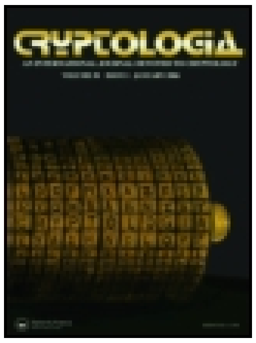

Cryptologia

ISSN: 0161-1194 (Print) 1558-1586 (Online) Journal homepage: http://www.tandfonline.com/loi/ucry20

THE USE OF GENETIC ALGORITHMS IN CRYPTANALYSIS

Robert A. J. Matthews

To cite this article: Robert A. J. Matthews (1993) THE USE OF GENETIC ALGORITHMS IN CRYPTANALYSIS, Cryptologia, 17:2, 187-201, DOI: 10.1080/0161-119391867863

To link to this article: http://dx.doi.org/10.1080/0161-119391867863

Published online: 04 Jun 2010.

THE USE OF GENETIC ALGORITHMS IN CRYPTANALYSIS

Robert A. J. Matthews

ADDRESS: 50 Norreys Road, Oxford OX2 9PT, ENGLAND.

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 GENALYST, an order-based GA for breaking a classic cryptographic system.

KEYWORDS: Genetic algorithms, cryptology, cryptanalysis, transposition ciphers

INTRODUCTION

There has recently been considerable excitement about the success of a new family of algorithms in attacking notoriously difficult (NP-complete) computational problems, such as the Travelling Salesman and resource-constrained scheduling [3]. These so-called "genetic algorithms" (GAs) use concepts drawn from the theory of evolution to "breed" progressively better solutions to problems with very large solution spaces [See [2] for an excellent introduction].

Cryptographic systems for data security rely on the existence of large solution spaces to deter attack. Indeed, in the design of any cryptographic system it is a *sine qua non* that the basic algorithm deter attempts to break it using so-called "brute force attacks", involving simply running through all possible ways in which the algorithm could have encrypted the target text.

Immunity to brute force attack is usually achieved by having as large a keyspace as possible. For example, the version of the Enigma machine used by the German armed forces at the outbreak of World War II had approximately 1.6×10^{20} different possible arrangements of rotors and plugboard [4].

So astronomically large a keyspace forces cryptanalysis to be based on more subtle approaches, such as knowledge of characteristic features of the language used (e.g. that E is the most common letter in many European languages), and/or knowledge of at least part of the text encrypted.

Although there is a considerable literature on the design of cryptographic systems, relatively little public-domain information exists on techniques for cryptanalysis. Much of it is old, and can only be used on relatively simple cryptographic systems. We present here an overview of a genetically-inspired cryptanalytic technique of wide applicability, and give a specific example of its use in breaking transposition ciphers, one of the classic cryptographic systems.

THE ARCHETYPAL GA

GAs typically have the following components:

(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 biassed 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, which randomly scrambles parts of some offspring.

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. Researchers have found that this is frequently a better solution than that found by traditional algorithms such as Monte Carlo searching [3].

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 recognising 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 optimised). Each initial guess is then coded as a chromosome of the form (0100111010...100) on which both crossover reproduction and mutation operators are applied.

However, problems such as scheduling, involving the optimisation 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 [8].

THE CRYPTANALYTIC USES OF GAs

With their ability to efficiently search huge solution spaces, GAs would seem a natural candidate for use in cryptanalysis. Initialised 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 text (see, e.g. [1]), which are used singly or in combination in virtually every cryptographic algorithm. *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.

The second broad category of cryptographic algorithm is *transposition*, 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 $26! \approx 4 \times 10^{26}$) 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. (After this paper had been submitted, the author learned that Spillman *et al.* have now used such a technique with great success [6]).

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 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 sequences.

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 [1, Chapter 2].

To illustrate in detail the explicit use of a GA in cryptanalysis, we now describe the construction of GENALYST (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 (see e.g. [1] and [5]), 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, GENALYST enables a ciphertext-only attack to be successfully made, based on only very general properties of the plaintext language.

THE BREAKING OF TRANSPOSITION CIPHERS

The type of transposition cipher attacked by GENALYST encrypts text according to the following classical two-stage algorithm:

(a) A 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 columns in the order dictated by the integers making up the key.

For example, suppose the key is 631254, and the text is NOW IS THE TIME FOR ALL GOOD MEN. By (a) we obtain the following

6	3	1	2	5	4
N	O	W	I	S	T
H	E	T	I	M	E
F	O	R	A	L	L
G	O	O	D	M	E
N					

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

WTROI IADOE OOTEL ESMLM NHFGN

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

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 $P(12) \approx 5.23 \times 10^8$, 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.

THE DESIGN OF GENALYST

Fitness assessment

GENALYST incorporates all the general features of a GA outlined above. However, being an order-based GA, a number of its characteristics are worthy of particular note. GENALYST begins with a set of chromosomes consisting of permutations of the integers 1 to N, e.g "86513247" for $N = 8$. 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 plain English. In essence, GENALYST attempts to break the cipher by finding which columns go next to each other. The fitness rating helps the algorithm achieve this by awarding scores according to the number of times two- and three-letter combinations (bi- and tri-grams) commonly found in English actually occur in the decrypted text. The more columns correctly put next to one another by GENALYST, the higher the fitness rating ascribed to that trial permutation. Whenever these combinations were found in decrypted text, points were awarded, the highest being given for the appearance of trigrams, as their appearance suggests that three columns have been correctly aligned. In addition, the appearance of the tri-gram "EEE", which virtually never appears in English despite featuring the most common letter, was punished by subtracting points. The table below gives the 10 combinations searched for, their percentage frequency in ordinary English (taken from Appendix 1 of [1]) and the score alloted to their presence in GENALYST.

	Bi/trigram	% freq	score		Bi/trig	% freq	score
1.	"TH"	3.0	+2	6.	"ED"	1.3	+1
2.	"HE"	3.0	+1	7.	"THE"	2.0	+5
3.	"IN"	1.9	+1	8.	"ING"	0.8	+5
4.	"ER"	1.9	+1	9.	"AND"	0.7	+5
5.	"AN"	1.4	+1	10.	"EEE"	0.0	−5

Table 1. Fitness scoring system used by GENALYST.

Fitness/correctness correlation

The correlation between fitness and the correctness of a chromosome is not perfect, nor can it ever be. It is always possible that GENALYST can award relatively high fitnesses to chromosomes that by chance produce relatively high numbers of bi- and tri-grams.

Nevertheless, the fitness rating of GENALYST does correlate well with the correctness of a chromosome. To show this, a target chromosome was used to encrypt a text, and a sample of random chromosomes used for decryption. Each of these chromosomes was then rated for its similarity to the target, as measured by their GENALYST-allotted fitnesses. These ratings were then compared to the actual similarity to the target, as measured manually by awarding points whenever an integer making up the chromosome had one or both of its neighbours the same as in the target key permutation, and/or in the same position. Regression analysis gave a correlation coefficient for the fitness/target-similarity relationship of $r = +0.66$, with a statistical significance against chance of better than the $p = 0.01$ level.

Selection of fittest chromosomes

Once the fitness ratings have been alloted, GENALYST uses linear normalisation [2, p. 31] to bring the various fitnesses into the range 0 (least fit) to 100 (fittest), all the other fitnesses being spaced at equal intervals between these two extremes. This waters down the rapid dominance of chromosomes whose apparently high fitnesses may have occurred by chance, and also splits up chromosomes with identical fitnesses, thus promoting a race for survival.

GENALYST then applies "roulette wheel" selection, in which the probability of a chromosome being selected is proportional to its normalised fitness. This constitutes the crucial "survival of the fittest" filter. Clearly, fitter chromosomes stand a better chance of passing through it, but it is important to note that even chromosomes with a very low initial fitness have some chance of getting through. This ensures that each has some chance of developing into a better chromosome by subsequent breeding.

As there is a risk of the fittest chromosome not surviving the filter, however, GENALYST incorporates "elitism", which ensures the fittest ("elite") chromosome found up to that point is always allowed into the breeding pool.

A proportion of those chromosomes surviving the filter are then used for breeding. This proportion linearly decreases as GENALYST continues its search, an approach based on research [2, p. 50] showing that moving from high to lower breeding proportions optimises the effectiveness of GAs.

The breeding process itself is achieved using the position-based crossover technique proposed by Syswerda [8], whose extensive numerical work has shown to be particularly effective for scheduling GAs, the family of order-based GAs to which GENALYST belongs. Two chromosomes are taken for breeding, and a random bit-string of the same length is generated. For example, for $N = 9$ we may have:

Chromosome A:	4	5	1	7	8	3	9	2	6
Chromosome B:	4	3	1	9	2	8	7	6	5
Bit-string:	1	0	0	1	1	0	1	0	1

To form one child of A and B, take those elements of A corresponding to a 1 in the bit-string:

4 * * 7 8 * 9 * 6

and enter the elements of A omitted, i.e. 5,1,3 and 2 in the order they appear in chromosome B, i.e. 3,1,2,5, giving:

Child A: 4 3 1 7 8 2 9 5 6

To form the other child of A and B, take those elements of B corresponding to a 0 in the bit-string:

* 3 1 * * 8* 6 *

and enter the elements of B omitted, i.e. 4, 9, 2, 7 and 5 in the order they appear in chromosome A, i.e. 4, 5, 7, 9, 2, giving

Child B: 4 3 1 5 7 8 9 6 2

As can be seen, this cross-over 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.

Following crossover, GENALYST applies two mutation operators to introduce genetic diversity into the evolving population of permutations. Both are applied with increasing frequency as the populations evolve to offset the tendency to evolve into evolutionary niches (i.e. local minima in the solution space).

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 [8] 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 also found to improve the performance of GENALYST.

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.

TEST PROCEDURE

To provide a useful indication of the performance of GENALYST, it is important that the results be as widely applicable as possible. For example, if the results were based on decryption of a test text with an unusually high "true fitness rating" (i.e. its fitness when correctly turned into English), this would cast an unrepresentatively favourably light on the performance of GENALYST. Small successes in obtaining the correct decryption chromosome for such a text would produce fitnesses unnaturally above the "noise" of completely incorrect chromosomes. Similarly, use of a test text with low true fitness would not do GENALYST justice. To address this issue, we note that the fitness rating ascribed to GENALYST of a typical sample of English may be calculated theoretically using the bi- and tri-gram frequencies given in, for example, [2]. Let the text length be L. Then the theoretical true fitness rating of a typical English text of this length is $\langle F \rangle$ where

$$\langle F_L \rangle = L \sum^{Q} (P_i S_i / 100)$$

P_i is the percentage frequency of that bi- or tri-gram in the text, S_i is the fitness score ascribed by GENALYST to the ith bi- or tri-gram tested for, and the summation is over the Q bi- and tri-grams checked. For example, in GENALYST, $Q = 10$, and from Table 1 we see that the first bi-gram tested for is "TH", which is alloted a fitness rating S_i of +2 by the algorithm, and has a frequency of 3 per cent in typical English. Proceeding similarly, we find that a typical English text will be given a "true fitness rating" of $\langle F_L \rangle = 0.33L$ by GENALYST. Measurements of the $\langle F_L \rangle$ of 20 texts drawn at random confirmed this theoretical figure. This formula for $\langle F_L \rangle$ thus gives us a criterion for assessing whether a test text has a true fitness which is representative of typical English texts.

The length of the test text can also affect the measured performance of GENALYST. Texts whose length are an integer multiple of the keylength are considerably easier to break, as all the transposition matrix columns then have the same length. This reduces the importance of getting each column in the right position for decryption.

Thus, for the performance tests, a text length of 181 characters was chosen. This figure is a prime number, but is almost exactly divisible by 6, 9, 10 and 12. The significance of this is that GENALYST uses each of these as trial keylengths in its search for the correct one. With 181 being prime, we will therefore obtain results under a worst-case scenario: not all the column lengths will be the same, making cryptanalysis considerably harder, and there is the added possibility that

GENALYST will be unable to discriminate between key-lengths of 6, 9, 10 and 12.

The specific text used was taken from page 74 of *The Newtonian Casino* by Thomas Bass (Longman 1990), this being a randomly-selected text with a true fitness rating $\langle F_L \rangle$ of 60, the same as that expected theoretically for a text of 181 characters:

> "Given the right equations and variables, four clicks tapped with the aid of a big toe will record everything required for roulette prediction by a Laplacian intelligence. Click the passage of a point on the rotor the f"

Surprisingly few reports of the use of GAs give any statistical analysis of their performance compared to that of traditional search techniques. For GENALYST the most obvious standard by which to assess its efficacy in breaking transposition ciphers is a Monte Carlo random search, in which a series of guesses is made to determine the right key. In each cycle of such a Monte Carlo search, an elite chromosome emerges which is kept until it is superseded by a fitter one. In what follows, we report the outcome of statistical tests applied to see how much better GENALYST did than the Monte Carlo approach (henceforth termed RANDOM).

Having described the core of the test procedures, we now move to a description of the tests proper. The results quoted were obtained using a Microsoft QuickBasic compiler on both a 286-based PC running at 12 MHz, and a 25 MHz 486 machine.

USING GENALYST TO FIND CORRECT KEYLENGTHS

As noted earlier, the breaking of a transposition cipher can be divided into two elements: finding the correct keylength N, and then finding the correct permutation of that key, i.e. the permutation of integers 1 to N.

To investigate the performance of GENALYST in the first task, we encrypted the standard text given above using a target transposition key, K. We then used GENALYST to search for the best possible decryption of the text using seven keys of lengths ranging from 6 to 12, one of which was the same as K. The aim was to see if the fittest of the seven elite chromosomes evolved during these seven runs of GENALYST had a length equal to that of the target key, K. This would imply that GENALYST can be used firstly to pick out the correct keylength, and then used separately to solve the harder task of finding the correct permutation of this key as well.

The results bore out this conclusion. Three target keylengths, $N = 7$, 9 and 11 respectively, were used to encrypt the test text. GENALYST then began with 20 random guesses of the correct key, starting with a keylength of 6, and "bred" progressively better key chromosomes over 25 iterations ("generations"). At the end of the run for each keylength, the fitness of the elite chromosome was noted. In all the results which follow, the initial and final probabilities of crossover were set at 0.8 and 0.5 respectively, the corresponding probabilities of point-mutation and shuffle mutation being (0.1, 0.5) and (0.1, 0.8) respectively, and all were linearly interpolated between these extremums during each run. These values are typical of those used in many GA applications (see, e.g. [2]).

Both the size of the initial population of chromosomes, and the number of generations used are, however, small by GA standards. They were chosen both by a desire to see how powerful GENALYST was even with relatively small "gene pools" to work with, and by constraints of CPU time.

For each of the three target keylengths, GENALYST was run five times. It was found that the fittest of the elite chromosomes generated did indeed have keylengths equal to that of the key used to encrypt the text in 13 of the 15 runs (87% success rate). More specifically, GENALYST obtained the correct keylength for all the runs involving a key of $K = 7$ and 9, and for 3 of the 5 runs for the $K = 11$.

Although impressive, repeat trials using the Monte Carlo search algorithm RANDOM produced an outcome often found by GA researchers, viz. that random searching often does quite well. Specifically, RANDOM equalled the performance of GENALYST in this task, also giving an overall 87% success rate, produced by obtaining the right keylength for all but one case in $K = 9$ and $K = 11$ respectively.

The real advantages of the genetically-based approach taken by GENALYST emerged in the more difficult problem of actually decrypting a ciphertext, i.e. of finding the correct permutation of integers within a keylength K.

USING GENALYST TO FIND CORRECT PERMUTATIONS

We can gauge the success of GENALYST in finding the correct permutation of the integers making up the key by measuring the fitness of the elite chromosomes generated in each of the 15 runs described above. The closer the average fitness ratings were to 60 - the true fitness of the test text used - the closer GENALYST came to completely decrypting the ciphertext. The results of beginning with 20 chromosomes and evolving over 25 generations are summarised in the table below:

Keylength	Mean elite fitness	
	GENALYST	RANDOM
7	51.6±5.7	44.8±7.0
9	52.6±7.6	37.6±6.5
11	31.0±4.5	29.6±6.2

Table 2. Comparison of performance of GENALYST and RANDOM algorithms.

GENALYST out-performed RANDOM for all keylengths. In one run with both $K = 7$ and $K = 9$, GENALYST completely broke the cipher, turning the encrypted text into plain English; RANDOM never achieved this. These results are all the more remarkable for being produced using a relatively small gene pool of 20 chromosomes reproduced over 25 generations; many GAs use 50-100 chromosomes over 50-100 generations.

It is also important to note that GENALYST involves very little extra computational effort than RANDOM. The single biggest task for a GA is the same as that for any search technique, namely fitness rating. Thus, as the genetic elements of a GA constistute only a small proportion of the total CPU demand, even a small advantage obtained by a GA is worth having.

That said, Student t-tests on the results in Table 2 show that only in the case of $K = 9$ was the difference in performance between the two algorithms significant. As expected, the performance of both fell away with increasing keylength (and thus with greatly increased search space, which scales as $K!$). It is thus clear that although GENALYST does surprisingly well even with a small gene pool, its real power is still not being exploited.

Using larger gene pools

Having shown that GENALYST can pick out the correct keylength using a relatively small gene pool, we looked at the success of the algorithm in finding the right permutation for the largest ($K = 11$) transposition key (solution space size $\approx 4 \times 10^7$) using a larger gene pool allowed much greater evolution. Following Syswerda [8] we took a gene pool of 30 chromosomes and evolved them for 100 generations. As expected, the results thus obtained on the test text with $\langle F_L \rangle = 60$ were more impressive. Averaged over 15 runs, the average fitness of the elite chromosome evolved after 100 generations by GENALYST was found to be

$$\langle F_G \rangle = 41.1 \pm 5.2$$

For comparison, the average fitness of the elite chromosomes found by RANDOM was

$$\langle F_R \rangle = 36.3 \pm 6.8$$

Analysis shows that although the performance of both GENALYST and RANDOM was improved by using a larger gene pool, only the performance of GENALYST had been boosted significantly ($p < 0.001$). Student t-tests also reveal that the performance of GENALYST with the larger gene pool is significantly better than that of RANDOM ($p < 0.05$) despite requiring little extra computational effort.

HYBRIDISING GENALYST

In view of the fact that no run of either algorithm explored more than about 10^{-4} of the total solution space of the $K = 11$ transposition, it is hardly surprising that no complete breaks were obtained. However, a complete break is ultimately required from any cryptanalytic technique. To obtain this goal we must introduce a notion stressed by a number of GA researchers: the hybridisation of the GA, i.e. the combination of the genetic abilities of the GA with other techniques.

We note first that the evidently sub-maximal fitness ratings given above correspond to GENALYST obtaining partly correct decryption permutations. Elite chromosomes from a set of runs of GENALYST typically contain elements of the actual decryption key.

Two techniques can then be used to turn the partially-correct elite chromosomes into a complete break. One is to use these elite chromosomes as the starting point for another run of GENALYST, which should combine the best features of each elite chromosome to produce an even more fit solution.

However, no GA can guarantee arriving at the true solution to a problem, and in trials this iterative use of GENALYST turned out to be less computationally efficient as a permutation technique henceforth termed "perming".

Perming

In this, a set of about half a dozen elite chromosomes found during runs of GENALYST is analysed to find common sequences of integers occurring within them. These fragments are then taken and anagrammed, and the resulting key sequence used to decrypt the ciphertext. Each attempted decryption is then visually studied for its resemblance to English.

"Perming" turns out to be very effective in finally breaking the transposition, typically requiring only a few anagrams of the fragments before the ciphertext is readable. The reasons are not hard to see. Whereas the initial problem of breaking a transposition cipher of length 11 has a solution space of almost 4×10^7, GENALYST typically enables the identification of a 3-4 fragments each consisting of up to four keynumbers. The problem of finding the single correct permutation out of the 11! (about 40 million) possible has been reduced to finding it among a selection of 4! (24), a problem six orders of magnitude easier.

Thus the most effective cryptanalytic technique appears to be hybridising GENALYST with the perming technique to anagram the key fragments, the final breaking of the cipher into English being achieved by use of a neural network, i.e. the human brain.

CONCLUSIONS

In this paper, we have argued that genetic algorithms are a valuable tool in the cryptanalysis of certain classes of cipher, and have shown that transposition ciphers can be broken using such a GA, namely GENALYST. Given the number of free parameters available (e.g. size of initial gene pool, probability of cross-over, number of generations) there is, as with any GA, plenty of scope for further research. Extra, or different, fitness functions might be used; different mutation and cross-over rate combinations may give even better results than those reported here.

The fruits of such research would not, however, be restricted to cryptanalysis. GENALYST is an archetypal order-based GA for a problem for which the optimal fitness can be accurately calculated in advance. It can thus serve as a useful test-bed for order-based GA research in general, with performance improvements found for GENALYST likely to prove valuable for GA research on many other notoriously hard problems.

ACKNOWLEDGEMENTS

It is a pleasure to thank Dr. David Davis of Tica Associates for his interest and encouragement in this project, and for his excellent book on GAs; Dr. Chris Bishop of AEA Technology for suggesting ways of speeding up GENALYST; Dr. Darrell Whitley of Colorado State University for providing pre-prints of GA research, Dr. Richard Spillman of the Pacific Lutheran University for critical comments and a pre-print on results from a simple substitution cipher GA, and Mike Fisher and Rachel Maling of Research Machines, Oxford, for providing additional computational facilities.

REFERENCES

1. Beker, H., and F. Piper. 1982. *Cipher Systems: the protection of communications*. London: Northwood Books

2. Davis, L. (Ed). 1991. *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold.

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

4. Hodges, A. 1992. *Alan Turing: the Enigma (2nd Ed)*. London: Vintage Press.

5. Matthews, R. A. J. 1988. An empirical method for finding the keylength of periodic ciphers. *Cryptologia*. 12(4): 220-224

6. Spillman, R., M. Janssen, B. Nelson, and M. Kepner. Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers. *Cryptologia*. 16(1): 31-44.

7. Starkweather, T., D. Whitley, K. Mathias, and S. McDaniel. 1991. *Sequence scheduling with genetic algorithms*. Report CS-91-130. Department of Computer Science, Colorado State University.

8. Syswerda, G. 1991. Schedule Optimisation using genetic algorithms. In Davis, L. (Ed). 1991. *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold. pp. 332-349.

BIOGRAPHICAL SKETCH

After graduating in physics from Oxford University in 1981, the author became a science journalist, and is currently Science Correspondent of *The Sunday Telegraph* in London. He continues to conduct academic research, with recent papers in algebraic number theory and solar system astrophysics.