Using Genetic Algorithms to Decrypt Substitution Ciphers

Scott Hodges and Adam Carcione
Franklin & Marshall College
publications18@aaai.org

Abstract

This paper will discuss the approach of finding a key for a substitution cipher through the avenue of genetic algorithms. Genetic algorithms are very efficient in searching through large problem spaces and thus are ideal for the task of trying to find the correct key out of the possible 26! Keys for a substitution cipher. This paper will cover the problems that arise when attempting to break a substitution cipher, and how to overcome these problems. It will also cover the design choices made regarding selection and mutation methods and which of them yielded the best results. Finally, it will cover the experimental data recorded from running many tests with varying parameters in order to find which set of parameters would return the best result.

Substitution Cipher

A cipher is an algorithm that is used for encrypting plain text into a cipher text and vice versa. The cipher studied in this setting is the substitution cipher, which functions by replacing every letter in the plain text with its corresponding letter from the cipher key. An example cipher key that could be used for encrypting plain text can be seen here:

For example, using the above key for encryption the string "This project was fun" is replaced with "Ptij yoqkvsp bej awg".

Substitution ciphers can be strong because they have 26! possible keys because there are 26 options for the first letter, 25 for the second, and so on. Our approach to this problem was to use the frequencies of certain groupings of letters in order to try and figure out the correct key by favoring keys that have frequencies similar to that of the frequencies in the training text.

Encoding Individuals

Every individual in our population consists of 26 characters where every character is a unique letter in the English alphabet. Therefore every individual in the population is defined by the letter and index of each character in its key. An example individual in a population might be "PQOWIEURYTLAKSJDHFGMZNXBCV", which would represent a key that maps every letter to the letters of the alphabet in order. For example P, Q, and O would map to A, B, and C.

N-grams and Frequency

n-grams are used to perform a frequency analysis of a given text. n-grams are a substring, of length n, taken from a larger string or text. So from the sentence "Experimental analysis is great" the first n-gram of length 3 would be "EXP", the first n-gram of length 2 would be "EX" and so on. Spaces are not included in n-grams; if an n-gram has a space in it, the n-gram is skipped and the next n-gram is analyzed.

A frequency table is a representation of the frequency of all possible n-grams, for a given n, in the training text. The frequency table was created by counting all n-grams in the training text, in this experiment n = 3, and creating a dictionary with the keys being all the possible n-grams and the value $log_2(n\ gram\ count)$ being assigned to each n-gram of length n. We will denote the value in the frequency table for a given n-gram x, as T(x).

Fitness

Fitness is the value assigned to each individual in the population, based on the number of bigrams in the text produced, when the individual is used to decrypt the given text. The term "decrypted text" is used here to denote the text produced by the individual, when it is used to for an attempt at a decrypted text. The term "decrypted text" will be used to denote this text, but by no means does this mean the produced text has been successfully decrypted. This is important to note, and this terminology will be used throughout this report. The fitness for a given individual is evaluated by counting all the individual n-grams in the decrypted text and storing the count for each n-gram. The n-grams present in the decrypted text will be denoted by B(x) and the count of the corresponding individual will be denoted by C(x). The following summation is the fitness score of an individual: Fitness(x) = $\sum_{i \in C(x)} C(x) * T(x)$. The fitness value, in words, is the sum of the count all n-

grams, which appeared in the decrypted text, multiplied by their value in the frequency table. The fitness score is an indication of how accurate a decrypted text is, because the more true to the training text a decrypted text is, the higher the fitness score will be. This occurs, because the training text is used to create the frequency table, so very frequently occurring n-grams in the training text will have higher values in the frequency table. If an individual creates a decrypted text with these frequently occurring n-grams in it, it is more likely to be an accurate individual, and will have a higher fitness score, because of it produces frequently occurring n-grams. This frequency analysis is what enables the merit of different keys to be compared to one another.

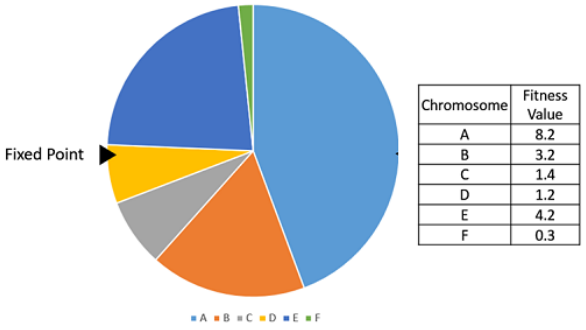
Selection

Tournament Selection

One of the selection methods used in this Genetic Algorithm was tournament selection. The size t of the tournament is a parameter which must be defined for this method, but in this implementation a size of t = 20 was used. This selection method is performed by choosing t number of individuals uniformly randomly from the population to compete against each other. The winner of the tournament is selected based on a simple formula. The formula relies on p which is a parameter which must be defined for this method. The best (or most fit) individual is selected as the winner with probability p. The second best individual is chosen with probability p*(1-p), and the third best with probability p*(1-p)*(1-p) and so on. In this implementation p = .75.

Fitness Proportional Selection

Fitness proportional selection is performed by transforming the fitness' of the population into an accumulated normalized list, where the final value of the list must equal 1. The resulting values will represent 'slices of a pie' where each slice has a probability of being selected as a parent with a probability proportional to its fitness. Individuals with a larger fitness will occupy a larger percentage of the 'pie' and will therefore have a much larger probability of being selected as a parent and will thus have a larger chance of passing its genes to the next generation.



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Crossover

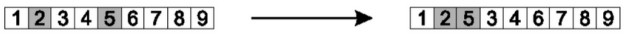
Position based crossover was used in this implementation of the Genetic Algorithm. Position based crossover works by selecting two parents (which were chosen by the selection method) and choosing up to k points for crossover. It then copies the values at those k points from the first parent into the same positions in the first child. Crossover then fills in the missing values in the first child with the values of the second parent in the order in which they appear, all while skipping over any values that were contained in the k-length substring copied from the first parent. The same is done to generate the second child except the k values copied will be from the second parent, and the missing values are filled in by the first parent. In this implementation k = 5.

This crossover method is used because it is imperative for the individuals to have a distinct set of genes, as any repeating letters in a key would make the problem unsolvable.

Mutation

Insert Mutation

One of the methods used for mutation was insert mutation. Insert mutation is performed by picking two unique and random indexes in an individual, and inserting the value of the second index directly after the value at the first index and shifting everything else back one. This preserves most of the order and the adjacency information within the individual and thus is not a very drastic mutation.



Inverse Mutation

Another mutation method used was inverse mutation. This method is performed by picking two indexes within the individual and reversing the substring in between them. This mutation preserves most of the adjacency information but is very disruptive to the order of information, unlike

insert mutation.

Swap Mutation

The final mutation method is swap mutation. This type of mutation is performed by picking two indexes within an individual and swapping the values at those locations. This type of mutation preserves most adjacency information and is not too disruptive to the order of the information.

Elitism

Elitism involves copying a certain percentage of the fittest individuals from each generation into the next generation without being subject to crossover and mutation. This percentage e = 20% in this implementation. Elitism played an important role in our project because without elitism we found that the average fitness of our population would not improve until it was implemented. It was at this point that we noticed a drastic improvement to the average fitness of our population from generation to generation.

Relevant Work

This project was created with inspiration from Jason Brownbridge from the Computer Science Department at the University of Cape Town. This project was designed to replicate the results detailed in the report mentioned above. The report can be found at: https://people.cs.uct.ac.za/~jkenwood/JasonBrownbridge.pdf . We used Brownbridge's report for details about: the default parameters, mutation method, crossover method, selection method, elitism and fitness. We implemented elitism, the fitness function and crossover the same way detailed in the essay, and used them consistently throughout all of our experimentation. We strayed away and tested different: mutation methods, selection methods and parameters. These changes and the results obtained will be discussed further in the Results section of the report. Brownbridge's report detailed much of what this project has become, and credit must be given where it is due.

Genetic Algorithm

Now that all of the methods have been described, it is important to note the ordering of these methods in the genetic algorithm. The algorithm proceeds as follows:
Repeat until convergence or until the max number of generations is reached:

- Bigrams are counted for each individual in the generation.
-Fitness scores for each individual are computed.
-Repeat until the next generation is filled:
  -Elitism
  -Selection
  -Crossover
  -Mutation

Results

Thorough testing was performed using different mutation methods, selection methods and parameters in order to amalgamate results for each different method of mutation, method of selection and parameter. The methods detailed in the Brownbridge report, and our default methods are swap mutation and tournament selection. The default parameters are as follows:

Population Size: 500
Tournament Size: 20
Tournament Winner Probability: 0.75
Probability of Crossover: 0.65
Probability of Mutation: 0.20
Percent Elitism: 15%
Crossover Points: 5
Maximum Number of Generations: 1000

The results of running the algorithm are evaluated based on how quick they converge, their maximum fitness score, the percent similarity between the highest fitness individual and the encryption key, the run time, and the percent of the final population made up of the most commonly occurring individual. The last statistic, "the percent of the final population made up of the most commonly occurring individual", is used to gauge the diversity of the population at the end of the algorithm.

Selection Methods

The two selection methods detailed in the Selection section of the report were used for testing. With all other methods and parameters kept at default, we tested the selection methods by running the algorithm using fitness proportional selection 5 times, and using tournament selection 5 times. The results are in the tables below. There are upsides and downsides to using both fitness proportional selection and tournament selection. Fitness proportional selection got 100% of the key correct more often than tournament selection. This would seem to be an advantage of FPS, but the fitness score for having 2 of the letters swapped, and having 24 of the 26 or around 92% of the key correct produces the same fitness score as a 100% correct key. FPS seems to be more susceptible to converging at local maxima, which is evident from it converging at a max fitness around half of the optimal fitness, and with only 19.2307% of the key correctly arranged. Tournament

Selection is more reliable and produced 80% or more correct individuals in each of the 5 trials. Tournament selection seemed to take on average more generations to converge however with its shortest run being 182 generations, and its longest run being 286 generations. Contrast this with FPS which found a perfect key in 136 generations and had a longest run of 257 generations.

Table I. Results for Fitness Proportional Selection

	Test 1	Test 2	Test 3	Test 4	Test 5
Generation Converged	218	183	136	224	257
Max Fitness Score	4632.4753	2924.1538	4632.4753	4632.4753	4632.4753
Percent of key correct	92.3076%	19.2307%	100%	100%	92.3076%
Run Time (seconds)	205.5907	172.3271	137.494	210.835	239.8117
Percent of Final Population is mode	31.19 %	31.79%	41.883%	35.437%	43.25955%

Table II. Results for Tournament Selection

	Test 1	Test 2	Test 3	Test 4	Test 5
Generation Converged	245	286	185	182	189
Max Fitness Score	4561.0668	4632.4753	4632.4753	4632.4753	4632.4753
Percent of key correct	84.6153%	100%	92.3076%	92.3076%	92.3076%
Run Time (seconds)	229.6500	261.5284	176.6877	174.5143	182.6627
Percent of Final Population is mode	43.2865%	32.0641%	51.7034%	55.1102%	49.2985%

Mutation Methods

All the mutation methods were tested to see how they affected the results. Swap mutation was clearly superior to both insert and inverse mutation. Insert and inverse seemed to be too disruptive and not preserve the order of the chromosomes well enough to meaningfully produce diverse individuals. Instead they both scrambled individuals and didn't allow for high fitness individuals to mate enough times to increase the fitness to the optimal level.

Table III. Results for Insert Mutation

	Test 1	Test 2	Test 3
Generation Converged	56	19	36
Max Fitness Score	2432.9887	2032.4248	2335.6606
Percent of key correct	7.6923%	3.8461 %	34.6153%
Run Time (seconds)	63.9440	34.4679	48.9072
Percent of Final Population is mode	99.7995%	99.7995%	99.7995 %

Table IV. Results for Inverse Mutation

	Test 1	Test 2	Test 3
Generation Converged	53	35	36
Max Fitness Score	2490.2032	2407.8029	2009.1636
Percent of key correct	19.2307%	26.9230%	15.3846%
Run Time (seconds)	63.8599	47.1122	48.0228
Percent of Final Population is mode	99.7995%	99.7995%	99.7995%

Elitism

Without elitism the generations converge to a suboptimal individual and fitness, because no high fitness individuals are preserved for the next generation. This opens up high fitness individuals to crossing over with a low-fitness individual who also won a tournament, and this can cause the fitness to stagnate. Without elitism, the maximum fitness from generation to generation can decrease, whereas with elitism this wouldn't be possible. The lack of elitism causes the generations to not converge quickly, if at all, also. The lack of preserving high fitness individuals means there is no constant from one generation to the next, so there is no way for the generations to converge. The lack of persistence is seen in the table. by the fact that the solution does not converge in 300 generations and the population doesn't have a single repeat of an individual in the entire population. This reaffirms the sentiment that without elitism the population will not converge, and will be constantly changing.

Implications

The results of this experiment have implications on the ways in which different methods of selection and mutation affect code breaking. It is clear that the order of the chromosomes in an individual is very important, because changing even a single location of a chromosome in an individual can have drastic effects on the fitness of the individual. Due to the importance of order, the type of mutation used is very important. This is why we tested inverse and insert mutation only to realize they were too disruptive and didn't allow for enough persistence in the population from one generation to the next. This revelation can be extended to any use of a genetic algorithm with individuals who are very sensitive to change. Our results on FPS and Tournament Selection are important, because FPS proved to be a very effective selection method. Brownbridge used tournament selection exclusively, but we showed FPS can be effective at selection and rival tournament selection.

Conclusion

The results of our experiment and implementation reaffirm the effectiveness of Brownbridge and his approach, but also validate the use of FPS as a selection method, and remove insert and inverse mutation from the picture in any problem with individuals very sensitive to relatively small changes. We also showed the effectiveness of elitism and its importance to convergence and to the persistence of high fitness scores, as well as the propagation of high fitness traits throughout the population.