Use Of Genetic Algorithm In The Cryptanalysis Of Transposition Ciphers

Ra`ad A. Muhajjar

Dept. Computer Science, College Science, University of Basrah

Abstract:

We consider the use of genetic algorithms (GAs) as powerful tools in the breaking of cryptographic systems. We show that GAs can greatly facilitate cryptanalysis by efficiently searching large keyspaces, and demonstrate their use with genetic cryptanalyst, an order-based GA for breaking a classic cryptographic system.

Keywords: Genetic algorithm, Cryptology, Cryptanalysis, Transposition ciphers, Key search.

Introduction

Cryptanalysis is the process of recovering the plaintext and\or key from a cipher. It may involve the use of just the ciphertext, portions of several ciphertext items, some plaintext, complicated mathematical procedures, computer algorithms, and usually some luck[1]. Many cryptographic systems have a finite key space and hence, are vulnerable to an exhaustive key search attack. Yet, these systems remain secure from such an attack because the size of the key space is such that the time and resources for a search are not available. In these cases, the cryptanalyst looks for trapdoors, exploitable regularities in either the cipher system or the language or both, combinations of plaintext and ciphertext, or anything else which may prove helpful in breaking the cipher[2]. To a cryptanalyst, the concept of a random search of the key space would be considered the last refuge of the hopelessly naïve. A random search through a finite but large key space is not usually an acceptable cryptanalysis tools[1].

In this paper, however, we explore the possibility of using a random type search to break a cipher. The focus of this work is on the use of genetic algorithm to conduct a directed random search of a key space. The ability to add direction to what seems to be a random

search is a feature of genetic algorithms which suggests that they may be able to conduct an efficient search of a large key space.

The goal of this initial work is to illustrate the feasibility of using genetic algorithms as a cryptanalysis tool. Hence, our algorithms are only applied to Monoalphabetic transposition ciphers in this first report in order to clearly present the fundamentals of the tool without becoming trapped in the details of a more complicated cipher.

Review Of Genetic Algorithms

Genetic Algorithm (GAs) typically have the following components[3,4]:

a) An initial population of random guesses "chromosomes" for the problem in hand, coded in a suitable form.
b) A fitness-rating system that assesses how effective each chromosome is at solving the problem.
c) A "survival of the fittest" filter, which is biased towards the selection of the better chromosomes.
d) A "crossover" operator, which takes some pairs of chromosomes surviving the filter, and randomly, combines elements of each to produce "offspring" with features of both "parent" guesses.
e) A mutation operator, a fixed, small mutation probability is set at the start of the algorithm. Bits in all the new strings (offspring) are then subject to change based on this mutation probability.

Steps (a) to (e) are used to generate new chromosomes which may be "fitter", i.e. better solutions, than those in the first, and the process is repeated over many generations. The end of the process is usually dictated either by CPU time constraints, or by the emergence of an "evolutionary niche" into which the chromosomes have settled. In either case, the fittest member of the final generation can be an optimal or near-optimal solution to the problem in hand.

A GA is typically run a number of times to probe different regions of the total solution space, and the best result from the ensemble of runs is taken. Researches have found that this is frequently, a better solution than that found by traditional algorithms such as Monte Carlo searching [5].

There are, however, two questions that have to be addressed before a problem is attacked using GAs. Most importantly, the problem must be such that a partially-correct chromosome has a higher fitness than a completely incorrect one. Without this, the GA has no way of recognizing the

emergence of a better chromosome and allowing it to breed.

Secondly, the problem must be coded in a form enabling the genetic operators outlined above to work. Most numerical optimization problems can be coded simply as binary strings made up of subsets representing numerical information about the parameters of a problem,(e.g shape and mechanical demands made of a structural member whose cost-effectiveness is to be optimized). Each initial guess is then coded as a chromosome of the form (010110110…0100) on which both crossover reproduction and mutation operators are applied.

However, problem such as scheduling, involving the optimization of a certain order of parameters, are more naturally coded as permutation chromosomes such as (10, 3,…, 8). The use of crossover and mutation operators on these is more subtle, as it is possible to create illegal chromosomes such as (10, 3,…, 3). However, various genetic operators suitable for order-based GAs have recently been developed and studied [3].

The Cryptanalytic Uses Of GAs

With their ability to efficiently search huge solution spaces, GAs would seem a natural candidate for use in cryptanalysis. Initialized with chromosomes in the form of guessed keys, the GA would use these keys in attempted decryptions of intercepted ciphertext, and assess the fitness of each attempt. The more successful keys would then be used to "breed" successive generations of fitter key combinations, until an optimal key allowing substantial decryption emerged. There are two classical techniques for encrypting, which are used singly or in combination in virtually every cryptographic algorithm[6]. Substitution involves the systematic replacement of characters in the text by a cipher character according to some algorithm. Being relatively easy to automate both mechanically and electronically, substitution has been very widely used in both government and commercial cryptographic systems. The algorithm can range from the movement of intricately-wired rotors (as in the Enigma machine) to non-linear feedback functions[7].

The second broad category of cryptographic algorithm is transposition[6], in which the relative order of characters making up the text is permuted according to some rule. This is less easy to automate, although the advent of microprocessors led to its being incorporated into a number of modern cryptographic systems, such as the Data Encryption Standard. Order-based GAs clearly have the potential to attack both substitution and transposition-based algorithms, used alone and in combination. To

take a simple example, a GA for breaking a random substitution cipher (total keyspace 26! $\approx 4*10^{26}$) might use chromosomes consisting of possible mappings of the plaintext characters into the ciphertext characters for an intercepted text. Specifically, the alphabetical sequence (A,B,C,D,E,…,Z) could be initially mapped to guessed decryption chromosomes (F,X,H,B,J,…,L),

(G,C,A,I,T,…,P),(W,X,Q,D,Z,…,U)etc .The text would then be decrypted using each mapping, and the fitness of each chromosome assessed by measuring the degree to which the resulting decrypted text matches English. One obvious way to do this would be to compare the frequencies with which the letters E,T,A etc appear in the decrypted text with their frequencies in plain English.

Similar ideas can be used against transposition ciphers, in which text is rearranged according to some order. However, there are a number of cryptographic algorithms that cannot be attacked using GAs, for reasons outlined earlier. For example, the Data Encryption Standard[1] uses a combination of substitution and transposition, but the way in which the key is used in DES ensures that partially correct keys are indistinguishable from totally incorrect ones. This will prevent a GA from capitalising on any partial success, and breeding more successful key sequence.

Nevertheless, a large number of cryptographic algorithms, including those underpinning many rotor-based systems, appear vulnerable to attack by GAs. Indeed, some of the handful of techniques published in the open literature for breaking rotor-based ciphers implicitly use some GA-like ideas [2].

To illustration detail the explicit use of a GA in cryptanalysis, we now describe the construction of Genetic Cryptanalyst, a genetic algorithm specifically designed to break classical transposition ciphers by finding the transposition sequence used. This particular class of algorithm was chosen because the automated breaking of such ciphers is notoriously difficult. In contrast to substitution ciphers, for which a number of statistical tools aiding automated breaking have been developed, cryptanalysis of transpositions is usually highly interventionist and demands some knowledge of the likely contents of the ciphertext to give an insight into the order of rearrangement used. In contrast, genetic cryptanalyst enables a known plaintext attack to be successfully made, based on only very portion of some plaintext of the ciphertext .

The Breaking Of Transposition Ciphers

The type of transposition cipher attacked by genetic cryptanalyst encrypts text according to the following classical two-stage algorithm[1,6]:

(a) K key of length N takes the form of a permutation of the integers 1 to N. The plaintext, of L characters, is written beneath the key to form a matrix N characters wide and at least L mod N characters deep.

(b) The text is then enciphered by reading it off in rows in the order dictated by the integers making up the key.

For example, suppose the key is 631254, and the text is (A TREE IS KNOWN BY ITS FRUIT). By (a) we obtain the following

6	3	1	2	5	4
A	▼	T	R	E	E
▼	I	S	▼	K	N
O	W	N	▼	B	Y
▼	I	T	S	▼	F
R	U	I	T	▼	▼

Reading off the text in rows in the order dictated by the key we then obtain the ciphertext:

TR▼EEAS▼INK▼N▼WYBOTSIF▼▼ITU▼▼R

NOTE: the symbol ▼ denote the empty symbol.

Decryption is achieved by writing down the key, calculating the lengths of each row of the matrix, writing the ciphertext back into it in the sequence dictated by the key, and reading across the rows.

The breaking of such a transposition cipher is typically also a two-stage process. First, the length of the transposition sequence (i.e. N) must be found, and then the permutation of the N integers determined. If the length of the keyword is allowed to be anything up to B integers long, then the total number of permutations possible for the transposition system is P where

$$P(N)=\sum^{B} N!$$

This number rapidly increases with N; P(6)=873 while $P(12)\approx 5.23*10^{8}$, making brute-force methods quickly impractical. However, an order-based GA offers the prospect of intelligently searching even a huge keyspace to find both the correct length and permutation of the transposition used, thus breaking the cipher. We now describe the design and use of such a GA.

Application Of GAs To Key Search

The focus of this paper is to apply a genetic algorithm to the problem of searching through the key space of a General transposition cipher. There are three steps which need to be defined in order to set up such an approach[9,10]. The first is the representation scheme. The problem is to set up a string representation of possible keys in a way that allows the mating and mutation processes to operate. The second is to determine a mating process consistent with the key representation. The third is to select a mutation process. Possible methods for all three are suggested in this section.

Key Representation

There are several ways in which a transposition key could be represented[8]. For the purpose of this study, a key for decoding a cipher is given by an ordered list of the integers 1 to N, e.g. "865132479" for N=9. It then applies each such guessed key to the ciphertext, and assesses the "fitness" of each by determining the extent to which the attempted decryption matches certain characteristics of plaintext. In essence, genetic cryptanalyst attempts to break the cipher by finding which characters go next to each other.

Once a representation scheme is available for a genetic algorithm, it is also necessary to supply an evaluation function. This function is used to determine the "best" representations. The evaluation function selected for this study is based on several ciphertext items and some plaintext. The process is as follows:

1) A given key in population is used to decipher the ciphertext.
2) Cut the resulting text of step 1 excite as long as the known plaintext.
3) For each character in results of step 2 are compared with known plaintext in the same position for Compute the sum of the similar characters between them to determine an error value.
4) The summation of the similar characters is divided by length of known plaintext characters to reduce sensitivity to large differences.
5) The error value is normalized and subtracted from 1 so that larger fitness values represent smaller errors.
6) The fitness value is raised to the 4^{th} power to amplify small differences.

When measured the result text of step 2 match the known plaintext, the summation evaluate to 1 so the fitness value is 0. The fitness equation is boundary between (0.0-1.0). As a result, the search process is always moving towards fitness values closer to 0. The overall guiding concept is that keys which produce plaintext.

The Mating Process

Given a population of keys, each one with a fitness value, the algorithm progresses by randomly selecting two keys for mating [3]. The selection is weighted in favor of keys with a high fitness value. That is, keys with a high fitness value have a greater chance of being selected to generate children for the next generation. The two parents generate two children using a crossover operation by Single point crossover [10]. For this study, one crossover point is selected, till this point the permutation is copied from the first parent, then the second parent is scanned and if number is not yet in the offspring it is added. For example, given two parents:

Parent 1	1	9	3	4	8	6	7	2	5

Parent 2	4	5	3	7	2	6	1

Child 1	**1**	**9**	**3**	**4**	**8**	5	7	2	6

Child 2	**4**	**5**	**3**	**7**	**2**	1	6

As can be seen, this crossover technique enables the children to inherit features of both parents without generating illegal permutations such as 4 3 3 1 5 7 8 9 9.

The Mutation Process

Following crossover, genetic cryptanalyst applies two mutation operators to introduce genetic diversity into the evolving population of permutations [8].

The first operator is a simple two-point mutation, which randomly selects two elements in the chromosome and swaps them. Thus 4 3 1 5 7 8 9 6 2

becomes 4 3 6 5 7 8 9 1 2 . Numerical studies of scheduling GAs by Syswerda [10] found this to be better than other mutation operators. The second operator is a shuffle mutation, which shunts the permutations forward by a random

number of places; thus 4 3 6 5 7 8 9 1 2 shunted forward six places becomes 8 9 1 2 4 3 6 5 7 . This was found to improve the performance of genetic cryptanalyst[3].

Once these various operations have been executed on one generation of chromosomes, the entire process is repeated with their offspring. The result is typically a fairly rapid increase in the fitness of the elite chromosome found, followed by a period of progressively less dramatic improvements evolving to an overall elite chromosome.

Conclusions

Our original goal was clearly met. The GA proved highly successful in finding the key for a simple substitution cipher. It is not believed, however, that this approach will always find the exact key. It is evident that in a short run, it will come close enough to the exact key that a visual inspection of the resulting plaintext could be used to determine any misplaced letters. Given that this is a promising algorithm, there are several open avenues for further research. For example, it may be possible to improve the evaluation function using a form of interval analysis such as that suggested by Anderson[11]. It may also be possible to include a trigram analysis in the evaluation function. Variations on the crossover and mutation procedures may significantly affect the behavior of the algorithm. In addition, other more complicated cipher may be analyzed. These concepts are currently under evolution.

Reference:

1- A. Menezes, P. Van Ooreschot, and S.Vanstone, “Handbook of applied cryptography”, CRC press,1996.

2- Beker, H., and Piper F. ,”Cipher Systems ”,the protection of communications. London: Northwood Books. 1982.

3- Davis, L.(Ed). “Handbook of Genetic Algorithms”. New York: Van Nostrand Reinhold. 1991.

4- Goldberg, David.. Genetic Algorithms in Search, Optimization, and Machine Learning. Reading MA: Addison-Weslay. 1980.

5- Garey, M. R., and D.S Johnson. Computers and Intractability: a guide to the theory of NP-completeness. New York: W. H. Freeman. 1979.

6- Dorothy E. and Robling D. , “Cryptography and Data Security”, PURUDUE University, 1988.

7- Hodges, A. Alan Turing : the Enigma (2^{nd} Ed). London: Vintage Press. 1992.

8- Matthews, R. A. J. An empirical method for finding the keylength of period. 1988.

9- Liepins, G. E. and M.R. Hilliard. 1989. Genetic Algorithms: Foundations and Applications. Annals of Operations Research. 21: 31-58.

10- Rawlins, G. Foundations of Genetic Algorithm. San Mateo CA: Morgan Kaufmann Publishers. 1991.

11- Anderson, Roland. Improving the machine recognition of vowels in simple substitution ciphers. Crytologia. 1986.