The Cryptanalysis of a Three Rotor Machine Using a Genetic Algorithm

A. J. Bagnall, G. P. McKeown and V. J. Rayward-Smith
School of Information Systems
University of East Anglia
Norwich, NR4 7TJ
England

Abstract

This paper describes a method of deciphering messages encrypted with rotor machines utilising a Genetic Algorithm to search the keyspace. A fitness measure based on the phi test for non randomness of text is described and the results show that an unknown three rotor machine can generally be cryptanalysed with about 4000 letters of ciphertext. The results are compared to those given using a previously published technique and found to be superior.

1 INTRODUCTION

A cryptographic system effects an enciphering transformation from plaintext, $\mathbf{x}$, to ciphertext, $\mathbf{y}$, both of which are vectors of n letters from a finite set of symbols (an alphabet) of size q, which we call Z_q. A *key*, k, for a cryptographic system is a parameter defined on a *key space*, K, with the property that knowledge of the value of k will enable the recovery of plaintext from ciphertext. Cryptanalysis is the process of attempting to recover encrypted plaintext without knowledge of the key. In this paper we consider cryptanalysis with ciphertext only, as opposed to with known or chosen plaintext and ciphertext. It is the former situation which presents the hardest problem for the attacker.

Genetic Algorithms (GAs) have previously been used in cryptanalysis for solving simple substitution systems [3], transposition ciphers [11] and knapsack based systems [12]. The cryptographic systems solved are all fairly straightforward, and readily solvable by other methods. For example [10] gives various methods for solving knapsack problems larger than those solved in [12].

We show that a genetic algorithm can be used to successfully search the very large discrete keyspace of a rotor machine, of magnitude up to $(26!)^3$, using a simple statistical measure of suitability. The method involves finding the last rotor of a three rotor machine using a GA and then solving the resulting two rotor machine using the iterative technique described in [1]. The plaintext is assumed to be n independent realisations of a random variable defined on alphabet Z_q, with probability distribution derived from observation of the single letter frequencies of English. The distribution we use is that given in [7]. The statistical measure of fitness is based on the fact that a two rotor machine with odometer-like rotation is periodic, with period q^2. If we map the ciphertext through the correct last rotor (with a known rotation pattern), and split this new ciphertext into q^2 ciphertext strings, these strings will exhibit the characteristics of monoalphabetically enciphered plaintext. Based on the ϕ statistic for testing the non-randomness of text (see [8]), the fitness for any proposed last rotor, ρ, is the probability of having observed an average ϕ value at least as large as the value actually observed, given ρ and that the text mapped through ρ results from q^2 monoalphabetic substitutions of the plaintext.

2 ROTOR MACHINES

Rotor machines formed the basis for most military and commercial cryptography until the late 1950s. The German Enigma, the Hebern machine and the Converter M-325, all described in [2], are variations on the basic machine we use.

A rotor machine is a cryptographic device consisting of N rotors. A rotor is a disk with a node on each side for each letter in the appropriate alphabet (size q) and electrical contacts between the nodes effecting a substitution π. After each such substitution

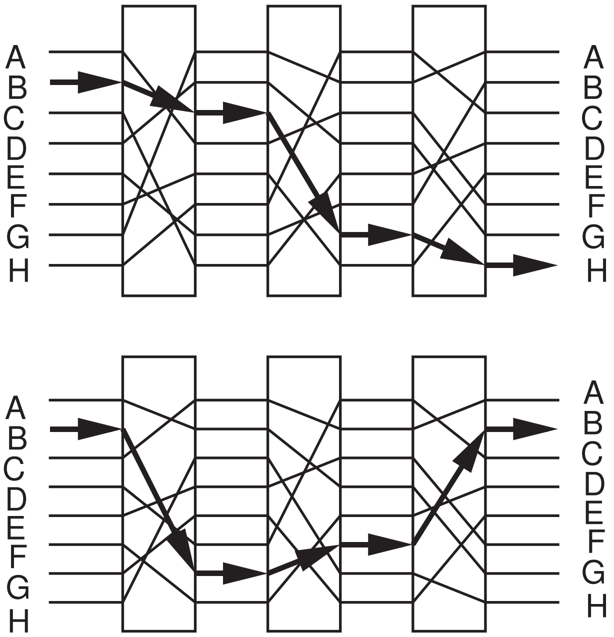

Figure 1: A three rotor machine for an eight letter alphabet before and after the first rotor has rotated one place.

the contacts rotate in accordance with a rotation pattern while the nodes remain in the same position (or vice versa). Rotating a rotor a places yields another (usually different) substitution,

$$S(a, \pi) = C^{-a} \pi C^{a}.$$

$S(a, \pi)$ is the composition of π with the Caesar substitutions C^{-a} and C^{a}. A Caesar substitution, C^{a}, is simply a shift of a places,

$$C_a : i \rightarrow (i + a) \quad 0 \leq a,\ i < q.$$

Rotor machines are built by concatenating rotors. Let

$$\Pi = \{\pi_0, \pi_1, \ldots, \pi_{m-1}\}$$

be a set of substitutions representing a bank of m rotors for encipherment of the first letter and

$$R = \{r_0, r_1, \ldots, r_{m-1}\}$$

be a set of *displacement functions*,

$$\begin{aligned} &r_s : Z_q \rightarrow Z_q \\ &r_s : 0 \rightarrow 0 \\ &r_s : i \rightarrow r_s(i), \quad 0 \leq s < m, i > 0 \end{aligned}$$

where $r_s(i)$ is the rotational displacement of the s^{th} rotor for the encipherment of the i^{th} element of $\mathbf{x}$. The enciphering substitution for the i^{th} letter of plaintext is given by

$$\begin{aligned} S(R(i), \Pi) \quad = \quad & C^{-r_{m-1}(i)} \pi_{m-1} C^{r_{m-1}(i)} C^{-r_{m-2}(i)} \pi_{m-2} \\ & C^{r_{m-2}(i)} \ldots C^{-r_0(i)} \pi_0 C^{r_0(i)} \end{aligned}$$

and the deciphering substitution is the inverse,

$$\begin{aligned} S(R(i), \Pi)^{-1} \quad = \quad & C^{-r_0(i)} \pi_0^{-1} C^{r_0(i)} C^{-r_1(i)} \pi_1^{-1} C^{r_1(i)} \\ & \cdots C^{-r_{m-1}(i)} \pi_{m-1}^{-1} C^{r_{m-1}(i)}. \end{aligned}$$

The set of displacement functions is usually assumed to be known. A common set of displacement functions are those that follow an odometer-like rotation pattern, such as

$$r_s(i) = \lfloor (i/q^s) \rfloor \quad 0 \leq s < m, \quad 0 < i < \infty.$$

An m rotor system with odometer-like displacement functions, R, is a *polyalphabetic substitution system* defined by the sequence of substitutions

$$< S(R(i), \Pi) \ \mid \ 0 \leq i < \infty >,$$

with period P, which divides q^m. A rotor machine with a set of displacement functions which are not odometer-like can, at most, produce q^m different substitutions. A fuller mathematical description of rotor machines is given in [7].

The key for a rotor machine must define:

1. the number of rotors, m,
2. the initial rotor substitutions, Π,
3. the displacement functions, R.

We assume the number of rotors is known and that the set of displacement functions follow the odometer-like pattern described above. So the keyspace, K, is the set of all N-tuples of possible substitutions and a key, k, is an element of a subset of K. The cryptanalysis problem is to find an estimate of k, $\hat{k}$, given a string of ciphertext.

3 CRYPTANALYSIS OF A THREE ROTOR MACHINE

Our method of attack is to find the last rotor of a three rotor machine using a Genetic Algorithm (GA), and then to solve the resulting two rotor machine using the iterative technique described in [1].

3.1 GENETIC ALGORITHMS

We implemented the problem on the *X-GAmeter toolkit* [9], developed at the UEA, which enables the easy use of GAs on a variety of problems. The decisions in implementing a GA for a particular problem are concerned with representation, fitness evaluation and the select, create and merge methods. The stopping condition we use is to halt the GA if there had been no change in the best solution so far in 500 generations.

3.1.1 Representation

The most obvious representation of a rotor is an array of m integers representing the wiring from each position. The subset of the keyspace containing correct keys for a three rotor machine with known rotation functions has magnitude q^2. That is to say there are multiple solutions to a rotor machine with more than one rotor. If we define the set of rotors

$$\rho = \{\rho_0, \rho_1, \rho_2\}$$

where

$$\rho_2 = \pi_2 C^{b_2}, \ \ 0 \leq b_2 < q,$$

and

$$\rho_1 = C^{-b_2}\pi_1 C^{b_1}, \ \ 0 \leq b_1 < q$$

then if

$$\rho_0 = C^{-b_1}\pi_0,$$

$$\begin{aligned} S(R,\rho) &= C^{-r_2}\rho_2 C^{r_2} C^{-r_1}\rho_1 C^{r_1} C^{-r_0}\rho_0 C^{r_0} \\ &= C^{-r_2}\pi_2 C^{b_2} C^{r_2} C^{-r_1} C^{-b_2}\pi_1 C^{b_1} \\ &\quad C^{r_1} C^{-r_0} C^{-b_1}\pi_0 C^{r_0} \\ &= C^{-r_2}\pi_2 C^{r_2} C^{-r_1}\pi_1 C^{r_1} C^{-r_0}\pi_0 C^{r_0} \\ &= S(R,\Pi). \end{aligned}$$

This means that we can arbitrarily fix a wiring in the last 2 rotors. For any proposed solution, ρ', we set $\rho'_1(0) = 0$ and $\rho'_2(0) = 0$.

3.1.2 Fitness

The basis for our fitness evaluation is that plaintext enciphered by a monoalphabetic substitution will still exhibit characteristics of the original distribution. A periodic polyalphabetic substitution system with period P can be reduced to a series of monoalphabetic substitution systems by splitting the ciphertext, $\mathbf{y}$, into P text strings, $\mathbf{y_j}$, where

$$y'_j(i) = y'(i*P+j) \ \text{ for } \ 0 \leq j < P, \ 0 \leq i < n/P.$$

Each string results from monoalphabetic encipherment. Mapping the ciphertext through the correct last rotor of a rotor machine with periodic rotation will produce a new ciphertext string with the property that the q^{m-1} text strings will exhibit the non-random properties of the plaintext distribution. We can, then, reduce the search space to the possible wirings of the last rotor and consider a fitness measure for this non-randomness.

Phi test

Consider v independent observations of random variable, X, with sample space Ω, of size q, and with range Z_q. Suppose that the observed frequencies of each of the elements in Ω is given by $f_0, f_2, \ldots, f_{q-1}$. If we define the statistic ϕ as

$$\phi = \sum_{i=0}^{q-1} f_i(f_i - 1),$$

then

$$\mu_\phi = E(\phi) = s_2 v(v-1),$$

and the variance is

$$\sigma^2_\phi = 4v^3(s_3 - s_2^2) + 2v^2(5s_2^2 + s_2 - 6s_3) + 2v(4s_3 - s_2 - 3s_2^2),$$

where

$$s_2 = \sum_{i=0}^{q-1}(P(X=i))^2 \ \text{ and } \ s_3 = \sum_{i=0}^{q-1}(P(X=i))^3.$$

See [8] for further description of the ϕ statistic and proof of the above results. If random variable X has a probability distribution given by our single letter frequency distribution, then

$$E(\phi) = 0.068944v(v-1)$$

and

$$\sigma^2_\phi = 0.004344v^3 + 0.110448v^2 - 0.114794v.$$

For a polyalphabetic substitution with period P, and ciphertext $\mathbf{y}$ of length n, if we assume n is divisible by P for simplicity, then the P strings of text, $\mathbf{y_j}$, of length n/P can be considered as P observations of random variable ϕ, defined on the sample space of all (n/P) possible letter combinations. Then

$$\bar{\phi} = \frac{\sum_{i=1}^{P}\phi_i}{P}$$

and

$$\sigma^2_{\bar{\phi}} = \frac{\sigma^2_\phi}{P}.$$

By the Central Limit Theorem, [13], the limiting form of the distribution of

$$Z = \frac{\bar{\phi} - \mu_\phi}{\sigma_{\bar{\phi}}}$$

is the standard normal distribution.

So, if we let $t = q^{m-1}$, the fitness of a proposed last rotor, ρ_2', is calculated as follows:

- let $\mathbf{y}'$ be the transformed ciphertext found by subjecting ciphertext $\mathbf{y}$ to the sequence of substitutions
$$S(r_2(i), \rho_2')^{-1}, \quad \text{for } \ 0 \le i < n.$$
- Let $\mathbf{y}_\mathbf{j}'$ be substrings of $\mathbf{y}'$ such that
$$y_j'(i) = y'(i * t + j) \ \text{ for } \ 0 \le j < t, \ 0 \le i < n/t.$$
- Calculate A, the observed $\bar{\phi}$ value for the substrings y_j'.
- Let the fitness of ρ_2' be
$$\frac{A - \mu_\phi}{\sigma_{\bar{\phi}}}.$$

Even though each text string has less than 10 letters, the fact we have so many strings means that $\bar{\phi}$ will be approximately normally distributed with small variance. Our estimate of the last rotor, $\hat{\rho}_2$, is the substitution which maximises the probability of observing a $\bar{\phi}$ value at least as large as A, conditional on the fact that $\mu_{\bar{\phi}} = 0.068944(n/t)(n/t-1)$ and $\sigma_\phi^2 = 0.004344(n/t)^3 + 0.110448(n/t)^2 - 0.114794(n/t)$.

In effect we are using the critical level of the test

$$H_0 : \mu_\phi = 0.068944(n/t)(n/t-1) \text{ vs}$$
$$H_1 : \mu_\phi < 0.068944(n/t)(n/t-1)$$

as the fitness of a proposed last rotor, assuming the variance is σ_ϕ^2.

3.1.3 Select, Create and Merge Methods

We experimented with various select, create and merge methods and, for reasons given below, settled on *roulette*, *PMX* and *new solutions* [9]. The mutation operators used are described below.

Mutation

In order to maintain diversity and avoid localised hill climbing it is desirable that when an offspring is created there is a chance that it may mutate. We used the following two mutation operators:

1. randomly swap two rotor wirings, i.e. effect a random transposition on an offspring.
2. shift a randomly selected substring of rotor wirings to random position.

offspring : 5 1 3 2 6 4 7 8 9

random substring: 1 3 2

random position = 6

mutant : 7 8 9 5 6 1 3 2 4

Figure 2: An example of shift mutation with a ten letter rotor

We allowed the mutation operators to move the first wiring, and then realligned the mutated offspring so that $\rho_2'(0) = 0$. The reason for this was that there was a tendency for the GA to converge to the wrong last rotor, for example, if by fixing the first wiring we are looking for

$$\rho_2 = \pi_2 C^{b_2} \quad \text{where} \quad b_2 = 7$$

then a proposed solution close to a different correct last rotor, for example,

$$\rho_2' = \pi_2 C^{b_2} \quad \text{where} \quad b_2 = 8$$

will have a high fitness. So the GA may converge to a solution with the first and one other wiring correct. Thus we allow mutation to move the position of the first wiring with respect to the rest of the permutation. However, since PMX is respectful and crossover occurs before mutation, the crossover operator will not move the position of the first wiring. This is important as PMX maintains position within the mapping section. Once an offspring had been created by the crossover operator, the probability of it mutating by one of the operators described above was set at 0.5. If an offspring was to mutate, one of the two mutation operators was selected randomly. Lower mutation probabilities tended to make the GA slower at finding the correct solution.

Merge Method

Introducing the offspring to the population pool requires the removal of some of the existing chromosomes to maintain a constant pool size. The method we use, termed *new solutions*, is to only allow offspring into the pool if they had better fitness than the worst member of the current pool and if they were not simply replicas of existing pool members. An offspring to be included replaces the chromosome whose fitness is closest to and less than (when maximising) the fitness of the offspring.

4 RESULTS

The three rotors we use are

$\pi_0 = \{9,15,6,21,14,20,12,5,24,16,1,4,13,7,25,17,3,10,0,18,23,11,8,2,19,22\}$,
$\pi_1 = \{13,25,9,7,6,17,2,23,12,24,18,22,1,14,20,5,0,8,21,11,15,4,10,16,3,19\}$,
$\pi_2 = \{5,10,16,7,19,11,23,14,2,1,9,18,15,3,25,17,0,12,4,22,13,8,20,24,6,21\}$.

They are rotors VI to VIII of the German Naval Enigma machine as given in [2]. Of the four sets of plaintext the first two were generated by random sampling of the distribution in [7], plaintext 3 was taken from an article in a newspaper and plaintext 4 was found by randomly sampling an intentionally skewed distribution. For the last distribution the probability of observing an E was decreased by 0.02, as was the probability of observing a T, and the probability of observing an X and the probability of observing a Z were increased by 0.02. This was done to test the method against deliberately deceptive text. All experiments were performed on an Alpha station 250 4/266. There seemed to be little difference between roulette and ranking selection methods, and the results below were obtained using roulette selection. Tables 1, reft2 and 3 give the average number of generations, evaluations and time required over 5 runs of the GA to find the correct last rotor, with poolsize p and textlength n as given.

Once the last rotor was found the simplest method of solving the resulting 2 rotor problem is to use Baum's maximisation algorithm described in [1] with alternating iterations between row stochastic and column stochastic constraints. We also implemented a GA for a two rotor problem using the likelihood function

n=4732	Generations	Evaluations	Time (s)
plaintext 1	716	18601	111
plaintext 2	1208	31287	190
plaintext 3	1477	38539	232

Table 1: Average results for finding the correct last rotor with 4732 letters of ciphertext and a GA population of 50

n=4056	Generations	Evaluations	Time (s)
plaintext 1	1447	37863	213
plaintext 2	1526	40683	231
plaintext 3	1582	41042	237

Table 2: Average results for finding the correct last rotor with 4056 letters of ciphertext and a GA population of 50

(see [13]) as a fitness measure to search the space of all possible 2-rotors, and obtained results comparable to those of the iterative technique.

Number of iterations required to solve the resulting two rotor problem with 2000 letters of ciphertext:

plaintext 1 : 42

plaintext 2 : 39

plaintext 3 : 46

plaintext 4 : 67

We performed a further series of experiments with 3380 letters. The 3380 letter problem is harder to solve because the fitness measure is coming to the limit of its usefulness. It is possible that the rotor with the highest fitness will not be the correct last rotor. For 15 runs of plaintext 1, 2, and 3 with 3380 letters, the GA found the correct rotor on only four occasions. However, on each run the GA would always get a substantial number of wirings correct, and usually it would be the same or very similar set of wirings. One method of finding the last rotor with 3380 letters is to run the GA two or three times and look for common wirings in the best solutions of each run. With plaintext strings 1 2 and 3 this usually gave a rotor with 16 or 17 positions correct. The rest could be found by an exhaustive search of all the remaining possible wirings (10! or 9!

n=4732	Generations	Evaluations	Time (s)
plaintext 4	3049	304102	1737

Table 3: Average results for finding the correct last rotor with 4732 letters of ciphertext and a GA population of 200

different possible rotors). For 4056 and 3380 letters of plaintext 4 the GA generally only found between 6 and 12 correct wirings.

4.1 ALTERNATIVE ROTATION PATTERN

It could be claimed that using an odometer-like rotation problem makes the rotor machine easier to crack. The technique requires that the rotations for the encipherment of the plaintext letters

$$x(i*t+j) \ \text{for} \ 0 \leq j < t, \ 0 \leq i < n/t.$$

not be the same for all i, otherwise each text string y_j has been monoalpabetically enciphered. This is not a restrictive condition, however, since a rotation pattern which did produce the same rotation for each string would reduce the rotor machine to a two rotor problem. Suppose the rotation patterns follow a known pseudo random pattern. We can still split the text so that the text mapped through the last rotor will have been monoalphabetically enciphered, but the rotations for the last rotor will now be different. The results in Table 4 are for a rotor machine with odometer-like rotation for rotors one and two and pseudo random rotation for rotor three.

n=4732	Generations	Evaluations	Time (s)
plaintext 1	876	22869	127
plaintext 2	1624	40602	223
plaintext 3	1448	36197	203

Table 4: Average results for finding the correct last rotor with 4732 letters of ciphertext and a GA population of 50

5 COMPARISONS

Iterative Technique for the Three Rotor Problem

We implemented the technique in [1] for a three rotor system on the same machine as the GA. The algorithm found the correct three rotor machine for plaintext 1 with 4732 and 4056 letters in 394 and 589 iterations but with 3380 letters it had only achieved a correct decipherment rate (CDR) of 10% after 1000 iterations. For plaintext 2 3 and 4, 1000 iterations (taking approximately 5 hours) gave a CDR of no better than 7%, little better than randomly guessing each letter, and half as good as guessing E for every letter.

Other Heuristic Techniques

The success of the GA would suggest that perhaps simulated annealing (SA) [6] and even possibly hill climbing (HC) might also be good at finding the correct last rotor of a three rotor machine. We experimented with both, using our previously defined mutation operators as neighbourhood definitions, and found that SA and HC did not, in fact, perform very well. We attempted SA and HC five times on each plaintext with 4732 letters for 500000 proposed moves. SA found the correct last rotor once with plaintext 1 and not at all with the other plaintexts, and HC never found the correct last rotor. Both methods tended to converge to a solution with less than 14 wirings correct, with SA generally outperforming HC. Further experimentation with different neighbourhood operators and cooling schedules (we use geometric cooling with tempering to 40% of the previous maximum) for SA may produce better results.

6 CONCLUSIONS

We have shown that a very basic implementation of a GA can be used to solve a fairly hard cryptographic system using a simple model of a language. By using more complex models, for example models based on observed two letter frequencies or utilising Markov chains, and more complex measures of fitness, possibly based on the test statistics for language recognition described in [4], we would expect an improvement in the performance of the GA. The choice of operators was dictated by the fitness landscape. As previously mentioned there are in fact q correct solutions for the last rotor and each of these solutions has identical adjacency information. For example, if one correct last rotor for a four letter alphabet is {0,2,1,3}, then the other correct solutions are {3,0,2,1}, {1,3,0,2} and {2,1,3,0}. By fixing the first wiring we are restricting ourselves to looking for the first solution only (a restriction justified by the reduction in the search space), and are therefore interested in maintaining positional information from the parents. With this in mind we use PMX crossover. However, PMX will not utilize the obvious adjacency information, and so the shift mutation operator was introduced. This is probably not the most effective way of combining the positional and adjacency information, but it worked up to the limit of the feasibility of the fitness measure. With less than 3380 letters of ciphertext each monoalphabetically enciphered string has 5 or less letters, and generally there are rotors with better fitness than the correct last rotor.

We also implemented a GA to search the space of all three rotors, using the likelihood function as our fitness measure. The nature of the problem suggested that this approach would not work. The interdependence of the rotors meant that solutions close to the correct rotors will not necessarily be particularly fit. For example a solution with the first two rotors correct could have a low fitness if the last rotor is not close to the correct solution. This is contrary to the idea of building blocks described by [5]. This implementation did work with a two rotor problem in times comparable to those obtained using the iterative technique, but this was probably because of the easier nature of the problem and the lesser importance of rotor interdependence. As we expected, the GA failed to find the correct solution to the three rotor problem with as many as 10000 letters for all 4 plaintexts.

Our preliminary experiments with the cryptanalysis of a four rotor machine using the GA to search the space of the last two rotors have been unsuccessful. Again, the reason for this is probably interdependence of the rotors. An alternative fitness measure or the use of a more sophisticated GA may produce some progress with this problem.

The applications of GAs and other heuristics in cryptanalysis is always going to be limited. An ideal cryptographic system is one that is defined on a landscape which is all noise except for a single peak, the key. An implementation of any search technique using the obvious representation to search the keyspace of any reasonable cryptographic system will most likely fail. However the possibilty of finding some transformation which produces a searchable space is something cryptanalysts and cryptographers should be aware of. It is possible that some progress may be made in the cryptanalysis of block ciphers by finding some transformation of the search space and suitable fitness measure which, although precluding enumerative search, allows a GA some chance of success.

References

[1] D. Andelman and J. Reeds. On the cryptanalysis of rotor machines and substitution-permutation networks. *IEEE Transactions on Information Theory*, IT-28(4), 1982.

[2] A. Deavours and L. Kruh. *Machine Cryptography and Modern Cryptanalysis*. Artech House, 1985.

[3] R. Spillman et al. Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers. *Cryptologia*, 17(1), 1993.

[4] R. Ganesan and A. Sherman. Statistical techniques for language recognition: An empirical study using real an simulated english. *Cryptologia*, 18(4), 1994.

[5] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[6] S. Kirkpatrick, C.D.Gellat, and M.P.Vecchi. Optimization by simulated annealing. *Science*, 220, 1983.

[7] Alan G. Konheim. *Cryptography:A Primer*. John Wiley & Sons, 1981.

[8] S. Kullback. *Statistical Methods in Cryptanalysis*. Aegean Park Press, 1976.

[9] J. W. Mann, A. Kapsalis, and G. D. Smith. The GAmeter toolkit. In V. J. Rayward-Smith, editor, *Applications of Modern Heuristic Methods*. Alfred Waller, 1995.

[10] S. Martello and P. Toth. *Knapsack Problems*. John Wiley & Sons, 1990.

[11] R.A.J. Mathews. The use of genetic algorithms in cryptanalysis. *Cryptologia*, 17(4), 1993.

[12] R. Spillman. Cryptanalysis of knapsack ciphers using genetic algorithms. *Cryptologia*, 17(1), 1993.

[13] A. Stuart and J. K. Ord. *Kendall's Advanced Theory of Statistics*. Edward Arnold, 1991.