A Dominant Gene Genetic Algorithm for a Substitution Cipher in Cryptography

Derrick Erickson and Michael Hausman

***Abstract*—This study is about breaking the substitution cipher in cryptography. The substitution cipher replaces every letter in a document with a different letter. This makes the document unreadable unless one can find the key to decrypt the document. A genetic algorithm is a way to combine the Darwin theory and genetics to converge on the solution after many generations or iterations. This genetic algorithm will decode the message by using unigram, bigram, trigram, and four-gram statistics to build the key. These statistics will not only determine the cost of a chromosome, how "good" a solution is, but will determine whether or not a gene is dominant. The mating function focuses on building a chromosome with the dominant genes. All dominant genes are passed to the child chromosomes to quickly converge on a solution. The mutation function searches "local" solutions by manipulating genes to make the overall solution better. After many generations, the encrypted message should look close to the original message.**

***Index Terms*—Bigram, Cipher, Convergence, Cryptography, Dominant Gene, Encrypted Message, Genetic Algorithm, Permutation Cipher, Recessive Gene, Trigram, Substitution Cipher.**

I. Introduction

Sometimes there are cases where when a message should only be viewable by the intended parties, and for this reason cryptography is used. For example, a commander needs to give attack plans to battalions in war. Only the battalion and the commander should be able to read the plans. Otherwise the enemy who confiscated the message could use the information to create a counter-attack or trap. A good cipher will take an someone a very long time to decrypt the information, if they ever succeed.

In this case, the substitution cipher is the cryptography technique in question. The substitution cipher is common in less important functions, such as crosswords, puzzles, and word games in the Sunday newspaper. Substitution is also a single step in many multi-step cryptography techniques used today, such as AES and DES. Breaking the substitution cipher is a method to gain insight into the inner workings of cryptography and its methodology. This study uses the Dominant Gene Genetic Algorithm to break the substitution cipher.

II. Substitution Cipher Details

The substitution cipher is very simple and has been used for centuries. The basic idea is to replace every letter or symbol with another letter or symbol in a manner that only the sender and receiver know. The receiver can decrypt the message and read it while anyone in between will look at a seemingly random set of letters. In this case, there are three main steps to this process.

1. Remove all non-letters from the message (spaces, punctuation, etc)
2. Create a character mapping for each letter
3. Use the character map to encrypt the message

The first step makes it harder to decrypt the message. This makes it very hard to detect words and sentences in the message. The second step is the key to the substitution cipher. This part is what needs to be the "secret" that only the sender and receivers know. The key tells us which letters in the encrypted message corresponds to which letters in the original message. For example, “a” in the original message may be a “q” in the encrypted message. The third step is simply to use that key to encrypt the original message into encrypted message.

Table 1: An example of the substitution cipher.

Key
Original Positions: a,b,c,d,e,f,g,h,i,j,k,l,m,...,z
Encrypted Positions: q,z,y,m,h,j,b,x,o,a,f,i,p,...,t

Example:
Original Message: `iamasubstitutioncipher`
Encrypted Message: `oqpqgvzglolvlodkyonxhw`

III. Substitution Cipher Research

There are multiple forms of the substitution cipher. Some remove spaces and punctuation before encrypting the text [8][9]. Other methods do not remove the spaces from the text [4][5][7]. Some substitution ciphers use a known word and create the key from that word. This was a common method in the Roman times. Another version is to replace the letters with symbols instead of substituting letters [10]. In all of these cases, the overall task is the same. The idea is to find the map of characters to decrypt the encoded message.

IV. General Genetic Algorithm Description

The Genetic Algorithm is based on Darwin’s theory of evolution. The general idea is to start with a set of solutions, use that set of solutions to make better solutions, and only keep the best solutions. In Genetic Algorithms, “chromosomes” define all of the information necessary to define a solution. The individual pieces of the chromosome are “genes” that define a certain aspect of the solution. In this case, the chromosome defines the mapping of the original text

to the cipher text. Each gene represents a map for one letter. The generic process of creating "better" chromosomes, or solutions, with Genetic Algorithms is below.

1. Create the initial chromosome set, or generation
2. Combine chromosomes, or "mate the chromosomes," to create a new "child" generation
3. Measure the chromosomes and keep the best half all the solutions (some may be from original "parent" generation, others from the "child" generation)
4. Modify, or "mutate," some of the chromosomes to prevent an entire generation from evolving to the same chromosome
5. Repeat 2-4

V. Initial Generation Creation

In general, Genetic Algorithms randomly create the initial generation. There is nothing, however, which states that all Genetic Algorithms must work this way. One can theoretically use statistics or information and create some educated guesses for the initial generation. The reason behind creating random chromosomes comes from the fact that randomly creating chromosomes has a high probability of creating information that can evolve to any solution eventually. Therefore, the educated guesses are a portion, but not all, of the initial generation.

Before one can make an educated guess, one needs information or statistics to go off of. To do this, the author uses a custom gram program. This gram program reads the input text, which is the bible, and returns the top unigram, bigram, trigram, and four-grams. Then the author uses the same process on the encrypted text. The top grams in the encrypted message would be the same as the top grams in the statistics. For example, the top trigram in the encrypted text could be "xbw," which would represent "the" according to the statistics.

Table 2: Statistics used for initial chromosome generation

Gram	Score
e	1
t	0.686449195
o	0.648787116
…	…
th	200
he	185.2286915
an	88.3373248
…	…
the	300
and	112.6185938
you	73.82056059
…	…
will	400
nthe	389.8032989
…	…

The next question would be what proportions of the chromosomes use the statistics or are random. Two chromosomes that only use unigram statistics would be approximately or completely the same. The same is true for two chromosomes that use the bigram statistics. The bigram solutions, however, will not necessarily be the same as the unigram statistics. In most cases, however, "unigram" and "bigram" solutions are different. Therefore, the author uses the proportions below. The Genetic Algorithm uses a small proportion for each of the different gram statistics. A majority of the chromosomes are random give the Genetic Algorithm a chance to find any solution.

Table 3: Proportions for initial chromosome creation

Way to Create Chromosome	Percentage
Unigram Statistics	10%
Bigram Statistics	10%
Trigram Statistics	10%
Four-Gram Statistics	10%
Random	60%

VI. Cost Function

The purpose of the cost function is to be able to tell how good a solution is. It is a mathematical way of saying that one chromosome is better than another chromosome. This is useful in mating and mutation later. Also, this study uses a "Dominant Gene" Genetic Algorithm, so there has to be a way to tell which genes are dominant. The cost function will mathematically figure this out as well.

A. Cost Function Research Details

There are a variety of cost functions used by other studies in the past. Some use dictionaries to pick out words and add the number found [10][11]. The most common cost function uses gram statistics, but the implementation and statistics vary from author to author. Some use a large amount of grams while others only use a few. Some use only unigram and bi-grams while others use bi-grams and tri-grams [1][2][3][4][5][6][7][8][9]. The Dominant Gene Genetic Algorithm, however, needs a very large number of statistics to differentiate the genes from one another.

B. Dominant Gene Cost Function

The same gram program from earlier also creates the statistics for this cost function. Just like before, the gram program looks at the gram statistics from the input text, the bible. The gram program then returns the unigram statistics, along with the other gram statistics based on letter. This includes the top ten bigram statistics with a letter a, then the top ten bigram statistics with the letter b, and so on. This way, there is a way to look at less common letters that may not appear in the top statistics. Each statistic is proportional to the number of grams found in the input text. This way a unigram like "e" is worth much more than other unigrams like "q." Also, all of the bigram scores are multiplied by 2, the trigram scores by 3, and the four-gram scores by 4. Then lower order grams cannot outscore higher order grams very easily. For example, two mid-scoring bi-grams cannot override a high-scoring four-gram. A table with some sample values is below.

Table 4: Statistics used for measuring chromosomes and genes

Gram	Score
an	88.3373248
ea	53.84633505
ha	49.59365553
at	44.90440108
…	…
but	14.07204436
bec	10.39076377
heb	9.59472339
llb	9.51349478
…	…
nthe	389.8032989
dthe	373.8908
thel	368.8100909
…	…

The next question is how does the Dominant Gene Genetic Algorithm use these statistics? The first step would be to decrypt the encrypted message by using the key in the chromosome. Then any of the bigrams, trigrams, or four-grams from the table found in this text are added up. This will be the cost of the chromosome. The cost function calculates the genes in a similar manner. Every time one of the bigrams, trigrams, or four-grams is found, 1 is added to each letter in the gram. Then after all of the text is searched, the total score for each letter is divided by the unigram score. This way the correct number of "e" occurrences does not overpower the correct number of one of the less common letters like "q" or "x." Realistically, the wrong grams could make the lesser grams very "dominant" with this technique, but the overall cost of the chromosome would not be very big. The Dominant Gene Genetic Algorithm would eventually throw out this wrong chromosome. Therefore, the gain of detecting more dominant genes overrides the risk of making false dominant genes.

VII. Mating Function

For Genetic Algorithms, the purpose of the mating process is to create new "child" chromosomes from two existing "parent" chromosomes. These children will be similar to the parent chromosomes and potentially closer to the "real" solution. This process creates new solutions using all the available information without eliminating the original results. Therefore, the Genetic Algorithm can evolve a solution towards the "real" solution over several generations.

A. Mating Research Details

Other studies use a variety of mating techniques. One of the most common techniques is crossover. There are two methods of crossover, single-point and multi-point [5][7][9]. The original intention of this study, however, is to use the Dominant Gene Genetic Algorithm. The authors of this paper have broken the transposition cipher with a Dominant Gene Genetic Algorithm, and the authors want to see if the results are as successful when breaking the substitution cipher [1].

B. Dominant Gene Mating Function

Before mating chromosomes, there has to be a way to judge which chromosomes to mate. The cost function defines how good a chromosome is by giving better chromosomes a higher score. Since the goal is to make the highest solution possible, only the top half of the generation mates. The Dominant Gene Genetic Algorithm randomly separates these solutions into pairs. Then each "parent" chromosome pair mates to create two new "child" chromosomes. This will create the new generation.

The mating process is a Dominant Gene algorithm, which means that the mating process revolves around "dominant" genes. Also, each child takes after one parent more than the other. This process takes 4 steps, as the example below shows.

1. Find Dominant Genes
   Parent 1: q e t v p r …
   Gene Cost: 11 12 3 24 15 4 …
   Select the upper 1/3
   Dom. Genes: v, p, etc

   Parent 2: z v y g d i …
   Gene Cost: 25 15 5 10 8 14 …
   Select the upper 2/3
   Dom. Genes: z, v, i, g, etc
3. Place Dominant Genes Based on First Parent
   Child 1: * * * v p * …
   Child 2: z v * g * i …
4. Fill in Blanks from Second Parent
   Child 1: z * * v p i …
   Child 2: z v * g p i …
5. Fill in any Remaining Blanks from First Parent
   Child 1: z e t v p i …
   Child 2: z v y g p i …

The first step is to select the dominant genes from each parent. Note that the cost function calculates the score of the genes. The Dominant Gene Genetic Algorithm selects the top third genes from the first parent and selects the top two thirds from the second parent. The second step is to take the dominant genes from the first parent and place them into the child. The third step is to take the dominant genes from the second parent and place them into the child. Note that this second set of genes cannot override the first parent's genes and cannot introduce copies of any character. Otherwise this would create an invalid solution. The fourth step is to fill in any remaining gaps in the child's chromosomes. These genes are placed in the same order that they appear in the first parent.

VIII. Mutation Function

In nature, mutations within a chromosome unexpected, random errors introduced by mistake. These mutations can be very helpful or very harmful depending on the situation. For

Genetic Algorithms, these mutations alter the evolution of a generation. This can be good or bad. Mutating the best solutions often make those solutions worse according to the cost function. Mutations will also introduce new, sometimes valuable, information to a homogenous generation. Then the generation can continue to evolve instead of wasting time generating the same result over and over.

A. *Mutation Research Details*

Most of the research reveals that many people do not vary the mutations very much. For the most part, only one mutation is used, which is to randomly swap two characters. Other mutations exist, but most people found that this mutation is sufficient for their needs. [1][5][6][7][9]. Most of the other mutations available were very specific to the type of attack on the substitution cipher and are not explained here.

B. *Dominant Gene Mutation Function*

The Dominant Gene Genetic Algorithm mutates all chromosomes except the top chromosome every generation. This prevents the Dominant Gene Genetic Algorithm from losing the "top" solution in a harmful mutation. In this case, the mutation is the only way to adjust the recessive genes to help improve the cost of a solution. The mating process focuses on the dominant genes while the mutation function helps modify the recessive genes.

The Dominant Gene Genetic Algorithm mutates the chromosome once. If the chromosome is better than before the mutation, the mutation process stops for that chromosome. Otherwise the solution goes through another mutation. This way a chromosome either improves or has enough of a change to potentially create a better solution in later generations. One swap often not enough to improve a solution.

The mutation in this study randomly takes two genes within a chromosome and switches them. This mutation is blind because it does not choose between dominant and recessive genes. This has the potential to help in two ways. The mutation could add to an existing dominant gene or it can modify a "wrong" dominant gene. Otherwise that "wrong" dominant gene will always exist. An example is below.

Original:	a	b	c	d	e	f	…	z
Mutated:	a	b	f	d	e	c	…	z
Swap Positions:	0	0	1	0	0	1	…	0

IX. RESULTS

A. *Measuring the Results*

In order to know how successful the Dominant Gene Genetic Algorithm is, one has to be able to measure the results. One way is to look at the number of letters correct in the key. Obviously, having the entire key correct would mean a success. It is not as easy to say how successful the Genetic Algorithm is when only a portion of the key is correct. For example, two solutions have 10 out of 26 letters in the key correct. The first key decrypts 80% of the cipher text correctly. The output is not completely correct, but it is "readable." The second solution, however, only decrypts 20% of the output text correctly. This solution is not "readable" by any means. The number of characters correct does not mean the output text will always be readable. Logically, the next step is to look at the percentage of text correct. This does not work either. It is possible to have 60% of the text correct with 5 or 15 letters in the key correct. Therefore, the success of the Dominant Gene Genetic Algorithm depends on both the number of letter correct in the key and the percentage of the cipher text correct. Unfortunately, this requires some judgment from the user. Some sample values are below.

Table 5: Deducted success examples for solutions

Number of Letters Correct	Percent of Cipher Text Correct	Deducted Success
26	100%	Perfect
21	100%	Near Perfect
15	95%	Very Good
16	85%	Very Good
16	60%	Good
10	80%	Good
5	60%	OK
10	40%	OK
3	30%	Bad
8	10%	Bad
0	0%	Fail

B. *Result Research Details*

The measured results from other papers are very inconsistent for several reasons. Some measure success by the value returned from the cost function [6][7][9]. Others measure success with the time it took to get a solution [4][5]. Another common way to measure success is to return the number of characters correct [5][8]. Unfortunately, all of these results are with slightly different tests. Some of the tests include spaces, which makes it much easier to detect words. Also, many people use different numbers of chromosomes and generations. One test that uses 10 chromosomes over 10000 generations is NOT the same as 1000 chromosomes over 100 generations. The authors even found a paper that claims to use the same techniques as other people and cannot get the same results! [9] This inconsistency makes it very hard to make a valid evaluation against other results. Also, this is an experiment to see how well a dominant gene genetic algorithm works for the substitution cipher. Therefore, the authors use the same setup from previous experiments against a different cipher [1].

C. *Dominant Gene Genetic Algorithm Results*

The setup of the Dominant Gene Genetic Algorithm is a key factor in obtaining good results. In this case, the algorithm evolves 100 chromosomes across 50 generations. One hundred chromosomes gives a big enough evolution tree that it is manageable and gives enough data points to work towards many possible solutions. Fifty generations gives enough generations to evolve a good solution. This process is run through 50 iterations to make the results below.

The first step, as outlined earlier, is to find how many characters are correct in the key. According to the data (shown below), the number of correct characters on average goes up as more cipher text is used. The highest number of correct character mappings was 15.82, which is only 10 letters off of the full 26.

Table 6: Average Number of Letters Correct

Cipher Text Length (Letters)	Average Number of Key Letters Correct
200	7.9
400	10.68
600	14.06
800	11.58
1000	15.82

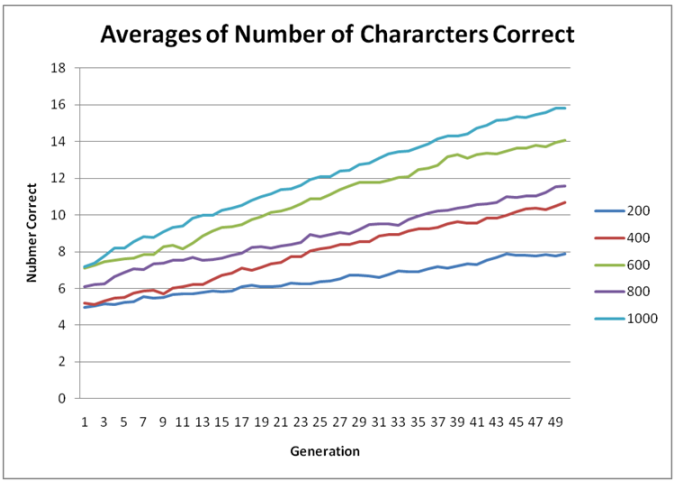

Figure 1: Average Number of Letters Correct

The second step is to find the percentage of cipher text correct. The results are below. Just like before, the percentages generally increase as the amount of cipher text supplied increases. In the case of 15.82 characters mapped correctly, 86.77% of the text was deciphered correctly. This is a "very good" solution according to Table 5.

Table 7: Percentage of Cipher Text Correct

Cipher Text Length (Letters)	Average Percent of Cipher Text Correct (%)
200	54.67
400	66.7
600	79.76
800	64.9
1000	86.77

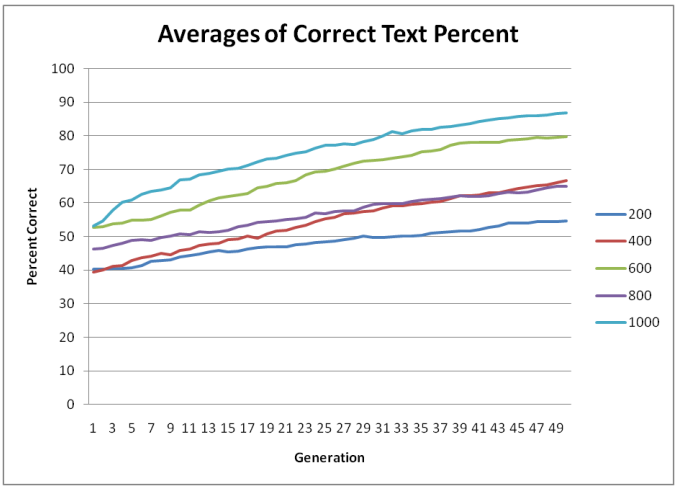

Figure 2: Percentage of Cipher Text Correct

When looking at the above data, the 800 cipher text results were below the 400, 600, and 1000 cipher text but above 200. This breaks the general trend noted earlier. One possible cause is that somewhere in the cipher text a sentence does not follow the statistical properties of the common text as well. This sentence confused the Dominant Gene Genetic Algorithm enough to lower the success rate. The 1000 cipher text, however, had enough surrounding text to mitigate the problems seen in the 800 cipher text. This indicates that using more text has a good probability of overcoming these statistical anomalies.

Just because the 800 cipher text example has this rouge sentence does not mean the results are always bad. Sometimes it will progress near the minimum path and other times near the maximum path. The potential for every solution is the same, but the difference is how often it follows a specific path. To expand on this, the minimum and maximum percentages are below.

Table 8: More data for percent of cipher text correct

Cipher Text Length (Letters)	Percent of Cipher Text Correct		
	Average (%)	Minimum (%)	Maximum(%)
200	54.67	35	86
400	66.7	34.25	92.5
600	79.76	44.3	93.66
800	64.9	34.12	95.37
1000	86.77	60.4	94.3

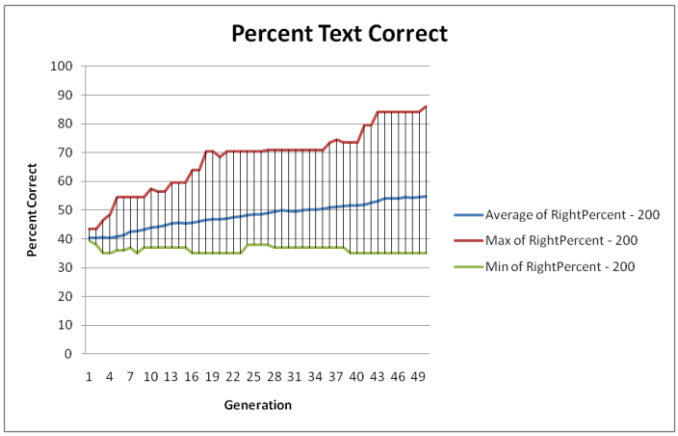

Figure 3: Range for Percentage of 200 Cipher Text Correct

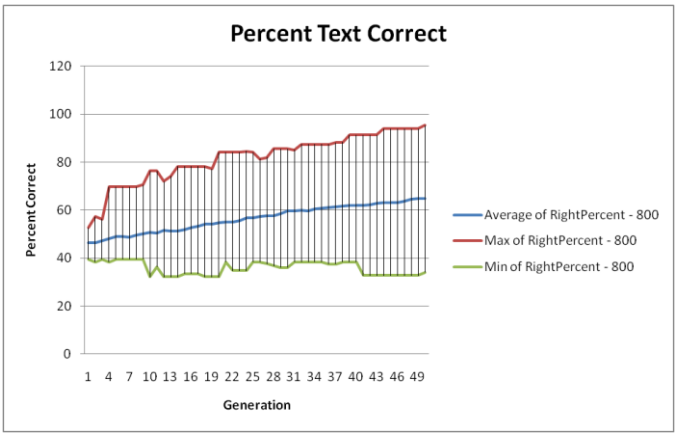

Figure 6: Range for Percentage of 800 Cipher Text Correct

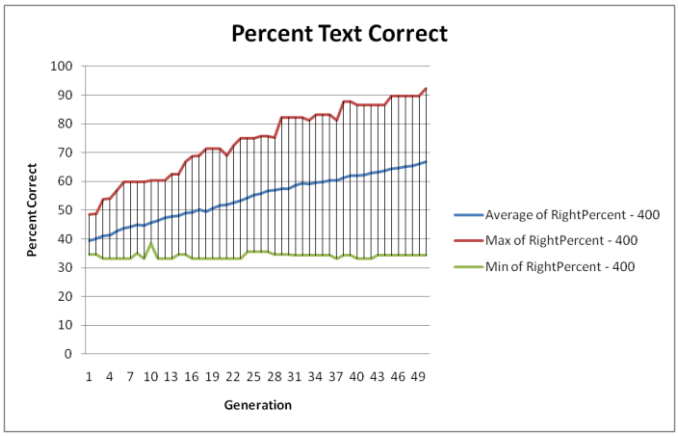

Figure 4: Range for Percentage of 400 Cipher Text Correct

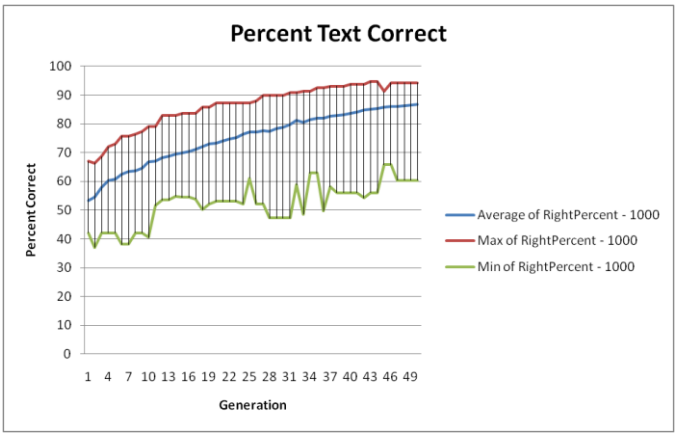

Figure 7: Range for Percentage of 1000 Cipher Text Correct

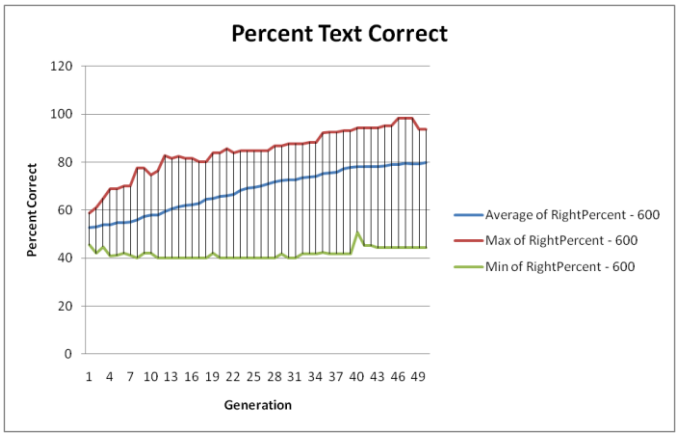

Figure 5: Range for Percentage of 600 Cipher Text Correct

The maximum potential of all groups of cipher text is approximately the same, given a few percentage points. The minimum, however, is drastically different. With the exception of the 800 cipher text example, more cipher text increases the minimum percentages. Also, the average drifts closer to the maximum as the cipher text increases. This indicates that more cipher text increases the chances for the Dominant Gene Genetic Algorithm to find common grams. Rogue sentences, like in the 800 cipher text example, bring the average and minimum percentages down and increase the range of possible results. To solve this, add even more cipher text and the surrounding text will bring the results up again, like the 1000 cipher text did. This indicates that there is a potential to detect rogue sentences. Rogue sentences will decrease the results and will increase the range of possibilities. If the results show this, add cipher text until this trend stops.

Given that sometimes the genetic algorithm will follow the minimum path, it helps to run the program multiple times. Then the user can pick the highest solution and have a better chance at obtaining a key that decrypts a message correctly. It also gives a user an indication on whether a rogue sentence exists in the encrypted cipher text.

X. Conclusion

The Dominant Gene Genetic Algorithm works very well for breaking the substitution cipher. The deciphered text is good

enough that a human can read the results. There will be a few spelling mistakes, but that is easily corrected and deciphered correctly. The mating process keeps the dominant genes from disappearing. This is good because dominant genes have a high probability of being correct. The mutation function modifies the lower genes and gives them the potential to become a dominant gene. This also helps to correct dominant genes that may have been wrong in the first place. In conclusion, the combination of these two ideas allows for a well balanced genetic algorithm that can produce an acceptable solution.

REFERENCES

[1] Hausman, Michael, "A Dominant Gene Genetic Algorithm for a Transposition Cipher in Cryptography," University of Colorado at Colorado Springs, May 2009.

[2] Russell, M.D.; Clark, J.A.; Stepney, S., "Making the most of two heuristics: breaking transposition ciphers with ants," Evolutionary Computation, 2003. CEC '03:. International Conference on Evolutionary Computation, vol.4, no., pp. 2653-2658 Vol.4, 8-12 Dec. 2003

[3] Li, Ho Yean; Samsudin, Azman; Belaton, Bahari, "Heuristic Cryptanalysis of Classical and Modern Ciphers," IN: Joint IEEE Malaysia International Conference on Communications and IEEE International Conference on Networks, 16-18 November 2005, Kuala Lumpur, pp. 710-715.

[4] Clark, A. and Dawson, E., "Optimisation Heuristics for the Automated Cryptanalysis of Classical Ciphers," Journal of Combinatorial Mathematics and Combinatorial Computing. v28. 63-86.

[5] Clark, A., "Modern optimization algorithms for cryptanalysis," Intelligent Information Systems,1994. Proceedings of the 1994 Second Australian and New Zealand Conference, vol., no., pp.258-262, 29 Nov-2 Dec 1994

[6] Spillman, Richard; Janssen, Mark; Nelson, Bob; Kepner, Martin, 'USE OF A GENETIC ALGORITHM INTHE CRYPTANALYSIS OF SIMPLE SUBSTITUTION CIPHERS',Cryptologia,17:1,31 — 44

[7] Brownridge, Jason, "Decrypting Substitution Ciphers with Genetic Algorithms," University of Cape Town, March 2007.

[8] Verma, A. K., Dave, M., and Joshi, R. C., "Genetic algorithm and tabu search attack on the mono-alphabetic substitution cipher i adhoc networks," Journal of Computer Science 3, 134-137, 2007.

[9] Gester, Joe, "Solving substitution ciphers with genetics algorithm." http://www.cs.rochester.edu/u/brown/Crypto/studprojs/SubstGen.pdf, December 2003.

[10] Oranchak, D., "Evolutionary Algorithm for Decryption of Monoalphabetic Homophonic Substitution Ciphers Encoded as Constraint Satisfaction Problems," Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, July 2008.

[11] Morelli, Ralph; Walde Ralph., "A Word-Based Genetic Algorithm for Cryptoanalysis of Short Cryptograms," FLAIRS, pp. 229-233, 2003.