Optimisation Heuristics for the Automated Cryptanalysis of Classical Ciphers

Andrew Clark and Ed Dawson
Information Security Research Centre
Queensland University of Technology
GPO Box 2434, Brisbane 4000, Queensland, Australia
tt aclark,dawson@isrc.qut.edu.au

Dedicated to Anne Penfold Street.

Abstract

This paper presents a comparison of the performance of three optimisation heuristics in automated attacks on a simple classical cipher. The three optimisation heuristics considered are simulated annealing, the genetic algorithm and the tabu search. Although similar attacks have been proposed previously, a comparison of a number of the techniques has not been performed. Performance criteria such as efficiency and speed are investigated. The use of the tabu search in the field of automated cryptanalysis is a largely unexplored area of research. A new attack on the simple substitution cipher which utilises the tabu search is also presented in this paper.

1 Introduction

Classical ciphers were first used hundreds of years ago. So far as security is concerned, they are no match for today's ciphers, however, this does not mean that they are any less important to the field of cryptology. Their importance stems from the fact that most of the ciphers in common use today utilise the operations of the classical ciphers as their building blocks. For example, the Data Encryption Standard (DES) [21], an encryption algorithm used widely in the finance community throughout the world, uses only three very simple operators, namely substitution, permutation (transposition) and bit-wise exclusive-or (admittedly, in a complicated fashion).

Given their simplicity, and the fact that they are used to construct other ciphers, the classical ciphers are usually the first ones considered when researching new attack techniques such as the ones discussed here.

Many flavours of classical ciphers exist, although most fall into one of two broad categories: substitution ciphers and transposition (permutation) ciphers. In this paper a number of combinatorial optimisation algorithms are used in attacks on the simple substitution cipher which is described in Section 3. Optimisation algorithms such as simulated annealing, the genetic algorithm have shown promise in the area of automated cryptanalysis. Forsyth and Safavi-Naini (in [6]) have published an attack on the simple substitution cipher using simulated annealingand Spillman et al (in [20]) presented an attack (again, on the simple substitution cipher) using a genetic algorithm. Also, an attack on the transposition cipher was proposed by Matthews (in [14]) using a genetic algorithm.The attacks on the simple substitution cipher have been re-implemented in the research presented in this paper in order to obtain a comparison of the techniques and also to evaluate a third technique, namely the tabu search. This paper introduces a new attack on the simple substitution cipher which utilises the tabu search. The previously published attacks were enhanced and modified in order that an accurate comparison of the three techniques could be obtained. Descriptions of the three techniques: simulated annealing, the genetic algorithm and the tabu search are given in Section 2. Attacks on the simple substitution cipher are described in detail in Section 3.

Each of the three techniques was compared based on three criteria: the amount of known ciphertext available to the attack; the number of keys considered before the correct solution was found; and the time required by the attack to determine the correct solution. The results are presented graphically in order to achieve a clear comparison. The results for the attack on the simple substitution cipher are given in Section 4.

Note that all experiments presented in this paper were performed on text using a 27 character alphabet, i.e., A – Z and the space character. All punctuation and structure (sentences/paragraphs) has been removed from the text before encryption. Any two words are separated by a single space character.

2 Combinatorial Optimisation Heuristics

The aim of combinatorial optimisation is to provide efficient techniques for solving mathematical and engineering related problems. Often it is impractical to use exact algorithms (ones which guarantee to find the optimal solution) because of their prohibitive complexity (time or memory requirements). In such cases approximate algorithms are employed in an attempt to find an adequate solution to the problem. Examples of approximate heuristics are simulated annealing, the genetic algorithm and the tabu search. Each of these techniques is now described in some detail.

2.1 Simulated Annealing

Simulated annealing is based on the concept of annealing. In physics, the term annealing describes the process of slowly cooling a heated metal in order to attain a "minimum energy state". A heated metal is said to be in a state of "high energy".The molecules in a metal at a sufficiently high temperature move freely with respect to each other, however, when the metal is cooled, the molecules lose their thermal mobility. If the metal is cooled slowly, a "minimum energy state" will be achieved. If, however, the metal is not cooled slowly, the metal will remain in an intermediate energy state and will contain imperfections. For example, "quenching" is the process of cooling a hot metal very rapidly. Metal that has been quenched commonly has the property that it is brittle because of the unordered structure of its molecules. In order to apply the analogy of annealing in physics to the field of combinatorial optimisation it is useful to think of the slowly cooled metal as having reached a crystalline structure in which the molecules are ordered and the energy is low. This is analogous to the optimal solution to a problem which is "ordered" and represents the lowest "cost" to solve the problem being optimised (assuming, of course, that the minimum cost is sought).

In 1953, Metropolis et al [15], showed that the distribution of energy in molecules at the "minimum energy state" is governed by the Boltzmann probability distribution. This discovery was applied to determine the probability of molecules moving between different energy levels, which depends upon the temperature of the metal and the difference in the energy levels. The molecule undergoes a transition from energy level E_1 to energy level E_2 ($\Delta E = E_2 - E_1$); the temperature of the metal is T; and Boltzmann's constant is k. If $\Delta E < 0$ the transition always occurs, otherwise it occurs with the probability indicated by Equation 1.

$$\Pr(E_1 \Rightarrow E_2) = e^{\left(\frac{-\Delta E}{kT}\right)} \tag{1}$$

The idea of mimicking the annealing process to solve combinatorial optimisation problems is attributed to Kirkpatrick et al [13], who, in 1983, used such an idea to find solutions to circuit wiring and component placement problems (from an electronic engineering perspective) and also to the travelling salesman problem (a classic combinatorial optimisation problem). The algorithm is (usually) initialised with a random solution to the problem being solved and a starting temperature. The choice of the initial temperature, T_0, is discussed below. At each temperature a number of attempts are made to perturb the current solution. Each proposed perturbation must be accepted by determining the change in the evaluation of the objective function (i.e., the change in the cost) and then consulting Metropolis' equation (Equation 1) which makes a decision based on this cost difference and the current temperature. If the proposed change is accepted then the current solution is updated. The technique used to perturb a solution is dependent on the problem being solved and the representation of the solution. For example, if the solution is represented as a binary string of fixed length, then a suitable mechanism for suggesting possible new solutions may be to complement one (or more) randomly chosen values in the bit string. Generally, the temperature is reduced when either there a predefined limit in the number of updates to the current solution has been reached or after a fixed number of attempts have been made to update the current solution. Methods for reducing the temperature are discussed below. The algorithm finishes either when no new solutions were accepted for a given temperature, or when the temperature has dropped below some predefined limit.

The variable parameters of the algorithm make up what is known as the *cooling schedule* which defines: the initial temperature, T_0; the number of iterations at each temperature, J; the temperature reduction scheme; and the stopping criteria. Each of these is now investigated in some detail.

- **The Initial Temperature.** Usually T_0 is chosen so that all candidate solutions proposed will be accepted. Such a choice of T_0 is dependent upon the magnitude of typical cost values for the problem being solved, or, more specifically, the magnitude of the largest expected cost difference between two solutions to the problem being solved. To ensure that the probability that the transition occurs is close to one, T_0 must be significantly greater than the largest expected cost difference (as shown by the following equations).

$$\begin{aligned} Pr(E_1 \Rightarrow E_2) \rightarrow 1 \quad &\Leftrightarrow \quad e^{\left(\frac{-\Delta E}{T}\right)} \rightarrow 1 \\ &\Leftrightarrow \quad \frac{\Delta E}{T} \rightarrow 0 \\ &\Leftrightarrow \quad \Delta E \ll T \end{aligned}$$

- **Iterations For Each T.** The number of iterations at each temperature is equivalent to the number of candidate solutions considered for each temperature. This value should be large but not so large that the performance of the algorithm is hindered. One value suggested in the literature is $100N$, where N is the number of variables in the problem being solved.

- **Temperature Reduction.** In theory, the rate at which temperature dissipates from a metal is governed by complicated differential equations. For the purposes of simulated annealing, two simple models are most commonly used. The first, and most simplistic, is a linear cooling model. In the linear model both an initial temperature (T_0) and a final temperature (T_∞) must be defined. The difference between these two ($T_0 - T_\infty$) is then divided by J to determine how much the temperature is reduced by. The temperature at iteration k is defined by Equations 2 and 3.

$$\begin{aligned} T_k &= T_0 - k \cdot \left(\frac{T_0 - T_\infty}{J}\right) && (2) \\ &= T_{k-1} - \left(\frac{T_0 - T_\infty}{J}\right) && (3) \end{aligned}$$

The second method of temperature reduction is exponential decay. This is a more accurate model of the true thermal dynamics in a heated metal than the linear model. At each iteration the temperature is reduced by multiplying with a factor, $\lambda < 1$. Equations 4 and 5 describe the temperature at iteration k.

$$\begin{aligned} T_k &= \lambda^k \cdot T_0 && (4) \\ &= \lambda \cdot T_{k-1} && (5) \end{aligned}$$

Because of its closer approximation to the expected temperature reduction in a practical sense, the exponential method is used in all work reported here.

- **Stopping Criteria.** The stopping criteria define when the algorithm should terminate. Possibilities are: a minimum temperature (eg, T_∞) has been reached; a certain number of temperature reductions have occurred; or the current solution has not changed for a number of iterations. The latter is often the best option as any candidate solutions will not be accepted (unless they are better than the current one) when the temperature is very low. This is because the probability of acceptance (as defined by Equation 1) is negligible.

2.2 The Genetic Algorithm

As may be evident from the simulated annealing algorithm, mathematicians often look to other areas in search of inspiration for new techniques which can be modelled for the purpose of optimisation. While simulated annealing is derived from the field of chemical physics, the genetic algorithm is based upon another "scientific" notion, namely Darwinian evolution theory. The genetic algorithm is modelled on a relatively simple interpretation of the

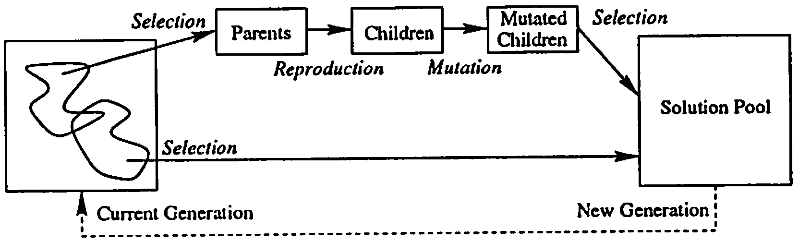

Figure 1: The Evolutionary Process.

evolutionary process, however, it has proven to be a reliable and powerful optimisation technique in a wide variety of applications.

It was Holland [11] in his 1975 paper, who first proposed the use of genetic algorithms for problem solving. Goldberg [9] and DeJong [5] were also pioneers in the area of applying genetic processes to optimisation. Over the past twenty years numerous applications and adaptations have appeared in the literature. Three papers containing applications to the field of cryptanalysis are worthy of mention here. The first, by Matthews [14] and the second, by Spillman et al [19], are used in attacks on the transposition cipher and the substitution cipher, respectively. The third paper, by Spillman [19], attempts an ill-fated attack on the knapsack cipher (see also [3]).

Consider a pool of genes which have the ability to reproduce, are able to adapt to environmental changes and, depending on their individual strengths, have varying life-spans. In such an environment only the fittest will survive and reproduce giving, over time, genes that are stronger and more resilient to conditional changes. After a certain amount of time the surviving genes could be considered "optimal" in some sense. This is the model used by the genetic algorithm, where the gene is the representation of a solution to the problem being optimised. Traditionally, genetic algorithms have solutions represented by binary strings. However, not all problems have solutions which are easily represented in binary (especially if the structure of the binary string is to be "meaningful"). To avoid this limiting property a more general area known as *evolutionary programming* has been developed. An evolutionary program may make use of arbitrary data structures in order to represent the solution. For simplicity all algorithms described in this thesis which use the evolutionary heuristics presented in this section are referred to as "genetic algorithms", although, from a purist's perspective, this may not be strictly accurate.

As with any optimisation technique there must be a method of assessing each solution. In keeping with the evolutionary theme, the assessment technique used by a genetic algorithm is usually referred to as the "fitness function". As was pointed out above, the aim is always to maximise the fitness of the solutions in the solution pool.

Figure 1 gives an indication of the evolutionary processes used by the genetic algorithm, During each iteration of the algorithm the processes of selection, reproduction and mutation each take place in order to produce the next generation of solutions. The actual method used to perform each of these operations is very much dependent upon the problem being solved and the representation of the solution. For the purposes of illustration, examples of each of these operations are now given using the traditional binary solution structure.

Consider a problem whose solutions are represented as binary strings of length $N = 7$. In this instance, a pool of M solutions is being maintained. The first phase of each iteration is the selection of a number of parents who will reproduce to give children.

- **Selection of parents.** A subset of the current solution pool is chosen to be the "breeding pool". This could be a random subset of the current solution pool, or in fact the entire current generation, or some other grouping. Another technique is to make the choice pseudo-randomly by giving the most fit solutions a higher likelihood of being selected, thus making the "better" solutions more likely to be involved in the creation of the new generation while at the same time not prohibiting the less fit solution from being involved in the breeding process.

Once the breeding pool has been created, parents are paired for the reproduction phase.

- **Reproduction.**A commonly used mating technique for solutions represented as a binary string is the "crossover" where a random integer in the range $[1, \ldots, N-1]$ is generated and all bits in the binary string after this position are swapped between the two parents. Consider the two parents P_1 and P_2 with the random chosen position 3.

$$\begin{array}{lccccccc} P_1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ P_2 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{array}$$

The two children created by this operation are C_1 and C_2.

$$\begin{array}{lccccccc} C_1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ C_2 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{array}$$

As can be seen, each of the children has inherited characteristics from each of its parents.

Finally, the newly generated children undergo mutation. Here, the solutions are randomly adjusted in a further attempt to increase the diversity of the new solution pool.

- **Mutation.**The most simple mutation operation for binary strings is complementation of some of the bits in the child. The probability that a bit is complemented is given by the "mutation probability", p_m. For example, if $p_m = 0.15 \approx \frac{1}{7}$, for the case when $N = 7$, one would expect that, on the average, one bit of each child would be complemented. If bit 3 of C_1 were complemented then C_1 would become as follows.

$$C_1 \quad 1\ 0\ 0\ 0\ 1\ 0\ 1$$

2.3 The Tabu Search

The final method for optimisation which is discussed in this paper is the tabu search. The use of the tabu search was pioneered by Glover who from 1985 onwards has published many articles discussing its numerous applications (for examples see [7, 8]). Others were quick to adopt the technique which has been used for such purposes as sequencing [17], scheduling [1, 4, 16, 2], oil exploration [10] and routing [18, 2].

The properties of the tabu search can be used to enhance other procedures by preventing them from becoming stuck in the regions of local minima. The tabu search, like the genetic algorithm, introduces memory structures into its workings. In this case the purpose of the memory is multi-faceted. The genetic algorithm utilises its solution pool as a mechanism for introducing diversity into the breeding process. The tabu search utilises memory for an additional purpose, namely to prevent the search from returning to a previously explored region of the solution space too quickly. This is achieved by retaining a list of possible solutions that have been previously encountered. These solutions are considered *tabu* - hence the name of the technique. The size of the tabu list is one of the parameters of the tabu search.

The tabu search also contains mechanisms for controlling the search. The tabu list ensures that some solutions will be unacceptable, however, the restriction provided by the tabu list may become too limiting in some cases causing the algorithm to become trapped at a locally optimum solution. The tabu search introduces the notion of *aspiration criteria* in order to overcome this problem. The aspiration criteria over-ride the tabu restrictions making it possible to broaden the search for the global optimum.

Much of the implementation of the tabu search is problem specific - i.e., the mechanisms used depend heavily upon the type of problem being solved. An initial solution is generated (usually randomly). The tabu list is initialised with the initial solution. A number of iterations are performed which attempt to update the current solution with a better one, subject to the restrictions of the tabu list. In each iteration a list of candidate solutions is proposed. These solutions are obtained in a similar fashion to the perturbation technique used in simulated annealing and the mutation operation used in the genetic algorithm. The most admissible solution is selected from the candidate list. The current solution is updated with the most admissible one and the new current solution is added to the tabu list. The algorithm stops after a fixed number of iterations or when a better solution has been found for a number of iterations.

3 Attacks on the Simple Substitution Cipher

The general strategy with a substitution cipher is to substitute symbols from the plaintext alphabet with different symbols from the ciphertext alphabet. The weakness with this strategy is that character frequency distributions are not significantly altered by the encryption process. Thus, most attacks on substitution ciphers attempt to match the character frequency statistics of the encrypted message with those of some known language (for example, English). Character frequency statistics (or n-grams) indicate the frequency distribution of all possible instances of n adjacent characters (for example, THE is a very common 3-gram (or *trigram*) in the English language).

The attack on the simple substitution cipher is particularly simple since the frequency of any n-gram in the plaintext (or unencrypted) message will correspond *exactly* to the frequency of the corresponding encrypted version in the ciphertext. The search for the corresponding n-gram frequencies can be automated using combinatorial optimisation algorithms. Here, a number of methods are utilised in attacks on the simple substitution cipher. As previously indicated, a method of assessing intermediate solutions (in the search for the optimum) is required.

A major factor influencing the success of an attack on the simple substitution cipher (or any cipher where the attack is based on n-gram statistics of the language) is the length of the intercepted ciphertext message which is being cryptanalysed. The amount of ciphertext required in order to recover the entire key (with a high degree of certainty) varies depending on the type of cipher. From Figure 2 it will be seen that for a message of 1000 characters it is possible to recover 26 out of the 27 key elements on the average. Note that in practice it is impossible to find a simple substitution cipher key which differs in exactly one place from the correct key. However it is not necessary to recover every element of the key in order to obtain a

		Frequency (%)				Frequency (%)	
Order	Letter	Relative	Cumulative	Order	Letter	Relative	Cumulative
1	_	18.4820	18.4820	15	M	1.9853	89.0537
2	E	10.3320	28.8140	16	F	1.9242	90.9778
3	T	7.8395	36.6535	17	W	1.9183	92.8961
4	A	6.6284	43.2819	18	P	1.5438	94.4399
5	O	6.0091	49.2909	19	G	1.4424	95.8823
6	I	5.7941	55.0850	20	Y	1.2656	97.1479
7	N	5.7526	60.8376	21	B	1.2026	98.3505
8	S	5.3997	66.2373	22	V	0.7474	99.0979
9	H	4.8210	71.0583	23	K	0.5482	99.6461
10	R	4.5744	75.6327	24	X	0.1466	99.7928
11	D	3.4530	79.0857	25	Q	0.0851	99.8779
12	L	3.2366	82.3223	26	J	0.0667	99.9445
13	U	2.4719	84.7941	27	Z	0.0555	100.0000
14	C	2.2742	87.0683				

Table 1: English language characteristics.

message that is readable. Table 1 shows the relative frequency of each of the characters in a sample of English taken from the book "20000 Leagues Under the Sea" by Jules Verne. A fact evident in almost all attacks on substitution ciphers is that the most frequent characters are decrypted first. Table 1 also shows the amount of message that will have been recovered after each of the characters is discovered (making the somewhat unrealistic assumption that the characters are discovered in order from most frequent to least frequent). It can be seen that approximately fifty percent of the message can be recovered by correctly determining the key element for the five most frequent characters in the intercepted message. Also, the eleven most infrequent characters account for only ten percent of the message (on the average).

3.1 Suitability Assessment

Naturally, the technique used to compare candidate keys to the simple substitution cipher is to compare n-gram statistics of the decrypted message with those of the language (which are assumed known). Equation 6 is a general formula used to determine the suitability of a proposed key (k) to a simple substitution cipher. Here, $\mathcal{A}$ denotes the language alphabet (i.e., for English, {A, ..., Z, _}, where _ represents the space symbol), 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.

$$C_k = \alpha \cdot \sum_{i \in \mathcal{A}} \left| K^u_{(i)} - D^u_{(i)} \right|$$

$$+ \quad \beta \cdot \sum_{i,j \in \mathcal{A}} \left| K^b_{(i,j)} - D^b_{(i,j)} \right| \tag{6}$$
$$+ \quad \gamma \cdot \sum_{i,j,k \in \mathcal{A}} \left| K^t_{(i,j,k)} - D^t_{(i,j,k)} \right|$$

Forsyth and Safavi-Naini, in their simulated annealing attack on the substitution cipher [6]and Jakobsen in his attack [12] use a very similar evaluation, namely the one in Equation 7. This formula is based purely upon bigram statistics which is often sufficient for attacks on the simple substitution cipher.

$$C_k \quad = \quad \sum_{i,j \in \mathcal{A}} \left| K^b_{(i,j)} - D^b_{(i,j)} \right| \tag{7}$$

Spillman et al [20], use a different formula again (see Equation 8). This equation is based on unigram and bigram statistics.

$$C_k \quad \propto \quad \sum_{i \in \mathcal{A}} \left| K^u_{(i)} - D^u_{(i)} \right| + \sum_{i,j \in \mathcal{A}} \left| K^b_{(i,j)} - D^b_{(i,j)} \right| \tag{8}$$

The only difference between these assessment functions is the inclusion of different statistics (Equation 7 is equal to Equation 6 when $\alpha = \gamma = 0$). In general, the larger the n-grams, the more accurate the assessment is likely to be. It is usually an expensive task to calculate the trigram statistics - this is, perhaps, why they are omitted in Equations 7 and 8. The complexity of determining the fitness is $O(N^3)$ (where N is the alphabet size) when trigram statistics are being determined, compared with $O(N^2)$ when bigrams are the largest statistics being used. Following the attack description given below there are details describing a method which can be used under some circumstances for reducing the complexity of the cost calculation by a factor of N. Thus a cost based on trigram statistics can be calculated with complexity proportional to $O(N^2)$.

Figure 2 indicates the effectiveness of using different n-grams in the evaluation of a simple substitution cipher key. The three curves in the plot represent the percentage of key recovered by an attack on a simple substitution cipher using simulated annealing (see below) versus the amount of known ciphertext used in the attack. Each curve resulted from using the cost function in Equation 6 with different values of the weights α, β and γ. For example, the "Unigrams only" curve was obtained with $\alpha = 1$ and $\beta = \gamma = 0$. Each data point on each curve was determined by running the attack on 200 different messages and three times for each message. Of the three runs for each message only the best result was used. The value on the curve represents the average number of key elements correctly placed

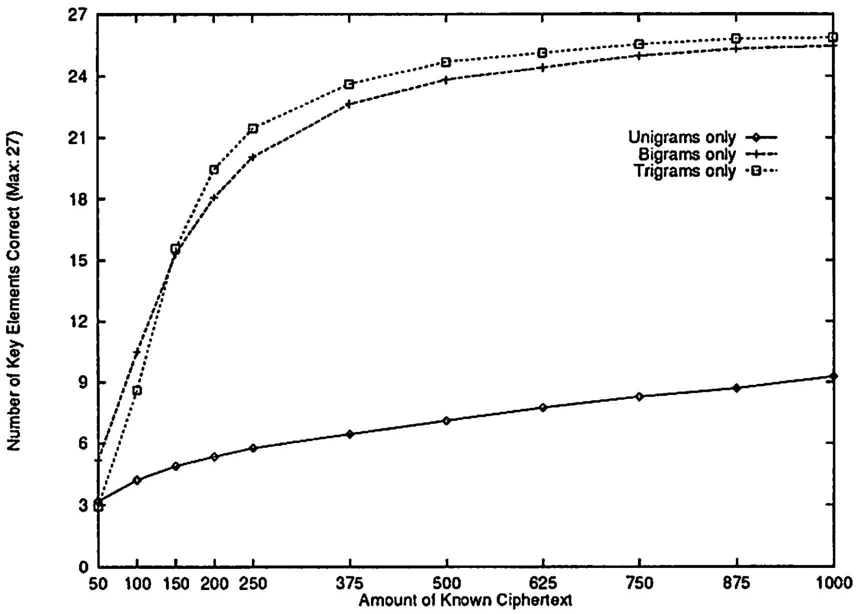

Figure 2: Results for cost functions using only unigrams, bigrams and trigrams, respectively.

(over the 200 messages). For the alphabet being used the maximum value attainable is 27.

For small amounts of known ciphertext it is interesting to note that an attack using a cost function based on bigrams alone is more effective than one which utilises only trigrams (see Figure 2). The crossover point of the "Bigrams only" and "Trigrams Only" curves in Figure 2 represents an approximate threshold value where a cost function based purely upon trigram frequencies out-performs one based only on bigram frequencies. The reason for this phenomenon can, in part, be gleaned by observing the cost function in Equation 6. When the length of the intercepted messages is short there are far fewer distinct bigrams or trigrams represented in the ciphertext than the total number of possible bigrams or trigrams. When $N = 27$ there are $N^3 = 19683$ possible trigrams (theoretically - of course not all trigrams are represented in the English language). The maximum number of distinct trigrams in a message of length say, 100, is 98. Thus the proportion of trigrams represented in the message is very small (the upper bound is $98/19683 \approx 0.005$). This means that are large number of unrepresented trigrams are effecting the evaluation of the cost function.

For bigrams the proportion is much larger ($99/729 \approx 0.136$) and hence the cost function is more reliable and accurate. This effect ceases when the amount of known ciphertext is greater than approximately 150 characters (see Figure 2).

While Figure 2 gives an interesting comparison of the effectiveness of each of the statistic types (i.e., unigrams, bigrams and trigrams), it is also interesting to experiment with different values of α, β and γ in order to determine how the statistics interact and in which proportions they work best. Figure 3 presents a comparison of different values of α, β and γ.

In the experiments used to produce Figure 3, the following restrictions were applied in order to keep the number of combinations of α, β and γ workable.

$$\begin{aligned}\alpha, \beta, \gamma &\in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}, \text{ and} \\ \alpha + \beta + \gamma &= 1.0\end{aligned}$$

There are 66 combinations of α, β and γ which satisfy these conditions. In order to obtain good statistical averages the same attack procedure was used as for Figure 2 except that only 100 different messages were used (instead of 200). The three curves represent the average number of key elements correctly placed for each of the eleven values of each of the three weights. (NB. "Unigram Weight" refers to α, "Bigram Weight" refers to β and "Trigram Weight" refers to γ.)

From Figures 2 and 3 it can be concluded that trigrams are generally the most effective basis for a cost function used in attacks on the substitution cipher. However, the benefit obtained from trigrams over bigrams is minimal. In fact,because of the complexity associated with determining trigram statistics - $\mathcal{O}(N^3)$ - it is often more practical to base a fitness of a mixture of unigrams and bigrams - which has a complexity proportional to $\mathcal{O}(N^2)$. For the remainder of the work relating to simple substitution ciphers a fitness based purely on bigrams is used (i.e., Equation 7).

3.2 A Simulated Annealing Attack

The simulated annealing attack of the simple substitution cipher is relatively straight-forward. Recall that the key is represented as a string of the N characters in the alphabet. A very simple way of perturbing such a key is to swap the key elements in two randomly chosen positions. This is the method utilised in the following algorithm which describes a simulated annealing attack on the simple substitution cipher.

1. The algorithm is given the intercepted ciphertext and the known language statistics as input.

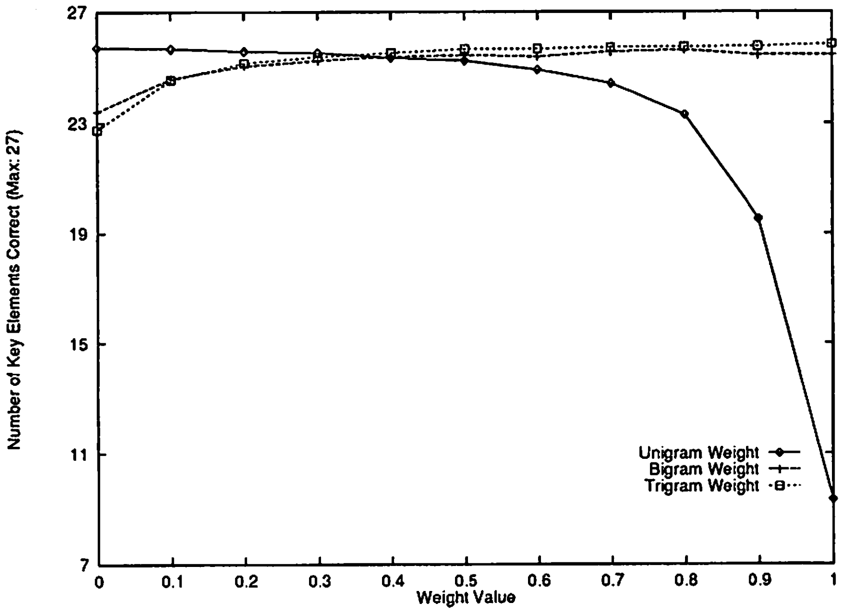

Figure 3: Results for cost functions with varying weights.

2. Generate the initial solution (randomly or otherwise), K_{CURR}, and calculate its cost using Equation 7 (C_{CURR}). Set $T = T_0 = C_{\text{CURR}}$ and the temperature reduction factor T_{FACT}. Set MAX_ITER, the maximum number of iterations to perform.
3. Repeat MAX_ITER times (MAX_ITER temperature reductions):
   (a) Set $N_{\text{SUCC}} = 0$.
   (b) Repeat $100 \cdot N$ times:
      i. Choose $n_1, n_2 \in [1, N], n_1 \neq n_2$.
      ii. Swap element n_1 with element n_2 in K_{CURR} to produce K_{NEW}.
      iii. Calculate the cost of K_{NEW} using Equation 7. Call this cost C_{NEW}. Calculate the cost difference ($\Delta E = C_{\text{NEW}} - C_{\text{CURR}}$) and consult the Metropolis criterion to determine whether the proposed transition should be accepted.
      iv. If the transition is accepted set $K_{\text{CURR}} = K_{\text{NEW}}$ and $C_{\text{CURR}} = C_{\text{NEW}}$ and increment N_{SUCC} ($N_{\text{SUCC}} = N_{\text{SUCC}} + 1$). If $N_{\text{SUCC}} > 10 \cdot N$ go to Step 3d.
   (c) If $N_{\text{SUCC}} = 0$ go to Step 4.
   (d) Reduce T ($T = T \times T_{\text{FACT}}$).
4. Output the current solution. .

The choice of $T_0 = C_{\mathrm{CURR}}$ was made based upon experimentation. It was found that for this choice of T_0 the necessary conditions are satisfied almost all of the time. In fact, this technique of choosing T_0 was found to be better than using a constant value since many times, when the choice of T_0 is too high, the algorithm spends a lot of time in a seemingly random search while the temperature decreases to a value which disallows some of the proposed solutions.

The values $100 \cdot N$ (Step 3b) and $10 \cdot N$ (Step 3(b)iv) are parameters of the algorithm which are arbitrary. They should, of course, be chosen so that a large number of possible solutions are assessed at each temperature, but not so large that time is wasted. By setting a limit to the number of successful updates to the solution at each temperature, the algorithm avoids assessing too many solutions when the temperature is high (since almost all suggested transitions are accepted at high temperatures).

As indicated above, there are circumstances under which it is possible to reduce the complexity of the cost calculation. One such case is when comparing the cost associated with two keys when the keys only differ in two elements (i.e., one key can be obtained from the other key simply by swapping two elements). This is exactly what happens in the simulated annealing algorithm above - two elements in the current key are swapped and the new cost calculated. The difference between the cost of the current solution and the new one (with the swapped elements) is used to determine if the new key will be accepted.

It should be clear that when two key elements are swapped only the statistics (of the decrypted message) for trigrams which contain one (or both) of the swapped elements will change. Similarly, only the statistics for bigrams which contain one (or both) of the swapped elements will change. Also, only two of the unigram statistics will change.

Making use of this property can lead to a significant increase in the efficiency of the simulated annealing attack. The improvement obtained using such a strategy is of the order of N. Table 2 displays the actual improvement in terms of the number of comparisons of statistics required in the evaluation of the cost difference. The numbers in brackets indicate the number of calculations required when $N = 27$. It can be seen that when $N = 27$ the complexity of determining a cost based on trigrams is not significantly reduced using this technique. However, for larger values of N - for example consider the ASCII alphabet of 256 characters - the saving is great.

This technique can be used for determining the cost difference between two keys in any system where one key is obtained from another by swapping two elements of the original key.

The results of the attack described in this section are compared with the attacks using the genetic algorithm and tabu search (described below), in Section 4.

Statistic	Exhaustive	Optimised
Unigrams	N (27)	4
Bigrams	N^2 (729)	$8(N-1)$ (208)
Trigrams	N^3 (19683)	$8(3N^2-6N+4)$ (16232)

Table 2: Improvement gained from optimised calculation of the cost difference.

3.3 A Genetic Algorithm Attack

The genetic algorithm is rather more complicated than the simulated annealing attack. This is because a pool of solutions is being maintained, rather than a single solution. An extra level of complexity is also present because of the need for a mating function.

The mating function utilised in this thesis for attacks involving substitution ciphers is similar to the one proposed by Spillman et al [20], who use a special ordering of the key. The characters in the key string are ordered such that the most frequent character in the ciphertext is mapped to the first element of the key (upon decryption), the second most frequent character in the ciphertext is mapped to the second element of the key, and so on. The correct key will then be a sorted list of the decrypted message single character frequencies. The reason for this ordering will become apparent upon inspection of the mating function. Given two parents constructed in the manner just described, the first element of the first child is chosen to be the one of the first two elements in each of the parents which is most frequent in the known language statistics. This process continues in a right to left direction along each of the parents to create the first child only. If, at any stage, a selection is made which already appears in the child being constructed, the second choice is used. If both of the characters in the parents for a given key position already appear in the child then a character is chosen at random from the set of characters which do not already appear in the newly constructed child. The second child is formed in a similar manner, except that the direction of creation is from left to right and, in this case, the least frequent of the two parent elements is chosen. An algorithmic description of this mating procedure for creating the two children is now given:

1. **Notation:** p_1 and p_2 are the parents, c_1 and c_2 are the children, $p_i(j)$ indicates character j in parent i (similarly $c_i(j)$ indicates the jth element in child i), $\{C_i^{j,k}\}$ denotes the set of elements in child i in positions j to k (inclusive) with the limitation that if $i = N+1$ or $j = 0$ then $\{C_i^{j,k}\} = \{\emptyset\}$ (the empty set), $f(x)$ denotes the relative frequency of character x in the known language.

2. Child 1: For $j = 1, \ldots, N$ (step 1) do

```
    • If $f(p_1(j)) > f(p_2(j))$ then
        – If $p_1(j) \notin \{C_1^{1,j-1}\}$ then $c_1(j) = p_1(j)$,
          else if $p_2(j) \notin \{C_1^{1,j-1}\}$ then $c_1(j) = p_2(j)$,
          else $c_1(j)$ = random element $\notin \{C_1^{1,j-1}\}$.
      else
        – If $p_2(j) \notin \{C_1^{1,j-1}\}$ then $c_1(j) = p_2(j)$,
          else if $p_1(j) \notin \{C_1^{1,j-1}\}$ then $c_1(j) = p_1(j)$,
          else $c_1(j)$ = random element $\notin \{C_1^{1,j-1}\}$.
3. Child 2: For $j = N, \ldots, 1$ (step $-1$) do
    • If $f(p_1(j)) < f(p_2(j))$ then
        – If $p_1(j) \notin \{C_2^{j+1,N}\}$ then $c_2(j) = p_1(j)$,
          else if $p_2(j) \notin \{C_2^{j+1,N}\}$ then $c_2(j) = p_2(j)$,
          else $c_2(j)$ = random element $\notin \{C_2^{j+1,N}\}$.
      else
        – If $p_2(j) \notin \{C_2^{j+1,N}\}$ then $c_2(j) = p_2(j)$,
          else if $p_1(j) \notin \{C_2^{j+1,N}\}$ then $c_2(j) = p_1(j)$,
          else $c_2(j)$ = random element $\notin \{C_2^{j+1,N}\}$.
```

This description of the mating operation for a simple substitution cipher differs from the method described in [20] where each element of the two children is chosen by taking the character from the two parents which appears *most* frequently (for both children) in the *ciphertext*. This technique is clearly less efficient since the first element of each key represents the most frequent *plaintext* character.

The mutation operation is identical to the solution perturbation technique used in the simulated annealing attack. That is, randomly select two positions in the child and swap the two characters at those positions.

The following is an algorithmic description of the attack on a simple substitution cipher using a genetic algorithm.

1. The algorithm is given the ciphertext (and its length) and the statistics of the language (unigrams, bigrams and trigrams).
2. Initialise the algorithm parameters. They are: M the solution pool size and MAX_ITER the maximum number of iterations.
3. Randomly generate the initial pool containing M solutions (keys of the simple substitution cipher). Call this pool P_{CURR}. Calculate the cost of each of the keys using Equation 7.
4. For iteration/generation $i = 1, \ldots,$ MAX_ITER, do

(a) Select $M/2$ pairs of solutions from the current pool, P_{CURR}, to be the parents of the new generation. The selection should be random with a bias towards the most fit of the current generation.

(b) Each pair of parents then mate using the algorithm above to produce two children. These M children form the new pool, P_{NEW}.

(c) Mutate each of the children in P_{NEW} using the random swapping procedure described above.

(d) Calculate the suitability of each of the children in P_{NEW} using Equation 7.

(e) Sort P_{NEW} from most suitable (least cost) to lease suitable (most cost).

(f) Merge P_{CURR} with P_{NEW} to give a list of sorted solutions (discard duplicates) - the size of this list will be between M and $2M$. Choose the M best solutions from the merged list to become the new P_{CURR}.

5. Output the best solution from P_{CURR}.

This genetic algorithm was implemented and results of the attack on the simple substitution cipher are given in Section 4. The genetic algorithm attack is compared with the simulated annealing attack (described above) and the tabu search attack (described now).

3.4 A Tabu Search Attack

The simple substitution cipher can also be attacked using a tabu search. This attack is similar to the simulated annealing one with the added constraints of the tabu list. The same perturbation mechanism (i.e., swapping two randomly chosen key elements) is used. The overall algorithm is described as follows:

1. The inputs to the algorithm are the known (intercepted) ciphertext and the language statistics for unigrams, bigrams and trigrams.

2. Set MAX_ITER, the maximum number of iterations, N_TABU, the size of the tabu list and N_POSS, the size of the possibilities list. Initialise the tabu list with a list of random and distinct keys.

3. For iteration $i = 1, \ldots,$ MAX_ITER, do

   (a) Find the key in the tabu list which has the lowest cost associated with it. Call this key K_{BEST}.

   (b) For $j = 1, \ldots,$ N_POSS, do

      i. Choose $n_1, n_2 \in [1, N], n_1 \neq n_2$.

      ii. Create a possible new key K_{NEW} by swapping the elements n_1 and n_2 in K_{BEST}.

      iii. Check that K_{NEW} is not already in the list of possibilities for this iteration or the tabu list. If it is return to Step 3(b)i.

Number of Ciphertexts	SA		GA		TS	
	$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s
100	10.73	4.70	7.74	4.82	6.02	4.04
200	17.70	3.40	14.17	6.13	12.76	6.32
300	21.07	2.72	18.77	6.01	17.33	6.48
400	22.86	2.27	21.72	4.61	19.45	6.34
500	23.73	2.16	22.44	4.37	21.77	5.47
600	24.50	1.86	23.69	3.68	23.50	4.14
700	24.72	1.73	23.82	3.39	23.68	4.42
800	25.08	1.59	24.64	2.23	24.18	4.12

Table 3: Mean and standard deviation data corresponding to Figure 4.

```
        iv. Add $K_{\text{NEW}}$ to the list of possibilities for this iteration and determine its cost.
    (c) From the list of possibilities for this iteration find the key with the lowest cost – call this key $P_{\text{BEST}}$.
    (d) From the tabu list find the key with the highest cost – call this key $T_{\text{WORST}}$.
    (e) While the cost of $P_{\text{BEST}}$ is less than the cost of $T_{\text{WORST}}$:
        i. Replace $T_{\text{WORST}}$ with $P_{\text{BEST}}$.
        ii. Find the new $P_{\text{BEST}}$.
        iii. Find the new $T_{\text{WORST}}$.
4. Output the best solution (i.e., the one with the least cost) from the tabu list.
```

Note that the choice of N_POSS must be less than $N(N-1)$ – where N is the key size – since this is the maximum number of distinct keys which can be created from K_{BEST} by swapping two elements.

The results of the tabu search are given in the following section along with a comparison with the two alternate techniques described above.

4 Experimental Results

In this section a number of experimental results are presented which outline the effectiveness of each of the attack algorithms described above. Each of the attacks was run a number of times with a variety of parameter values. The results here are presented as plotted curves (Figures 4, 5 and 6). Each data point on these curves represents three hundred runs of the particular algorithm. One hundred different messages were generated in each case and the attack was run three times on each message. Of these three runs on each message only the best result is considered. The numerical value that is used

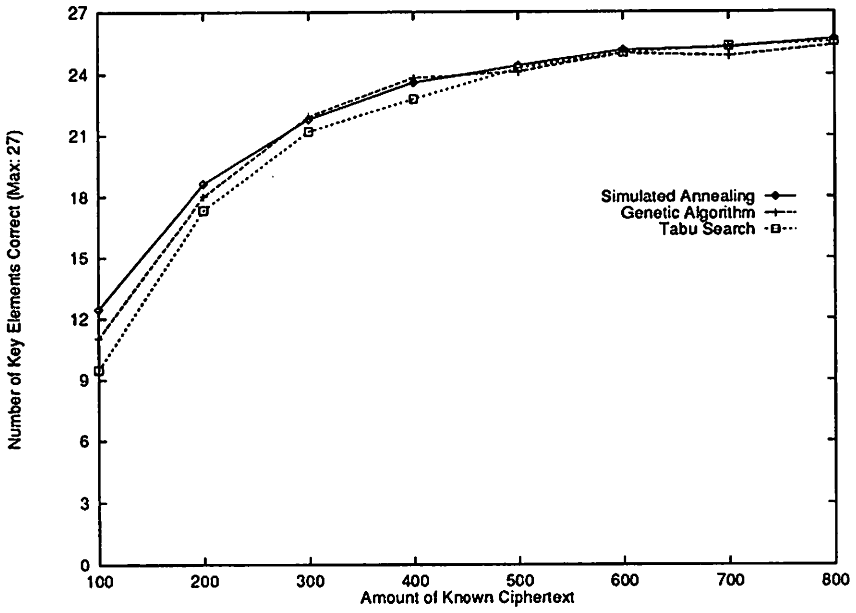

Figure 4: A comparison based on the amount of known ciphertext.

in each of the plots is then the average over the one hundred messages of the best result for each message. Why was this technique used?It is common when using approximation algorithms to run the algorithm a number of times and then take the best result. This is because of the random nature of the algorithms. By averaging the results of a large number of independent attacks (in this case one hundred) a good representation of the algorithm's ability is obtained.

The first point to note is that each of the algorithms performed (approximately) as well as the other with regard to the ultimate outcome of the attack. This is illustrated in Figure 4 which compares the average number of key elements (out of 27) correctly recovered versus the amount of ciphertext which is assumed known in the attack. The plot shows results for amounts ranging from 100 to 800 known ciphertext characters. In each case the results obtained are very similar for each of the algorithms. The mean, $\bar{x}$, and standard deviation values, s, for the results in Figure 4 are given in Table 3. It can be seen that the standard deviation values for simulated annealing are less than for the other two methods. This indicates that the simulated annealing approach, as well as being slightly superior with respect to the mean values, has less variance in its results.

Each of the algorithms perform roughly equally well when the compar-

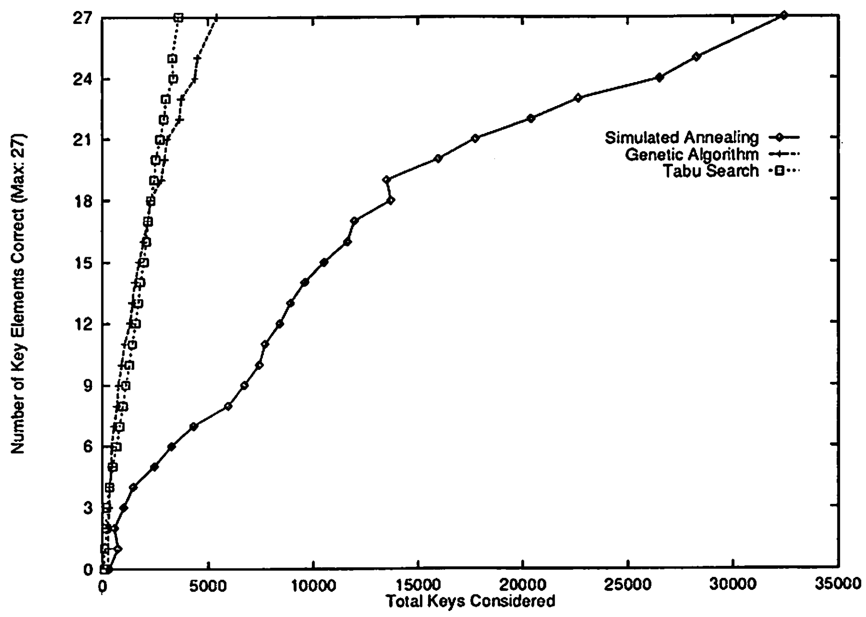

Figure 5: A comparison based on the number of keys considered.

ison is made based on the amount of ciphertext provided to the attack. It is interesting to make a comparison based on the complexity of each of the attack algorithms. The remaining results in this section aim to make a comparison of the algorithms based on complexity. In each case, the following results were obtained using cryptograms of length 1000. Figure 5 compares the number of correctly determined key elements with the total number of keys considered up to that point by the algorithm. It is clear that the number of keys considered by the simulated annealing technique in order to obtain the correct key is significantly greater than for the other two techniques with the genetic algorithm and the tabu search performing roughly equally in this respect. However, this comparison does not accurately indicate the relative efficiencies of the three algorithms. In Figure 6, which compares the number of correct key elements discovered with the amount of time used by the algorithm up to that point, it is clear that simulated annealing is actually more efficient (with respect to time) that the genetic algorithm. By comparing Figures 5 and 6 it can be concluded that the simulated annealing algorithm is able to consider a far greater number of solutions that the genetic algorithm and in much less time. It is also clear from both of these figures that the tabu search is most efficient in both respects.

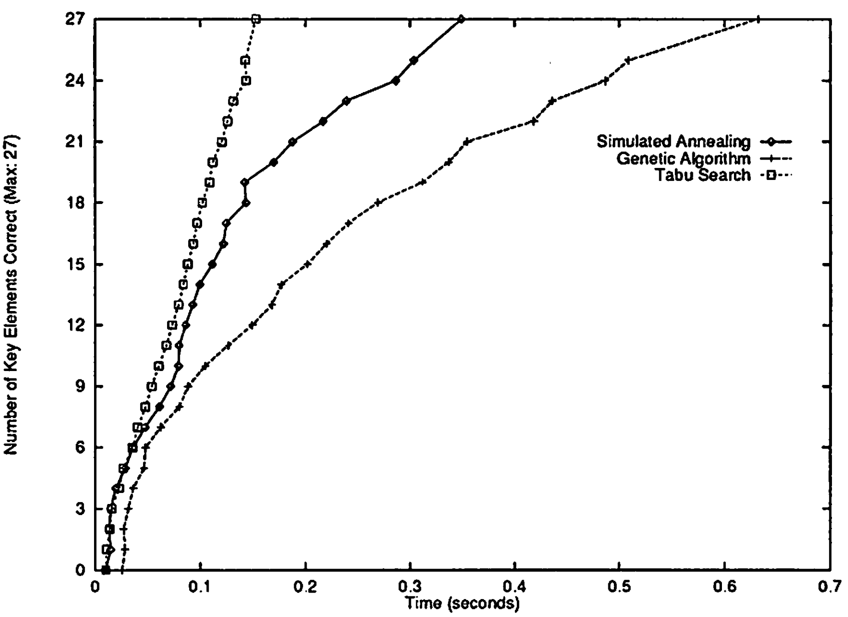

Figure 6: A comparison based on time.

5 Conclusions

This paper has developed the theory and presented a number of automated attack methodologies against simple substitution ciphers. In the first instance, properties of these ciphers which make them vulnerable were discussed. The common failing of the classical ciphers is that none is sophisticated enough to hide the inherent properties or statistics of the language of the plaintext.

Comparisons of the three techniques were also made based on the number of keys considered and the time taken by the algorithm. It was found that for the simple substitution cipher the tabu search out-performed both simulated annealing and the genetic algorithm with simulated annealing taking roughly twice as long as the tabu search and the genetic algorithm taking roughly twice as long as simulated annealing.

Overall, optimisation heuristics are ideally suited to implementations in attacks on the classic ciphers. This has been shown by the experimental results in the paper.

References

[1] J. Wesley Barnes and Manuel Laguna, A tabu search experience in production scheduling. *Ann. Oper. Res.* 41 (1993), 141–156.

[2] Paulo Brandimarte, Routing and scheduling in a flexible job shop by tabu search. *Ann. Oper. Res.* 41 (1993), 157–183.

[3] Andrew Clark, Ed Dawson, and Helen Bergen, Combinatorial optimisation and the knapsack cipher. *Cryptologia* 20 (1996), 85–93.

[4] Richard L. Daniels and Joseph B. Mazzola, A tabu-search heuristic for the flexible-resource flow shop scheduling problem. *Ann. Oper. Res.* 41 (1993), 207–230.

[5] K.A. DeJong, *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems.* (Doctoral Dissertation, University of Michigan Press, Ann Arbor, Michigan, 1975)

[6] W. S. Forsyth and R. Safavi-Naini, Automated cryptanalysis of substitution ciphers. *Cryptologia* 17(4) (1993), 407–418.

[7] Fred Glover, Tabu search: A tutorial. *Interfaces* 20 (1990), 74–94.

[8] Fred Glover, Eric Taillard, and Dominique de Werra, A user's guide to tabu search. *Ann. Oper. Res.* 41 (1993), 3–28.

[9] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* (Addison Wesley, Reading, Mass., 1989).

[10] Pierre Hansen, Eugenio de Luna Pedrosa Filho, and Celso Carneiro Ribeiro, Location and sizing of offshore platforms for oil exploration. *Europ. J. Oper. Res.* 58 (1992), 202–214.

[11] J. Holland, *Adaptation in Natural and Artificial Systems.* (University of Michigan Press, Ann Arbor, Michigan, 1975.)

[12] Thomas Jakobsen, A fast method for cryptanalysis of substitution ciphers. *Cryptologia* 19 (1995), 265–274.

[13] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, Optimization by simulated annealing. *Science* 220(4598) (1983), 671–680.

[14] Robert A. J. Matthews, The use of genetic algorithms in cryptanalysis. *Cryptologia* 17 (1993), 187–201.

[15] N. Metropolis, A. W. Rosenblunth, M. N. Rosenblunth, A.H. Teller, and E. Teller, Equations of state calculations by fast computing machines. *J. Chem. Phys.* 21 (1953), 1087–1092.

[16] Abraham P. Punnen and Y. P. Aneja, Categorized assignment scheduling: A tabu search approach. *J. Oper. Res. Soc.* 44 (1993), 673–679.

[17] Colin R. Reeves, Improving the efficiency of tabu search for machine sequencing problems. *J. Oper. Res. Soc.* 44 (1993), 375–382.

[18] Frederic Semet and Eric Taillard, Solving real-life vehicle routing problems efficiently using tabu search. *Ann. Oper. Res.* 41 (1993). 469–488.

[19] Richard Spillman, Cryptanalysis of knapsack ciphers using genetic algorithms. *Cryptologia* 17 (1993), 367–377.

[20] Richard Spillman, Mark Janssen, Bob Nelson, and Martin Kepner, Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers. *Cryptologia* 17 (1993), 31–44.

[21] U.S. Department of Commerce/National Bureau of Standards, *Data Encryption Standard*, January 1988. (Federal Information Processing Standards Publication 46-1.)