Applying Genetic Algorithms for Searching Key-Space of Polyalphabetic Substitution Ciphers

Ragheb Toemeh[1] and Subbanagounder Arumugam[2]
[1]Department of Computer Science and Engineering, Government College of Technology, India
[2]Directorate of Technical Education, India

Abstract: *In this paper the Cryptanalysis of polyalphabetic by applying Genetic algorithm is presented. The applicability of Genetic algorithms for searching the key space of encryption scheme is studied. In Vigenere cipher, guessing the key size is done by applying Genetic Algorithm. The frequency analysis is used as an essential factor in objective function.*

Keywords: *Polyalphabetic cipher, Vigenere cipher, genetic algorithm*.

Received May 27, 2006; Accepted August 5, 2006

1. Introduction

Cryptanalysis is the process of attempting to recover the plaintext and /or key from a ciphertext. In the Brute Force attack the attacker tries every possible key on a piece of ciphertext until an intelligible translation into plaintext is obtained; it has the disadvantage of high computational complexity. In order to overcome this drawback, the optimization heuristics techniques like Genetic Algorithm (GA) are used. The possibility of using a random type search to break a cipher is explored. The focus of this work is on the use of a GA to conduct a directed random search of a key space and to guess the key size. In 1863, a prussian major named Kasiski proposed a method for breaking a Vigenere cipher that consisted of finding the length of the keyword and then dividing the message into many simple substitution cryptograms [9]. In this paper, to find the key length, new method is used; this method is based on the use of GA and frequency analysis. One of the main problems with simple substitution ciphers is that they are so vulnerable to frequency analysis. Given a sufficiently large ciphertext, it can easily be broken by mapping the frequency of its letters to the know frequencies of English text. Therefore, to make ciphers more secure, cryptographers have long been interested in developing enciphering techniques that are immune to frequency analysis. One of the most common approaches is to suppress the normal frequency data by using more than one alphabet to encrypt the message. A polyalphabetic substitution cipher involves the use of two or more cipher alphabets. Instead of there being a one to-one relationship between each letter and its substitute, there is a one-to-many relationship between each letter and its substitutes. To give a concrete example of redundancy in English, single character frequencies (including the apostrophe) are shown for a sample corpus in Figure 1.

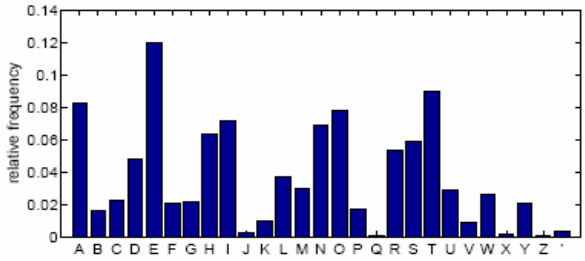

Figure 1. Relative Character Frequencies from *Great Expectations* (1860-1861).

2. Genetic Algorithms

A GA is general method of solving problems to which no satisfactory, obvious, solution exists. It is based on the idea of emulating the evolution of a species in nature and so the various components of the algorithm are roughly analogous to aspects of natural evolution. Common mathematical tasks amenable to genetic solutions include computing a curve to fit a set of data or approximating Nondeterministic Polynomial (NP) problems. Often these operators consist of flipping a single random bit of one individual or swapping two randomly selected substrings from a pair of parents to generate a new child. To simulate Darwinian survival of the fittest some representation of the fitness of the individuals must be generated. GA is applied in four steps:

1. Initialize algorithm variables: G the maximum number of generations to consider, M the solution pool size and any other problem dependent variables.
2. Generate an initial solution pool containing M candidate solutions.
3. For g iteration, using the current pool:

a. Select a breeding pool from the current solution pool and make pairings of parents.
b. For each parental pairing, generate a pair of children using a suitable mating function.
c. Apply a mutation operation to each of the newly created children.
d. Evaluate the fitness function for each of the children.
e. Based on the fitness of each of the children and the fitness of each of the solutions in the current pool, decide which solutions will be placed in the new solution pool. Copy the chosen solutions into the new solution pool.
f. Replace the current solution pool with the new one. So, the new solution pool becomes the current one.

4. Choose the fittest solution of the final generation as the best solution.

3. Polyalphabetic Substitution Cipher

The polyalphabetic substitution cipher is a simple extension of the monoalphabetic one. The difference is that the message is broken into blocks of equal length, say *B*, and then each position in the block (1... *B*) is encrypted (or decrypted) using a different simple substitution cipher key. The block size *(B)* is often referred to as the period of the cipher. An example of a polyalphabetic substitution cipher is shown in Table 1. *B* is chosen to be three, and Table 1 gives an example key and shows the corresponding encryption [4].

Table 1. Example of the polyalphabetic substitution cipher key and encryption process.

KEY:
Plaintext: ABCDEFGHIJKLMNOPQRSTUVWXYZ_ Ciphertext: ND_WIEURYTLAKSJQHFGMZPXOBCV (Position 1) LP_MKONJIBHUVGYCFTXDRZSEAWQ (Position 2) GFTYHBVCDRUJNXSEIKM_ZAOLWQP (Position 3)
ENCRYPTION:
Position: 12312312312 Plaintext: HOW_ARE_YOU Ciphertext RYOVLKIQWJR

The decryption process is reversal of the encryption. The polyalphabetic substitution cipher is somewhat more difficult to cryptanalyse than the simple substitution cipher because of the independent keys used to encrypt successive characters in the plaintext, but it is still relatively simple to cryptanalyse the polyalphabetic substitution cipher based on the *n*-gram statistics of the plaintext language. So, despite the monoalphabetic substitution cipher where every bigram (for example _A) is mapped to the same encrypted bigram each time, this is not the case for the polyalphabetic substitution cipher, where the encrypted value of a bigram is dependent upon two factors: the individual key values and the position of the characters within the block. The algorithm which proposed in [4] is re-implemented.

4. The Vigenere Tableau

The Vigenere Cipher, proposed by Blaise de Vigenere from the court of Henry III of France in the sixteenth century, is a polyalphabetic substitution based on the following tableau:

Table 2. Vigenere tableau.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

For example, suppose we wish to encipher the plaintext message: “TO BE OR NOT TO BE THAT IS THE QUESTION", using the keyword SUBSTITUTION. We begin by writing the keyword, repeated as many times as necessary, above the plaintext message. To derive the ciphertext using the tableau, for each letter in the plaintext, one finds the intersection of the row given by the corresponding keyword letter and the column given by the plaintext letter itself to pick out the ciphertext letter.

*Keyword:
SUBSTITUTIONSUBSTITUTIONSUBSTI

*Plaintext:
TOBEORNOTTOBETHATISTHEQUESTION

*Ciphertext:
LICWHZGIMBCOWNISMQLNAMEHWMUAHVG

Decipherment of an encrypted message is equally straightforward. One writes the keyword repeatedly above the message:

*Keyword:
SUBSTITUTIONSUBSTITUTIONSUBSTI

*Ciphertext:
LICWHZGIMBCOWNISMQLNAMEHWMUAHVG

*Plaintext:
TOBEORNOTTOBETHATISTHEQUESTION

This time one uses the keyword letter to pick a column of the table and then traces down the column to the row containing the ciphertext letter. The index of that row is the plaintext letter.

5. Proposed Algorithm for Vigenere Cipher

The following is an outline of proposed algorithm:

1. Input to the algorithm the cipher text, the key size and relative character frequencies.
2. Initialize the algorithm parameters: maximum number of iterations (M).
3. Generate 10 keys randomly each one is having the same known key length.
4. Decrypt the ciphertext by the 10 generated keys.
5. Calculate the suitability of each key from every decrypted text using the formula.
6. Sort the keys based on the increased fitness values.
7. for 1 to (M) do:
   a. Choose 5 pairs from 10 keys.
   b. For 1 to (5 pairs) do
      i. Apply crossover to get children
      ii. Generate random number from 2 to (key size -1)
      iii. Swap the parts of parents as example:
         Parent 1 sungti | hutior
         Parent 2 subdti | dution
         Child1 sungtidution
         Child 2 subdtihutior
      iv. Generate random position number between (1 to key size) for each child and mutate the letter in that position.
   c. Decrypt cipher text by 20 keys.
   d. Calculate the suitability of each key.
   e. Sort the 20 keys based on increased fitness values.
   f. Choose best 10 keys.
   g. go to 7.
8. Output is the best solution.

This algorithm is illustrated by the following example:
the 10 random keys are listed as:

AAIAJFNFSYHL
CSOBEVRTVYFL
YVYVLPRPOSCK
DRMMOIWGEKSE
NQQAUGLKJPPH
EUHIKGKBMZAK
BBGZJKFFFGTX
XTLNADLMPGCS
QXWBONQUGFGI
ODPAYFVQSTOO

After one generation the output is:

EBHIKHKUMZPG
ODPAYFVQSTOO
DRMMOIWGEKSE
EUHIKGKBMZAK
OXWBONQSQTQO
YVYEOGWIVKSE
BBGZJKFFFGTX
ZAOBEVRTVYFA
NQQAUGLKJPPH
AQQAPGLKJUNK
XTMNADLPLGTX
AAIAJFNFSYHL
QXWBONQUGFGI
CSFAJISFNYHL
BBGZSKFJFGCF
UDPAYVFOGFGI
DRPMLMSPORCK
XTLNADLMPGCS
YVYVLPRPOSCK
CSOBEVRTVYFL

After 30 generations the output is:

BUBSTHXFHIOO
JUBSTHXFCIOO
HIBSTQFUTIBN
HUBSFHXHTIOO
JUBSFHXHTIOO

After 100 generations output is:

SUBSTIUUTBON
SUBSTUBUTION
SUBSTUBUTION
SUBSTUDUTION

The final solution is (SUBSTITUUTBON) for ciphertext, which is having 1500 bytes size for 100 generations.

6. Guessing the Length of Key in Vigenere Cipher

To find the key length, new method is used; this method is suitable for the text which is having the size more than 500 bytes, because the main idea in this method is to employ frequency analysis. GA is applied here to find the key length. The first proposal key length will be chosen two and GA operations are applied for small population size and small number of generation., Fitness value is saved for next generation and key length must be increased to three, again fitness is saved and compared with previous key length; if it is better, the new will be taking for next generation, the method is still working till any key size. Best solution is expected to be the key length; this number will be submitted to next stage to get most of correct key letters. If the decrypted text is not readable, guessing the key length mater should be continued more than the key length itself.

7. Implementing the Attack for Vigenere Cipher

The attack is implemented by generating 10 independent keys to represent the target key. The first generation is generated randomly using a

simple uniform random number generator. The fitness value is incremented and finally normalized to the number of pairs. The GA then goes in the normal way to generate new generations. The algorithm is terminated based on the criteria described earlier. The algorithm has been implemented to get fitness; essentially the attack shall continue upward to get the best key. These functions are used in the code:

void Encrypt ()
This function performs encryption.

void Decrypt ()
This function performs decryption taking input as key.

void Keygen ()
This function creates the initial population it will generate *n* keys randomly.

void Getfitness ()
This function measures fitness of a particular chromosome in the population set indexed by its position in the population.

void Sorting()
This function is responsible for sorting population of chromosomes (The genetic material of an individual - represents the information about a possible solution to the given problem) based on fitness value.

void Crossover()
This function performs cross over between chromosomes and stores them in the new population set as indexed by pos*1*, pos*2*.

void Mutation ()
This function is responsible for mutate the new generated chromosomes.

8. Fitness Measure

The technique used to compare candidate keys is to compare *n*-gram statistics of the decrypted message with those of the language (which are assumed known). Equation 1 is a general formula used to determine the suitability of a proposed key (*k*).

$$C_k = \alpha \cdot \sum_{i \in A} \left| K^{u}_{(i)} - D^{u}_{(i)} \right| + \beta \cdot \sum_{i,j \in A} \left| K^{b}_{(i,j)} - D^{b}_{(i,j)} \right| + \gamma \cdot \sum_{i,j,k \in A} \left| K^{t}_{(i,j,k)} - D^{t}_{(i,j,k)} \right| \quad (1)$$

Here, *A* denotes the language alphabet (i.e., for English, [A. . . Z], *K* and *D* denote known language statistics and decrypted message statistics, respectively, and the indices *u*, *b* and *t* denote the unigram, bigram and trigram statistics, respectively. The values of α, β and γ allow assigning of different weights to each of the three *n*-gram types [4].

The ability of directing the random search process of the GA by selecting the fittest chromosomes among the population is the main characteristic of the algorithm. So the fitness function chosen is the main factor of the algorithm. This may be the main factor, selecting the fitness function, that GA were not applied the cryptanalysis problems. The fitness function relied on the language statistical characteristic to represent the fitness the key. For example, the letter "E" is the most common letter in English language, so the fitness of the key can be measured based on how likely it is going to give correct letter frequency in the de-ciphered text. This choice of fitness measure depends entirely on the language characteristics and hence these characteristics must be known and of course the language used itself.

9. Terminating Criteria

This is one of the classical problems of using GA in problem solving, that is when to terminate. The problem at hand really needs a terminating criterion that can be used to determine when it is enough for GA. The one chosen here is also one of the classical choices that are when the algorithm can generate no more better solutions for a number of generations.

10. Characteristics of Testing Environment

The algorithm is implemented using C++ and tested in machine with following configurations

- Intel P IV processor with speed 3 GHz.
- RAM 512 MB.

11. Result

The attack was implemented with a polyalphabetic substitution cipher with a block size of three. The average results for the polyalphabetic substitution cipher are given in Figure 2.

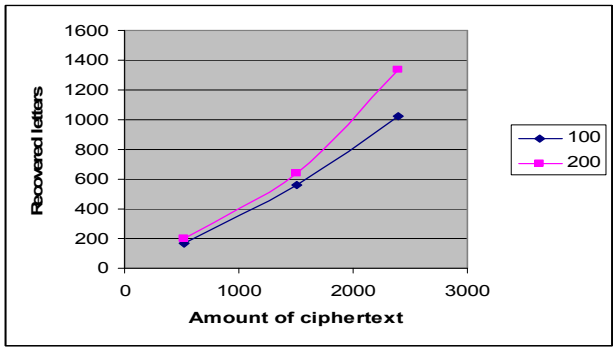

Figure 2. Known cipher text versus percent recovered letters.

The result is showing the effect of number of generations on the recovered letters. The result of guessing method of the key length is tabulated as comparison of fitness of some key lengths proposed, the population size is six, number of generation is 30, and text size is 1.5 kb, as in Table 3.

Table 3. Result of guessing key length.

Number of proposed key length	Fitness value
5	32.47
8	32.27
12	21.66
15	30.69

As the fitness value should be less, as Equation 1, the expected key length is 12. This number will be given to next stage in code to find the key word.

The result for vigenere cipher is tabulated as a comparison of performance for different cipher text lengths and these were encrypted using key 'SUBSTITUTION' as Table 4.

Table 4. Result of Vigenere cipher.

Cipher Text Size (in chars)	Solution Found	No of Generat -ions	Time (in sec)
500	subativusior	100	10
1000	substevusior	100	11
2000	substibution	100	13

12. Conclusion

The results in the implementation of Polyalphabetic Substitution Cipher are showing the effect of ciphertext size on recovered plaintext letters. The time in cryptanalysis of vigenere cipher is less by using GA. The method to find key length gives good result with compare with other methods.

References

[1] Albassal A. and Wahdan A., "Genetic Algorithm Cryptanalysis of the Basic Substitution Permutation Network," *in Proceedings of the 46th IEEE International Midwest Symposium on (MWSCAS'03)*, pp. 471-475, 2003.

[2] Bagnall A., "The Application of Genetic Algorithm in Cryptanalysis," *Master Degree Thesis*, School of Information Systems, University of East Anglia, 1996.

[3] Clark A. and Dawson E., "Optimisation Heuristics for the Automated Cryptanalysis of Classical Ciphers," *Journal of Combinatorial Mathematics and Combinatorial Computing*, vol. 28, pp. 63-86, 1998.

[4] Dimovski A. and Gligoroski D., "Attack on the Polyalphabetic Substitution Cipher Using a Parallel Genetic Algorithm," *Technical Report*, Swiss- Macedonian Scientific Cooperation Trought SCOPES Project, Ohrid, Macedonia, March 2003.

[5] Gester J., *Solving Substitution Ciphers with Genetics Algorithm*, www.cs.rechester.edu/ ubrown /Crypto/studprojs/SubstGen.pdf, 20th December 2003.

[6] http://www.trincoll.edu/depts/cpsc/cryptography/caesar.html, 5/8/2006.

[7] http://www.trincoll.edu/depts/cpsc/cryptography/index.html, 5/8/2006.

[8] http://www.trincoll.edu/depts/cpsc/cryptography/substitution.html, 5/8/2006.

[9] http://www.trincoll.edu/depts/cpsc/cryptography/vigenere.html, 5/8/2006.

[10] http://en.wikipedia.org/wiki/Letter_frequencies, 5/8/2006.

[11] Spillman R., Janssen M., Nelson B., and Kepner M., *Use of a Genetic Algorithm in the Cryptanalysts of Simple Substitution Ciphers*, Cryptologia, vol. 16, no. 1, pp. 31- 4, January 1993.

[12] Stallings W., *Cryptography and Network Security*: *Principles and Practices*, 3rd edition, Pearson Education, 2004.

Ragheb Toemeh received the BE in electronics engineering degree from Tishreen University, Lattakia, Syria in 1996 and the ME in computer science and engineering degree from PSG College of Technology, Coimbatore, Anna University in 2004. He is currently working toward the PhD degree at Government College of Technology, Coimbatore, Anna University, Chennai, India. His research interest area is in network security.

Subbanagounder Arumugam received the PhD degree in computer science and engineering from Anna University, Chennai in 1990. He also obtained his BE in electrical and electronics engineering and MSc in engineering, applied electronics degrees from PSG College of Technology, Coimbatore, University of Madras in 1971 and 1973, respectively. He is working in the Directorate of Technical Education, Government of Tamil nadu from 1974 at various positions from associate lecturer, lecturer, assistant professor, professor and principal. Presently, he is working as additional director of Technical Education. He has guided four PhD students and currently guiding ten PhD students. He has published 70 technical papers in international and national journals and conferences. His area of interest is including network security, biometrics, and neural networks.