCLUSIONS

A. Experiments Conducted

The experiments were performed with permutations of the following:

- 2 Data Files ($data1.txt$ and $data2.txt$)
- 2 Crossover Methods (Uniform and Two-Point Crossover)
- 2 Mutation Methods (Inverse and ASCII Mutation)
- 2 Crossover Rates (0.9 and 1)
- 3 Mutation Rates ($0, 0.1$ and 0.5)

There were a total of 48 configurations($2 * 2 * 2 * 2 * 3$), each of which were run 5 times. The population size was set to be 300 and 100 generations were simulated each run. The averages of each configuration with all possible mutation and crossover rates were used for statistical analysis.

B. Impact of Crossovers and Mutations Types

The mean fitness was the same for a given Crossover type in spite of different Mutation types. Mutation rate still held significance when compared on the same type of crossover.

Two-Point Crossover overall showed a higher standard deviation compared to Uniform crossover while showcasing relatively poor results. However, Two-Point crossover performed the best for Data2 with higher mutation rate of 0.5.

Inverse Mutation consistently outperformed ASCII mutation expect in one instance with Uniform Crossover for Data1.

C. Author's Opinion

Mutation rate 0.5 was expected to give premature convergence. However, this was not always the case. On the contrary, it led to the best results in some cases. This indicates that high mutation can yield better results in certain settings.

ASCII Mutation conducts a local search in the problem search space. Inverse Mutation is relatively more exploratory since it changes multiple chromosome indices. This led to the expectation that ASCII Mutation would largely outperform Inverse Mutation. However, this was not the case at all. Inverse Mutation generally outperformed ASCII Mutation.

D. Future Considerations

Here are some things to consider when repeating a similar experiemnt in the future:

- All the results compared the average results across 5 runs for each mutation and crossover configuration. One possible avenue for further consideration is comparing the further decomposing the results into different mutation and crossover rates to further optimize the results.
- Further, more experiments should be done with higher mutation rates to see if they provide even better results.

- Lastly, the experiment was only done with a 300 population size and 100 generations. Increasing both could lead to better results and insights.
- The fitness function could be improved by using a dictionary-based fitness metrics leading to a more accurate heuristic.

E. Conclusion

Genetic Algorithms can be effectively applied to break the Vigenère cipher. Performance varies with different combinations of crossover and mutation strategies. This report explores some possible approaches to this problem.

REFERENCES

[1] B. Ombuki-Berman, "Spring 2025 COSC 3P71 Artificial Intelligence / Genetic Algorithms Based Assignment," Assignment, Brock University, 2025.

[2] B. Ombuki-Berman, "Genetic Algorithms: An Introductory Overview," COSC 3P71 Lecture Slides, Brock University, 2025.

[3] R. Khosrowshahli, "Genetic Algorithm for Cipher Cryptanalysis—COSC 3P71—Assignment 1 Tutorial," unpublished tutorial notes, Brock University, 2025.