The Impact of a Genetic Algorithm System on Solving Vigenère Ciphered Strings

Karanjot Pabla
Department of Computer Science
Brock University
St. Catharines, Canada
Email: kp16km@brocku.ca

I. Introduction

When developing an intelligent agent that perceives and acts upon its environment, the performance measure (or goal) can be difficult to optimize without the introduction of genetic algorithms. Natural patterns found in life can help model the approach of how all perceive and act upon their environment. As the role of an intelligent agent is to replicate the processes that we can do, it makes the most sense to model them around patterns found in our existence. These patterns can help with the optimization in providing an agent`s goal, in which algorithms like the Genetic Algorithm come into play. Algorithms like Genetic Algorithms help forward computation of common paradigms, hard to compute otherwise.

From the great surge in impact the internet happens to play on our daily lives, Cryptography and Cryptanalysis has come to be the foreground in developing healthy and secure methods of storing our valuable information online. Whether this be confidential documents, personal account information, or online transactions; we all desire the safest and most protected storage of our personal information.

A simple, yet valuable example of a crypto system is the Vigenère Cipher. The Vigenère Cipher outlines a procedure of encrypting a provided string of characters with a specific key [7]. This key is what then can be used to decrypt the encrypted version of the initial string of characters. Using any other combo of characters to decrypt the encrypted text will result in a different string from the initially encrypted string of characters. This can make it difficult for outsiders to try uncovering an encrypted version of our information online. To try and generate a nearly infinite number of keys to decrypt a single string of characters can be extremely difficult to compute; knowing the limited amount of resources available within modern day machines. A great implementation for a crypto system like the Vigenère Cipher would be within online social media forums requiring the entry of our account passwords; as this entry will be passed through a numerous number of other machines [7].

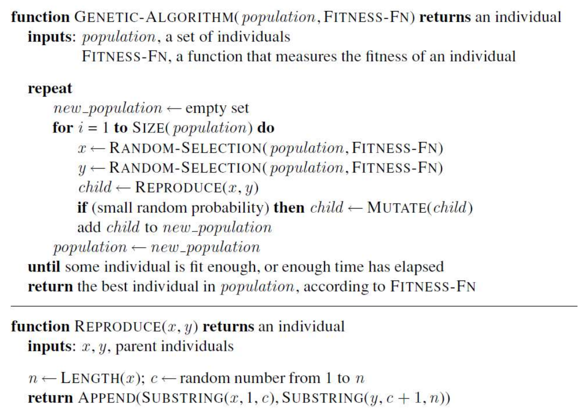
Though it can be difficult to decrypt an encrypted version of a string, with the help of Genetic Algorithms, the chances of uncovering a key for an encrypted string of characters can be significantly increased. To test this practice, using a simple genetic algorithm, I aimed to decrypt any encrypted string of characters. To conduct this, generating possible keys to an encrypted piece of text given how *fitting* each one is to the encrypted text, via a specific performance measure, helps allow the genetic algorithm to return the most *fitting* key from its set of computed test runs of generating possible keys.

Though my genetic algorithm may not compute the exact key to decrypt an encrypted piece of text every single run; it generates a near close version of the key most often. The significance in a simple genetic algorithm impacting the performance of decrypting a string without a key, goes to show the great potential genetic algorithms can provide to world of computation.

Understanding the finite number of resources available within today's machines, the benefit of Genetic Algorithms not only shows its significance in computing simple tasks like decrypting simple pieces of text, but also more revolutionary topics still in need of desperate research [2].

II. Background

To understand how a Genetic Algorithm System helps forward the computation of its goal, one must understand the background procedure of how genetic algorithms work.



```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
     new_population ← empty set
     for i = 1 to SIZE(population) do
        x ← RANDOM-SELECTION(population, FITNESS-FN)
        y ← RANDOM-SELECTION(population, FITNESS-FN)
        child ← REPRODUCE(x, y)
        if (small random probability) then child ← MUTATE(child)
        add child to new_population
     population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN

function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

Fig. 1. Pseudocode Example (Page 129) [1]

When looking at genetic algorithms, there are numerous ways of customizing one's algorithm, yet the general outline, principles, and parameters are shared amongst all. In the development of a simple genetic algorithm, one would follow a similar pseudocode as to the one shown in Figure 1.

To summarize the general pseudocode for a genetic algorithm; the genetic algorithm:

i) Accepts an input parent population solution (chromosome)

ii) Declares an evaluation function which is used to determine the fitness of an individual (gene) in a current solution (chromosome)

iii) For the duration of specific number of generations; starting at 1

   a. Computes fitness of each gene in the chromosome

   b. Creates a second chromosome parent using a specific selection technique (i.e. K Tournament Selection)

   c. Conducts Crossover and Mutation operators on the two parents

iv) And then returns the most fit solution from all the generations

III. Experimental Setup

For my experimentation using the genetic algorithm program I had developed, I aimed at evaluating the difference in using different crossover operators when running my genetic algorithm. As my genetic algorithm ultimately was aiming to return a solution it computed to be the most *fitting* according to its ability to decrypt a corresponding encrypted text, I evaluated a crossover operator by testing its ability to return a *fitting* solution after a specific amount of generations.

When preparing for my experimentation, there were various parameters, and crossover and mutator operators used when deciding my evaluation.

A. *Crossover Operators*

Crossover operators help create resulting offspring for a future generation, given the current generation parents; allowing for a way to evaluate upon a greater solution space than available initially [4].

The crossover operators I aimed at testing were:

i) 1-Point Crossover

   a. Random cut off point in two parent chromosomes occur, in which the genes preceding or succeeding the cutoff point are swapped in both parents; resulting in children of 2.

   b. In my experimentation, I chose to swap genes preceding the cut-off point. Otherwise, swap of succeeding genes works just as fine, as shown in Figure 2.

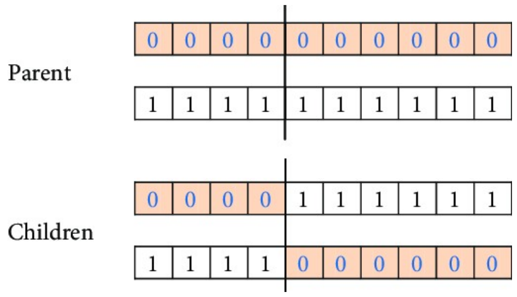


Fig. 2. One Point Crossover Example [2]

ii) 2-Point Crossover

   a. Two random cut off points in two parent chromosomes occur, in which the genes in between the cutoff points are swapped in both parents; resulting in children of 2.

   b. In my experimentation, I followed this exact procedure, as shown in Figure 3.

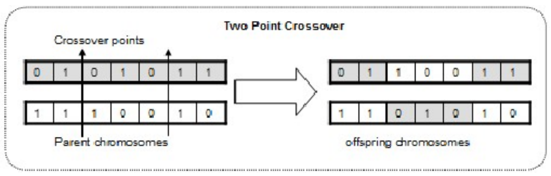


Fig. 3. Two Point Crossover Example [3]

B. *Mutation Operator*

Mutation operators help replicate the purpose of mutation in nature. This allows for an enhanced procedure in determining varied searches, allowing for less chances of premature convergence when running a genetic algorithm. [5].

The mutation operator I used when running my genetic algorithm was:

i) Bit-Flip Mutation

   a. This mutation operator aims to select a random gene within a given chromosome/solution and switch it with a random gene.

   b. In my experimentation, I performed this by switching a certain character (gene) in the string solution (chromosome), with another valid, randomly generated gene, as shown in Figure 4.

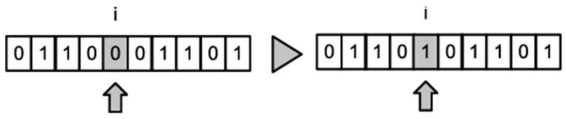


Fig. 4. Bit Flip Mutation [7]

C. *Genetic Algorithm Parameters*

These parameters helped tailor the way the genetic algorithm I performed testing with determine the evaluation of the crossover operators being used.

i) Encrypted Text

a. The string of text used to evaluate a generated chromosome/solution against to determine the ultimate fitness of the chromosome/solution; by decrypting the encrypting text using the current chromosome/solution. This helps the genetic algorithm determine the most optimal solution for the encrypted text.

b. In my experimentation, there were two encrypted texts I was evaluating upon. Both can be found in my program code for the genetic algorithm I developed.

ii) Chromosome (Solution) Size

a. Allows the genetic algorithm to help discover various fitting solutions to the encrypted text; knowing the length of the solution.

b. In my experimentation, as I was testing two different encrypted pieces of text. My first chromosome size was set at 26; to test the first encrypted piece of text. While my second chromosome size was set at 34; to test the second encrypted piece of text.

iii) Mutation Rate

a. To help test the effects of mutation on the overall evaluation of discovering a most optimal solution.

b. In my experimentation, this mutation rate is between 0 and 10; 0 meaning to perform mutation 0% of the time, while 10 means to perform mutation 100% of the time.

iv) Crossover Rate

a. To help test the effects that crossover can have on the overall evaluation of determining a most optimal solution.

b. In my experimentation, this crossover rate is between 0 and 10. 0 meaning to perform crossover 0% of the time, while 10 means to perform crossover 100% of the time.

v) Rate of Elitism

a. The chances of the best solution in a population (i.e. a parent) being passed on to the next generation.

b. For my experimentation purposes, I set this at 0.01; causing this to occur 1% of the time.

vi) Selection Pressure

a. The chances of selecting genes to form a new parent population from the initial population.

b. For my purpose of experimentation, I set this rate at 0.03. Meaning 3% of the time, genes will be selected from the initial population to form a selected population; the second parent.

vii) Seed

a. The seed determines the sequence of the random number generator. This helps allow for tracking of experimentation, helping to evaluate experimentation.

b. For my experimentation, I set the seeds at 2,3,4,5,6 respectively; allowing for 5 different runs of my genetic algorithms, with 5 different retraceable seeds.

viii) Pop-Size

a. The size of the population a parent chromosome should be considered.

b. For my experimentation, I set this as 100 in all experiments; making it controlled.

ix) K Tournament Value

a. The value used to determine the number of genes to select from a chromosome; to perform selection when generating a second parent/population

b. In my experimentation, I set this value at 3 for a controlled test on all runs.

x) Selection Operator

a. The type of operator used to select a second parent/population from an initial population

b. In my experimentation, I set this as K Tournament Selection. This works by selecting a k-number of individuals from an initial population and adding the best one to the second population/parent. Repeating this procedure until forming another population.

xi) Generations

a. The number of generations a genetic algorithm runs for to return an optimal solution.

b. In my experimentation, I set this at 200 for the first encrypted piece of text being evaluated. While, I had it set to 500 on my second encrypted piece of text. *Note; as I was testing the evaluation of crossover operators, experimentation of different encrypted text is isolated. Making it acceptable to run 200 generations on my first experimentation text, and 500 on my second experimentation text; for the purpose of a complete evaluation.

Now having to highlight all the set parameters and requirements of my genetic algorithm, I followed a simple scheme in evaluating the two crossover methods I selected:

For each encrypted piece of text, given a chromosome size:

Per each crossover operator (1 Point & 2 Point Crossover):

1. Per each of the following crossover and mutation rates:
    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. Crossover Rate: 90%, Mutation Rate: 20%
        i. Run the Genetic Algorithm 5 times; using 5 different random number seeds every time.
        ii. Seeds used: 2, 3, 4, 5, 6.

These tests were then stored and analysed to determine the difference in performance of the tested crossover operators; with the introduction of viable factors like mutation.

IV. Results

a) Text One Results

These tests were testing a given encrypted text with a chromosome size of 26.

After computing each of the crossover and mutation rates outlined in my procedure, for analysis I chose to evaluate the results from tests running on:

a. Crossover Rate: 100%, Mutation Rate: 0%

b. Crossover Rate: 90%, Mutation Rate: 20%

As the objective of experimentation was to evaluate the performance of using different crossover operators, I feel analyzing a crossover operator when running completely (100%), versus running slightly minimally (90%) with the introduction of mutation (20%), will help describe the difference in the tested crossovers.

The following shows a summary of the analyzed test results:

1-Point Crossover:

a) Crossover Rate: 100%, Mutation Rate: 0%

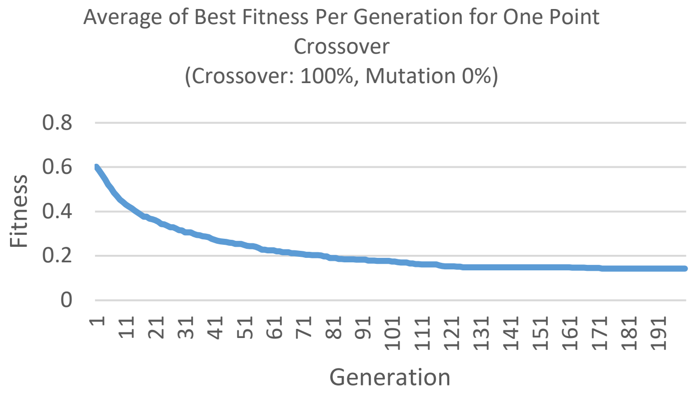


Fig. 5. Average of Best Fitness Per Generation for One Point Crossover

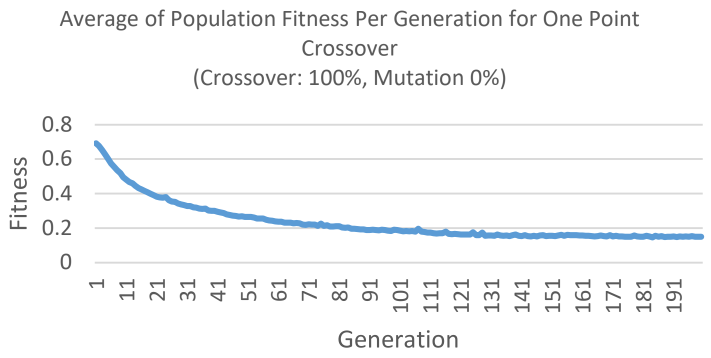


Fig. 6. Average of Population Fitness Per Generation for One Point Crossover

b) Crossover Rate: 90%, Mutation Rate: 20%

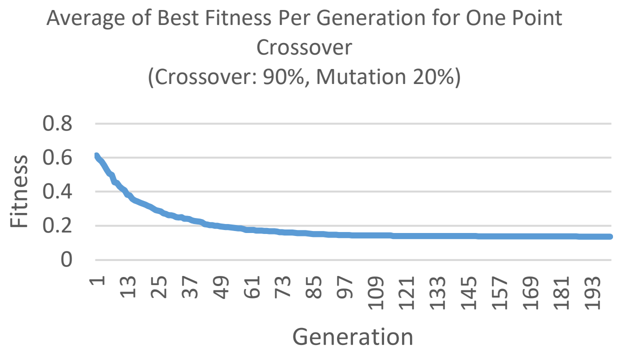


Fig. 7. Average of Best Fitness Per Generation for One Point Crossover

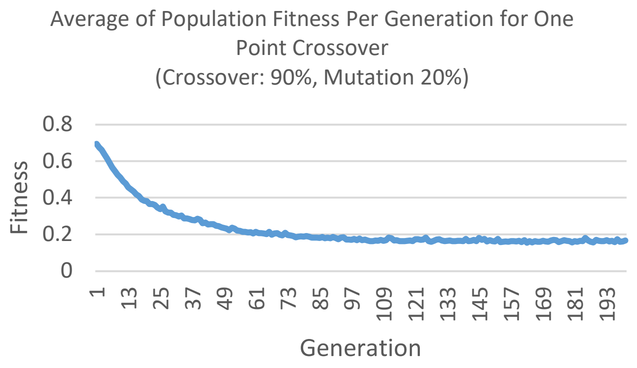


Fig. 8. Average of Population Fitness Per Generation for One Point Crossover

2-Point Crossover:

a) Crossover Rate: 100%, Mutation Rate: 0%

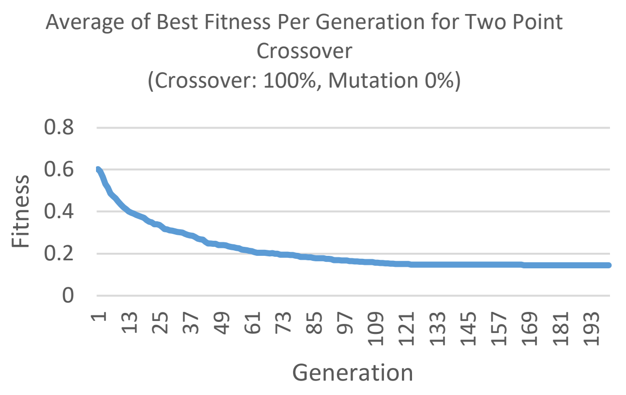


Fig. 9. Average of Best Fitness Per Generation for Two Point Crossover

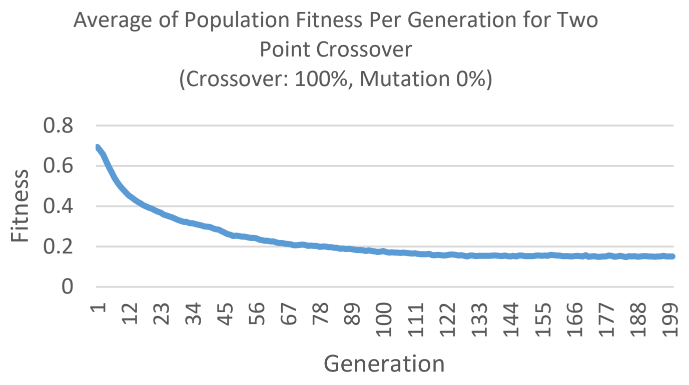


Fig. 10. Average of Population Fitness Per Generation for Two Point Crossover

c) Crossover Rate: 90%, Mutation Rate: 20%

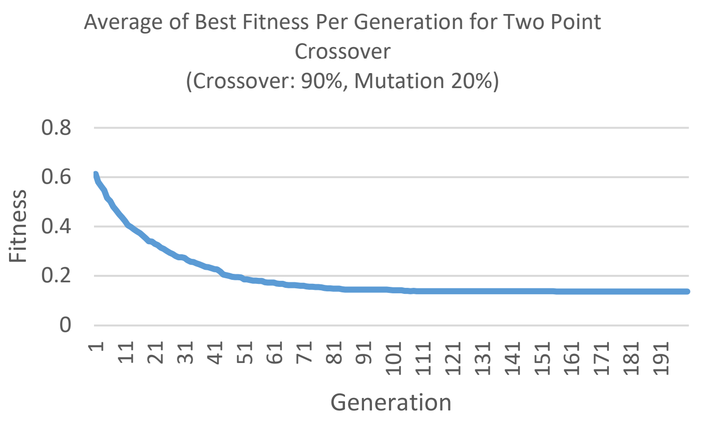


Fig. 11. Average of Best Fitness Per Generation for Two Point Crossover

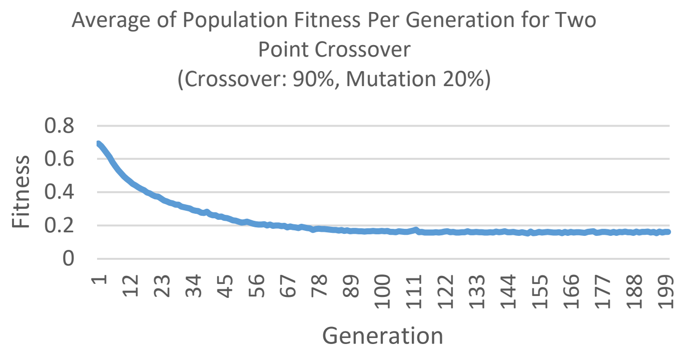


Fig. 12. Average of Population Fitness Per Generation for Two Point Crossover

To help show the variation in the results for One Point Crossover and Two Point Crossover from Text One results, the following summary tables and graphs compare the same tests that were run on the different crossover methods (for the best fitness per generation):

a. Crossover Rate: 100%, Mutation Rate: 0%

TABLE I. COMPARISION OF 1 POINT AND 2 POINT CROSSOVER

Regression Statistics	
Multiple R	0.998698092
R Square	0.997397878
Adjusted R Square	0.997384736
Standard Error	0.00499681
Observations	200

[a] Average of Best Fitness (Crossover:100%, Mutation: 0%)

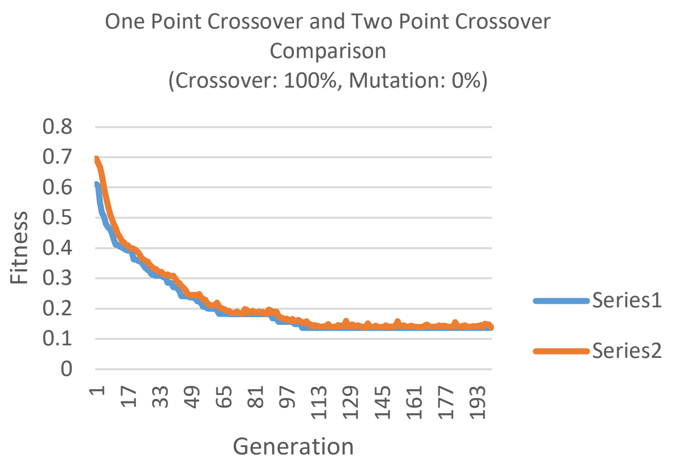


Fig. 13. Comparison of Best Fitness Per Generation for One Point Crossover and Two Point Crossover (Series 1 = Two Point Crossover, Series 2 = One Point Crossover)

b. Crossover Rate: 90%, Mutation Rate: 20%

TABLE II. COMPARISION OF 1 POINT AND 2 POINT CROSSOVER

Regression Statistics	
Multiple R	0.995538338
R Square	0.991096583
Adjusted R Square	0.991051616
Standard Error	0.009370595
Observations	200

b. Average of Best Fitness (Crossover:90%, Mutation: 20%)

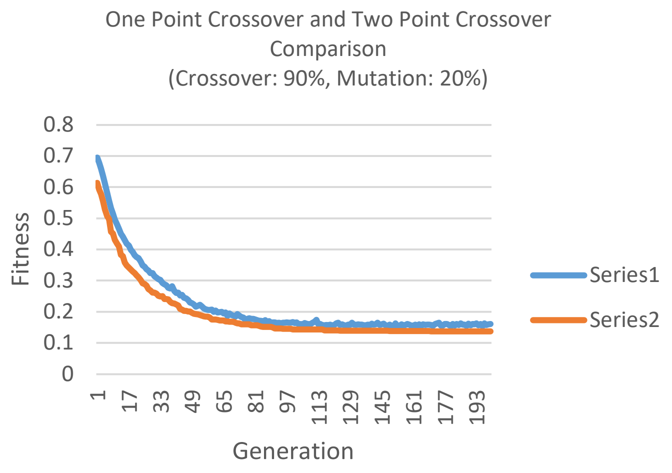


Fig. 14. Comparison of Best Fitness Per Generation for One Point Crossover and Two Point Crossover (Series 1 = Two Point Crossover, Series 2 = One Point Crossover)

From just a general glance at the graphs and summary tables formulated above. There tends to be a significant similarity in the behavior of the Genetic Algorithm when determining a fit chromosome. The graphs show that the general determination for a One Point Crossover fueled Genetic Algorithm and Two Point Crossover fueled Genetic Algorithm have similar convergence and even a high correlation coefficient.

For instance, in Figure 13, though both algorithms start off at a different fitness, the both follow a similar trendline; where they converge and drop at similar rates. To add to this, Table I describes the R Square value to be significantly high. This would mean the two Crossover operators have a low variance. In return, most of the variance in any observed variation is determinable by the given model.

V. DISCUSSIONS AND CONCLUSIONS.

When going about my experimentation, my objective was to see the impact the change in a crossover operator can have on the performance of a genetic algorithm; specifically, the One Point and Two Point Crossover operators. To test this, I used different mixes of crossover and mutation rates to form a general conclusion of whether there really is a great impact on the use of a different crossover operator.

To narrow down the results, I chose to compare the following tests from my experimentation:

a. Crossover Rate: 100%, Mutation Rate: 0%

b. Crossover Rate: 90%, Mutation Rate: 20%

After conducting my experimentation, the results showed a similar trend in the use of different operators. As shown in figure 13 and Figure 14; it is noticed that though the genetic algorithm is running on different crossover methods, they both converge and return a solution at similar pace.

Though this being true, for some reason it seems that the two-point crossover performs a little poorly when compared to the one-point crossover with the introduction of mutation. When the one-point crossover runs and provides the most fit solution from all generations, it is slightly more fit (closer to 0) when compared to the two-point crossover; under the same conditions, as shown in Figure 14.

Otherwise, when the two crossover methods are being compared under conditions with no mutation, and a 100% crossover rate, they both perform almost equivalently. Sure, the two-point crossover starts at a slightly more fit chromosome, but they both tend to converge to a similar solution fitness.

Collectively, taking a step back and understanding the similarity in the two crossovers does help highlight on the results of this experimentation. Rather than having to swap genes in a parent solution with another parent solution at one point, the two-point crossover aims to swap a parent solution`s genes at two points. Due to this similarity, it can be justified in the results showing a similar behaviour in both genetic algorithms.

As I kept generally all the parameters the same, except for the mutation and crossover rates, it helped evaluate the true value being added in a genetic algorithm running on a specific crossover operator.

When testing both crossover operators, they both were able to return the correct solution to the encrypted text I provided it with for some of it runs. As Genetic Algorithms are randomly determined, it was obvious that the genetic algorithm would not return the correct solution every single time, as sometimes it would provide a more *fit* solution than the one that would be considered the correct key for the encrypted piece of text.

For Text 1, under one-point crossover, with a seed of 2, it provided a solution of “hisisasupersecurepasswordt”, with a fitness value of 0.13709809316694738.

To summarize my findings, from the results of my experimentation, it was observed that after the introduction of mutation, only then did the two-point crossover method start to perform better than the one-point crossover method. As mutation does add the enhanced procedure in determining varied searches, allowing for less chances of premature convergence when running a genetic algorithm [5]; it is assumed that the introduction of a more varied search would help two-point crossover perform better than the one-point crossover method. Whether this being raising the mutation rate, or possibly even reducing the selection pressure; in return would allow from a more varied space for the two-point crossover method to perform effectively. Otherwise, the difference in both crossover methods would be negligible, given the impact of mutation on results.

References

[1] S. Russell, and P. Norvig, *Artificial intelligence: A Modern Approach.* 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2010.

[2] S. Asghar, and R. Shah.. *Privacy Preserving in Association Rules Using Genetic Algorithm.* Turkish Journal of Electrical Engineering and Computer Sciences. 22. 10.3906/elk-1206-66, 2014

[3] Y. Kaya, M. Uyar, and R. Tekin. *A Novel Crossover Operator for Genetic Algorithms: Ring Crossover.* CoRR. abs/1105.0355, 2011

[4] G. Pavai, and T. V. Geetha, *A Survey on Crossover Operators.* ACM Comput. Surv. 49, 4, Article 72 43 pages, 2016. DOI: https://doi.org/10.1145/3009966

[5] T. Hong, H. Wang, and W. Chen, *Journal of Heuristics* 6: 439., 2000. https://doi.org/10.1023/A:1009642825198

[6] O. Rifki, and H. Ono, *A survey of computational approaches to portfolio optimization by genetic algorithms,* 2012. 10.13140/RG.2.1.5192.8561.

[7] G. Simmons, *Vigenère cipher*. Encyclopædia Britannica Inc, 2017