Assignment 2

Fouzan Abdullah
6840797
Brock University
St. Catherine's, Canada
fa19vm@brocku.ca

***Abstract*—**

I. Introduction

Genetic algorithms (GA's) are evolutionary algorithms that are designed using evolution methods observed in the natural world. The main concepts used in these algorithms are reproduction, variation, and selection. The process of reproduction allows for individuals to replicate and variation is used to ensure the offspring are not identical. The process of selection assures that the fittest individuals of the population are selected to reproduce [1]. The benefit of using a genetic algorithm is that they can be applied to many kinds of problems that have an easy way to calculate the fitness of a solution. An example, and the focus of this report, is to implement a GA that generates keys which are responsible for performing crypt-analysis on given data files in order to decrypt various encrypted text files. [2] In this report, we will conduct several experiments using different algorithm parameters, different crossover methods such as uniform crossover, and one-point crossover and summarize our findings.

II. Background

A genetic algorithm (GA) is implemented in the provided code to crack a substitution cipher. A substitution cipher is a text encryption technique in which a different letter is used for each letter in the plain-text. The goal of the GA is to optimize the fitness function, which gauges how closely the letter frequencies in the decrypted text resemble those of the English language, by evolving a population of potential decryption keys. In a genetic algorithm, the different procedures involved are fitness evaluation, tournament selection, mutation, and crossover.

The fitness evaluation function calculates the fitness score of a decryption key by comparing the frequency distribution of decrypted test with the provided English alphabet letter frequencies. The parameters used within the fitness function is a 'key' which acts as a decryption key, and 'text' which is the encrypted text that we aim to decrypt. This function returns a fitness score that is selected within each generation by checking the fitness of all members of the population within that generation. This repeats for the number of generations we have and eventually our genetic algorithm provides a best key out of all generations that is used to decrypt the encrypted text that we provide. Next is the function that initializes the population. It generates an initial population at random and checks the fitness of each individual within that population. It takes a parameter of 'pop-size' which is the size of the population that is predefined by the user, and it takes the chrom-length, which is set to be the length of the chromosome (i.e. the length of each decryption key). It returns the initial population to use for our functions defined further. The third function is 'select-parents' which implements tournament selection that picks the two best parents for crossover. This takes the parameters of 'population' which is the current population elements, 'fitness-func' which is the fitness function that we have already created, and 'text' which is the encrypted text that it reads through. This function then returns two selected parents that are used for our further functions.

Then we move on to our crossover functions, which include the one-point, uniform order, and three-point crossover. The 'one-point-crossover' function performs one-point crossover on the two selected parents and returns two children that are then used further. The parameter input are 'parent1' and 'parent2' and the output is 'child1' and 'child2'. Similarly, the 'uniform-crossover' function performs uniform crossover on the two selected parents and returns two children which are used. The parameter input are also 'parent1' and 'parent2' and the output is 'child1' and 'child2'. Lastly, just like the other two the parameters for the 'three-point-crossover' is also 'parent1' and 'parent2'. It would also return the two children after crossover for further use. Then we move on to the 'mutate' function which introduces random mutations by changing a little bit of each population to test out the genetic algorithm. This is done for each generation by taking a parameter of 'chromosome' which is the character set that undergoes mutation, and 'mutation-rate' which is the rate of mutation as defined by us at the start of the code. This function iterates through each character in the set and checks whether a random number between 0 and 1 is less than the mutation rate. If that is the case, that character is mutated by randomly selecting a character in the set and replacing it. These new mutated characters are stored in a 'mutated-chromosome' set. Then finally the function returns the mutated-chromosome key as a string.

Then we have the elitism function which selects a portion of the population with the best fitness (so called elites) and carries them over to the next generation, ensuring that it is included within the next generation for best results. The parameters used in this function are 'current-population' which is the current population, 'previous-population' which is concatenated with

the current population to create a 'combined-population' parameter. 'fitness-func' and 'text' are also used which are the fitness function and encrypted text respectfully. The fitness scores for each individual in the combined population is calculated and sorted based on the scores. The top individuals (or elites) are selected and a new population is formed combining those elites and random individuals to reach the population size. And finally 'elite-percentage' which is the percentage of the population that are selected as elites. The function returns a new population which contains the elites. Then we have the 'decrypted-text' function which decrpyts the given text using a decryption key. The input parameters are 'key' which is the decryption key and 'text' which is the encrypted text that we have to decrypt. This function returns the decrypted text which is then printed out for our final output.

Finally we have the main genetic algorithm loop where all the previously defined functions are ran in. It initializes a population and iteratively evaluates the fitness, performs crossover, mutation, and updates the population based on the output. The genetic algorithm also collects data on the best fitness for each generation, as well as the average fitness for each generation. This is done due to ease of results and to be easily able to compare the different crossover methods that I have implemented. Pseudocode for my genetic algorithm is as follows:

Algorithm 1 PSeudocode

```
initialize population
for generation in range(num-generations):
    Evaluate fitness for each individual in the population
    Select elites from the population
    new-population = elites

while new-population size less than population size do
    Select parents from the current population
    Apply crossover to create offspring
    Apply mutation to the offspring
  current-population = new-population
Output best individual
```

III. Experimental Setup

To setup the experiment, there are a few algorithm parameters that we need to initialize before running the code.

A. Population size

'population-size'. This is initialized at the start of the code. For my results we will be taking a population size of 50, and a 100 to test out my code thoroughly. The population consists of individuals in each generation of the genetic algorithm.

B. Chromosome length

'chromosome-length'. This is also initialized at the start of the code. This represents the length of our chromosome, which will be the decryption key as well. It varies for both of the pieces of text that I have. For Data1.txt I used a chromosome length of 26, whereas for Data2.txt I used a chromosome length of 40.

C. Mutation rate

'mutation-rate'. This is also initialized at the start of the code. This is the rate at which mutation occurs. I will be testing it at 5 different values which are 0 percent, 10 percent, and 100 percent. It represents the probability of mutation for each character in the key.

D. Number of generations

'num-generations'. This is also initialized at the start of the code. This represents the number of generations for which the genetic algorithm runs. For my results, I will test the genetic algorithm at a constant number of 50 generations.

E. Crossover rate

'crossover-rate'. This is also initialized at the start of the code. The crossover rate is the probability at which crossover occurs during reproduction. It combines genetic material from parent1 and parent2 to create offspring. For my results, I will test the genetic algorithm at a crossover rate of 100 percent, 90 percent, and 10 percent.

F. Elitism percentage

'elite-percentage'. This is initialized within the elitism function. This represents the percentage of the population that are considered elites, and are carried over to the next generation.

G. Crossover operators

1) Selected crossover method: This allows the user to select between the three types of crossovers that I have implemented. The user will pick 'o' for one point crossover, 'u' for uniform order, and 't' for three point crossover.

2) One Point Crossover: In this crossover a single crossover point is selected within the parent node, and the remainder portion of the parent is swapped on either side to create two children.

3) Uniform Crossover: For each crossover point (or index) a random decision is made whether to inherit the gene from either parent.

4) Three Point Crossover: Three crossover points are selected at random, and the portion between these three points are swapped to create two children.

H. Mutation operators

1) Mutation function: Contains a mutation function that introduces random mutations into the chromosome. A character is selected at random based on the probability determined by the mutation rate and the corresponding character is replaced by the random character.

I. Selection operator

1) Selection function: This function implements tournament selection to pick two parents for crossover. These individuals are randomly selected from the population based on their best fitness. This process is then repeated to select the second parent and perform the required operators on them.

J. Elitism operator

1) Elitism function: This function takes a combination of the current and previous population, evaluates their fitness and picks elites which are chromosomes with the best fitness. These chromosomes then have a guaranteed spot in the next generation.

K. Other operators worth mentioning

1) Fitness function: This function is responsible for evaluating the fitness of the chromosomes which then picks the best one to be used as the decryption key. This is done by comparing the fitness of each chromosome to the frequency of each letter in the alphabet.

2) Initialization function: This function is responsible for creating and initializing a population of chromosomes.

3) Decryption function: This function performs decryption on the given text file by using a decryption key that we generated using the previous functions.

In order to reproduce the results for all the test cases, the user needs to:

1. Set the parameters according to the specified values.

2. Pick a crossover method by providing the appropriate input 'o' for one point, 'u' for uniform, and 't' for three point.

3. Ensure the text files 'Data1.txt' and 'Data2.txt' contain the appropriate data required for decryption.

4. Finally, run the code with the selected parameters and crossover method.

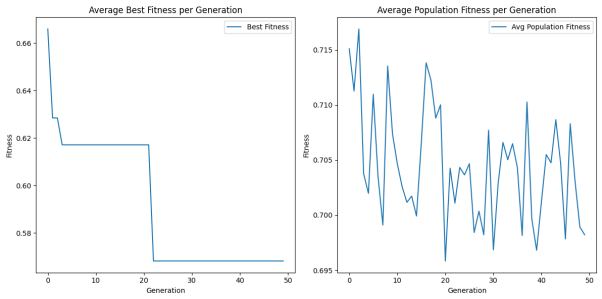
The aim of this setup is to find the best decryption key that maximizes the fitness function which would lead to a successful decryption. The combination of the genetic operators, algorithm parameters, and crossover methods lead to a in depth exploration of the genetic algorithm, which gives an optimal solution.

IV. Results

This table shows the parameters that I will be using to display the results. I will run these parameters on Data1.txt and using the two different crossover methods to produce results that will be displayed as graphs and a summary table for all the statistics.

Parameters	Crossover Rate	Mutation Rate	Chromosome Length	Population Size	Number of Generations
Run 1	100%	0%	26	50	50
Run 2	100%	10%	26	50	50
Run 3	90%	0%	26	50	50
Run 4	90%	10%	26	50	50
Run 5	10%	100%	26	50	50

For Run 1 using Data1.txt and the one-point crossover method:

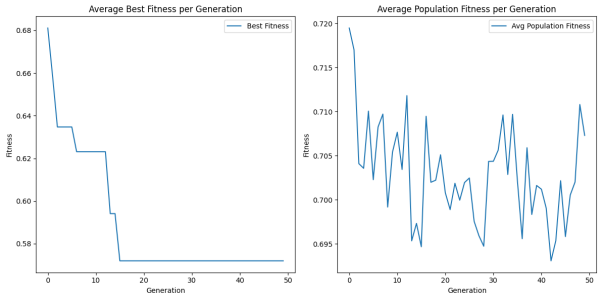


Data1.txt Run1 One Point Crossover

The figure shows the best fitness per generation for the number of generations I ran, as well as the average fitness per generation. As you can see the best fitness goes downhill which indicates that it improves every generation. There is a plateau which indicates that the fitness stays the same for a few generations in between. The average fitness varies for each generation which shows that the randomized population may not have the best average fitness every next generation. I ran Run1 5 times to get the best overall result and displayed that in the graph.

For Run 2 using Data1.txt and the uniform crossover method:

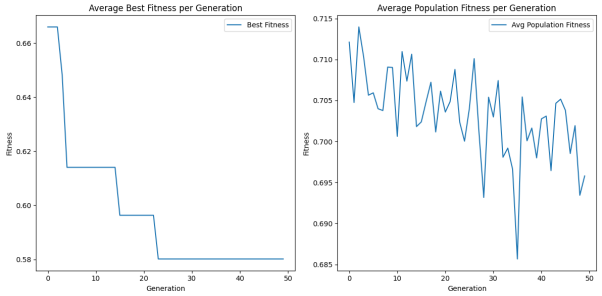
Data1.txt Run2 Uniform Crossover



The figure shows the best fitness per generation using the parameters for Run 2 in the same Data1.txt file. Similarly to figure 1, the best fitness goes downhill. Run 2 was run 5 times as well to ensure that we get the best overall fitness per generation and a more exact generation of the average fitness per generation.

For Run 3 using Data1.txt and the one point crossover method:

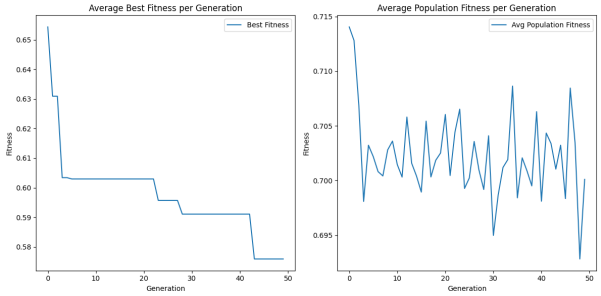
Data1.txt Run3 One Point Crossover



This figure uses the parameters for Run 3 in Data1.txt. Run 3 was also implemented 5 times with the parameters to display the best fitness per generation.

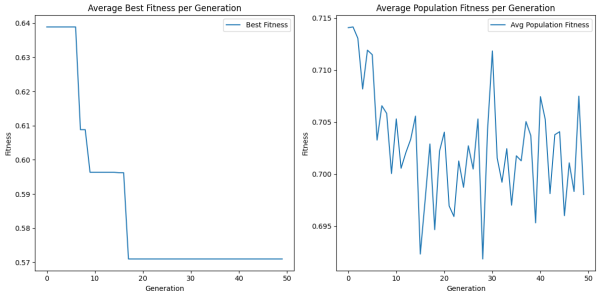
For Run 4 using Data1.txt and uniform crossover method:

Data1.txt Run4 Uniform Crossover



For Run 5 using Data1.txt and one point crossover method:

Data1.txt Run5 One Point Crossover



The graph for Run 5 is also run 5 times to ensure best overall fitness per generation. The parameters I used were a crossover rate of 10% and a mutation rate of 100%. The rest of the parameters remain the same.

The summary statistics are as follows:

Data1.txt

	Min-Value	Max-Value	Median	Mean	Std. Deviation
Run 1: One point	0.5682388219544847	0.6659630522088352	0.5682388219544847	0.5911904899598396	0.026806390194313547
Run 2: Uniform	0.5719598393574299	0.6811140562248996	0.5719598393574299	0.588946168674699	0.028305828548074435
Run 3: One point	0.5801499330655955	0.6659630522088353	0.5801499330655955	0.5966991807228915	0.023555909824971803
Run 4: Uniform	0.5758835341365462	0.6543338688085675	0.5956819277108434	0.5970226827309238	0.014117147518906 46
Run 5: One point	0.5709686746987951	0.6388942436412316	0.5709686746987951	0.5860482463186077	0.024055795129415285

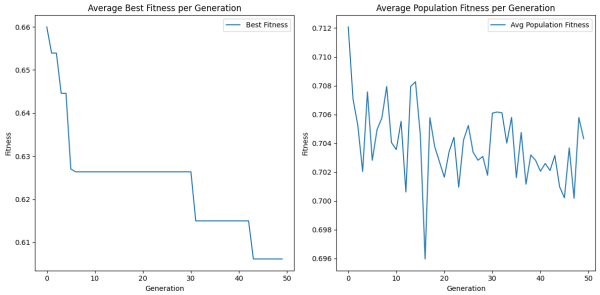
This table shows the parameters that I will be using to display the results. I will run these parameters on Data2.txt and using the two different crossover methods to produce results that will be displayed as graphs and a summary table for all the statistics.

Parameters

	Crossover Rate	Mutation Rate	Chromosome Length	Population Size	Number of Generations
Run 1	100%	0%	40	100	50
Run 2	100%	10%	40	100	50
Run 3	90%	0%	40	100	50
Run 4	90%	10%	40	100	50
Run 5	10%	100%	40	100	50

For Run 1 using Data2.txt and the one-point crossover method:

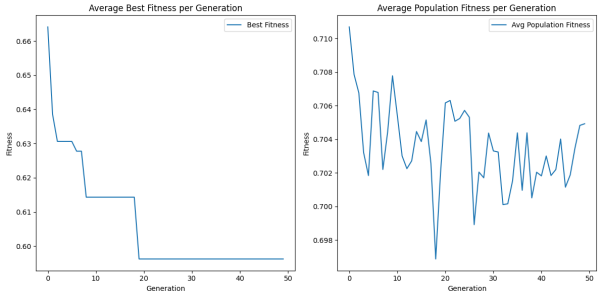
Data2.txt Run1 One Point Crossover



The figure shows the best fitness per generation for the number of generations I ran, as well as the average fitness per generation. As you can see the best fitness goes downhill which indicates that it improves every generation. There is a plateau which indicates that the fitness stays the same for a few generations in between. The average fitness varies for each generation which shows that the randomized population may not have the best average fitness every next generation. I ran Run1 5 times to get the best overall result and displayed that in the graph.

For Run 2 using Data2.txt and the uniform crossover method:

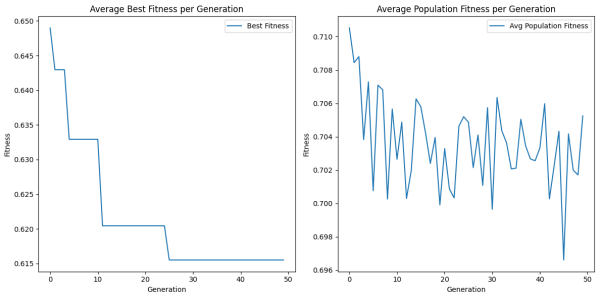
Data2.txt Run2 Uniform Crossover



The figure shows the best fitness per generation using the parameters for Run 2 in the same Data1.txt file. Similarly to figure 1, the best fitness goes downhill. Run 2 was run 5 times as well to ensure that we get the best overall fitness per generation and a more exact generation of the average fitness per generation.

For Run 3 using Data2.txt and the one point crossover method:

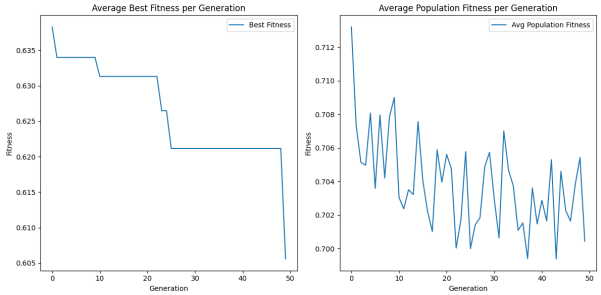
Data2.txt Run3 One Point Crossover



This figure uses the parameters for Run 3 in Data1.txt. Run 3 was also implemented 5 times with the parameters to display the best fitness per generation.

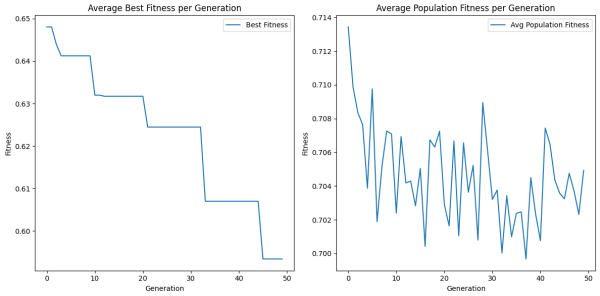
For Run 4 using Data2.txt and uniform crossover method:

Data2.txt Run4 Uniform Crossover



For Run 5 using Data2.txt and one point crossover method:

Data2.txt Run5 One Point Crossover



The graph for Run 5 is also run 5 times to ensure best overall fitness per generation. The parameters I used were a crossover rate of 10% and a mutation rate of 100%. The rest of the parameters remain the same.

The summary statistics are as follows:

Data2.txt

	Min-Value	Max-Value	Median	Mean	Std. Deviation
Run 1: One point	0.6061475234270414	0.6386572958500671	0.6275668005354752	0.6270865542168675	0.0082320828462428 17
Run 2: Uniform	0.5996979919678714	0.6351732262382864	0.6001234270414992	0.6062098045515395	0.011427593066773746
Run 3: One point	0.6155183400267737	0.6489855421686748	0.6179829986613119	0.6216508915662651	0.008898570911588913
Run 4: Uniform	0.6055850066934406	0.6382696111780455	0.6238132530120482	0.6263417563587685	0.0064214137600435 34
Run 5: One point	0.593429986613119	0.6480281124497993	0.6244637215528782	0.6224570441767069	0.015694444637039454

V. Discussion and Conclusion

As we can see from both pieces of data, Data1.txt and Data2.txt, the one point crossover yields better results than uniform crossover. Both crossovers were applied consistently over the generations for both data sets, with one point giving a better overall fitness. As for the mean, or average fitness per generation, uniform crossover yielded a better result as compared to the one point crossover method. The graph for uniform crossover of the average fitness per generation is a bit more stable as compared to the one for one point crossover. Moreover, one point crossover reaches a plateau faster than the uniform crossover this is because it is easier to perform one point crossover than uniform crossover.

To conclude,

The genetic algorithm I implemented was applied to decrypt the given two sets of encrypted text using a substitution cipher. Three crossover methods were implemented where the user picks which one they want to go with. The parameters have to be changed every run based on the user requirements. The genetic algorithm was run for a population size of 50, over 50 generations for Data1.txt and a population size of 100, over 50 generations for Data2.txt. The mutation rate varied for each run and elitism was performed every run to ensure that the best individual are preserved.

The best fitness values for each generation were recorded and plotted along side the average fitness values for each generation. This is shown within the graphs that I included with all the different runs for both data sets with different crossover methods applied to each run.

1) Crossover Performance: The user is prompted to pick one of the 3 crossover methods implemented and they are executed in each generation.

2) Mutation: The mutation rate is set to 0%, 10%, and 100% which means that mutation is applied to every gene in the chromosome.

3) Elitism: Elitism is applied to preserve the best individuals in each generation. The elite percentage is set to 10% which I believe to a fair percentage to test out the function.

4) Fitness Function: The test for the fitness function is to see how well it aligns with the encrypted text and how well it is able to decrypt it.

5) Population size and Number of Generation: I experimented with a set population size of 50 and a set number of generations of 50 for the first data set, and a set population of 100 and a number of generations of 50 for the second data set. This allowed me to thoroughly test my functions.

6) Analysis and Data Visualization: I plotted the values for the best fitness of each generation and the average fitness of each generation which provides insight on how well the algorithm works.

7) Decryption: The decrypted text is printed with the best individual selected to show how well that individual works on decrypting the text.

References

[1] *B. Ombuki, "Genetic Algorithms" presented to COSC 3P71 Intro. to Artificial Intelligence, Brock University, St. Catharines, ON, Canada, November, 2023.*

Fig. 1. Parameters, Crossover types, Crossover rate, Mutation rate, Minimum Values, Maximum Values, Mean Values, Median Values, and Standard Deviation for both Data sets

Data Set	Crossover Type	Run Number	Crossover Rate	Mutation Rate	Min_Values	Max_Values	Mean_Value	Median_Value	St. Deviation
Data1.txt	One Point	Run 1	100%	0%	0.606147523	0.638657296	0.627566801	0.627086554	0.008232083
		Run 2	100%	10%	0.599697992	0.635173226	0.600123427	0.606209805	0.011427593
		Run 3	90%	0%	0.61551834	0.648985542	0.617982999	0.621650892	0.008898571
		Run 4	90%	10%	0.605585007	0.638269612	0.623813253	0.626341756	0.006421414
		Run 5	10%	100%	0.593429987	0.648028112	0.624463722	0.622457044	0.015694445
	Uniform	Run 1	100%	0%	0.568238822	0.665963052	0.568238822	0.59119049	0.02680639
		Run 2	100%	10%	0.571959839	0.681114056	0.571959839	0.588946169	0.028305829
		Run 3	90%	0%	0.580149933	0.665963052	0.580149933	0.596699181	0.02355591
		Run 4	90%	10%	0.575883534	0.654333869	0.595681928	0.597022683	0.014117148
		Run 5	10%	100%	0.570968675	0.638894244	0.570968675	0.586048246	0.024055795
	Three Point	Run 1	100%	0%	0.631719946	0.648028112	0.637611442	0.622457044	0.012861262
		Run 2	100%	10%	0.631719946	0.638894244	0.626106718	0.624463722	0.015290396
		Run 3	90%	0%	0.575883534	0.654333869	0.595681928	0.588946169	0.014832696
		Run 4	90%	10%	0.570968675	0.638894244	0.570968675	0.624463722	0.006421414
		Run 5	10%	100%	0.631719946	0.648028112	0.637611442	0.586048246	0.012861262
Data2.txt	One Point	Run 1	100%	0%	0.606147523	0.638657296	0.627566801	0.627086554	0.008232083
		Run 2	100%	10%	0.599697992	0.635173226	0.600123427	0.606209805	0.011427593
		Run 3	90%	0%	0.61551834	0.648985542	0.617982999	0.621650892	0.008898571
		Run 4	90%	10%	0.605585007	0.638269612	0.623813253	0.626341756	0.006421414
		Run 5	10%	100%	0.593429987	0.648028112	0.624463722	0.622457044	0.015694445
	Uniform	Run 1	100%	0%	0.607006693	0.648028112	0.626106718	0.596699181	0.012861262
		Run 2	100%	10%	0.599697992	0.635173226	0.600123427	0.624463722	0.012861262
		Run 3	90%	0%	0.61551834	0.648985542	0.617982999	0.596699181	0.008232083
		Run 4	90%	10%	0.605585007	0.638269612	0.623813253	0.624463722	0.012861262
		Run 5	10%	100%	0.605585007	0.638269612	0.626106718	0.621650892	0.028305829
	Three Point	Run 1	100%	0%	0.568238822	0.665963052	0.568238822	0.59119049	0.02680639
		Run 2	100%	10%	0.571959839	0.681114056	0.571959839	0.588946169	0.028305829
		Run 3	90%	0%	0.580149933	0.665963052	0.580149933	0.596699181	0.02355591
		Run 4	90%	10%	0.575883534	0.654333869	0.595681928	0.597022683	0.014117148
		Run 5	10%	100%	0.570968675	0.638894244	0.570968675	0.586048246	0.024055795