Automated Cryptanalysis of Transposition Ciphers

J. P. GIDDY AND R. SAFAVI-NAINI

Department of Computer Science, University of Wallongong, Northfields Ave., Wollongong 2500, Australia

In this paper we use simulated annealing for automatic cryptanalysis of transposition ciphers. Transposition ciphers are a class of ciphers that in conjunction with substitution ciphers form the basis of all modern symmetric algorithms. In transposition ciphers, a plaintext block is encrypted into a ciphertext block using a fixed permutation. We formulate cryptanalysis of the transposition cipher as a combinatorial optimization problem, and use simulated annealing to find the global minimum of a cost function which is a distance measure between a possible decipherment of the given ciphertext and a sample of plaintext language. The success of the algorithm depends on the ratio of the length of ciphertext to the size of the block. For lower ratios there are cases that the plaintext cannot be correctly found. This is the expected behavior of all cryptanalysis methods. However, in this case, examining the output of the algorithm provides valuable 'clues' for guiding the cryptanalysis. In summary, simulated annealing greatly facilitates cryptanalysis of transposition ciphers and provides a potentially powerful method for analyzing more sophisticated ciphers.

Received April, 1994

1. INTRODUCTION

In a *secrecy system* (Shannon, 1949), a transmitter wants to send a message to a receiver over a public channel. An *enemy* has access to the communication in the channel. To hide the content of the message from the enemy, the transmitter applies an *encryption* transformation to the *plaintext* and obtains a *ciphertext* (or *cryptogram*). An *encryption algorithm* is a family of transformations indexed by a piece of information called a *key*. A basic assumption in cryptography is that the encryption algorithm is publicly known, and the key is the only piece of information that is kept secret and allows the communicants to encrypt/decrypt securely. In a *ciphertext-only* attack, the enemy has access to a cryptogram and tries to find the corresponding plaintext (or equivalently the key in the case of unique decipherment).

Two major types of cryptographic algorithms are *substitution* and *transposition*. In the former one, every plaintext character is substituted by a ciphertext character, using a *substitution alphabet* (which is the key information in this case), and in the latter one, plaintext characters are permuted using a predetermined permutation (key information). The importance of these two types of ciphers stems from Shannon's observation that alternating a number of rounds of substitution and transposition implements the basic *confusion* and *diffusion* required in a secure cipher and creates a very strong cryptographic algorithm. This is in fact the approach taken in all modern block cipher algorithms such as Data Encryption Standard (DES).

A possible algorithm for the enemy, to find the plaintext without knowing the key (also called *breaking the cipher*), is to systematically generate all the keys and use them to decrypt the intercepted cryptogram and accept a decryption that is meaningful. (We note that the chance of having two plaintexts mapping into the same cryptogram tends to zero as the length of the cryptogram increases.) Ths is called *brute force attack*. A brute force attack is highly resource expensive because it requires execution of the decryption algorithm for every possible key. Moreover, the process of accepting/rejecting a decryption requires human intervention and/or implementation of sophisticated pattern matching algorithms, dictionary searches or other methods of text recognition. Automating this process has been a challenging area of research in recent years. This research falls into two main categories. In the first category, the emphasis is on automating the text recognition part (Peleg, 1979; Carroll and Robbins, 1987, 1989) of the cryptanalysis, while in the second category the search for the key in the key space is driven by a cost function (or in some cases a *fitness criterion*) (Forsyth and Safavi-Naini, 1993; Matthews, 1993; Spillman *et al.*, 1993).

In Forsyth and Safavi-Naini (1993), a successful application of simulated annealing to the cryptanalysis of substitution ciphers is reported, and in Spillman *et al.* (1993) and Matthews (1993) genetic algorithms are used for the analysis of substitution ciphers and a subclass of transposition ciphers, known as *columnar transposition* ciphers.

In this paper we formulate cryptanalysis of the general periodic transposition cipher as a combinatorial optimization problem and use the method of simulated annealing to find the global minimum. We prove the convergence of the algorithm for a specific neighborhood and cost function, describe a finite time approximation of the algorithm, and give experimental results which demonstrate the success of this approach. The method, as presently implemented, completely eliminates human intervention and provides an elegant and efficient method for cryptanalysis of transposition ciphers. If the length of the cipher is 'short', the algorithm might fail to find the plaintext (this is the typical behavior of all cryptanalytic attacks); however, it can still provide valuable 'clues' for conducting further analysis. This is mainly done through recognizing 'probable words' in the output of the algorithm due to its partial success.

In Section 2 we review the preliminary background and in Section 3 we describe a simulated annealing algorithm for cryptanalysis of transposition ciphers. In Section 4 finite time implementation of the algorithm is discussed and performance results of the algorithm are reported. We discuss our results and directions for future research.

2. PRELIMINARIES

2.1. Simulated annealing

Simulated annealing is a probabilistic algorithm used for combinatorial optimization problems. Such problems can be described by a pair (S, C), where S is a finite set of possible *configurations* and C is a *cost function* (also called *score function*) that attaches a positive real number (*cost*) to every state in S. The aim of the algorithm is to find the state with a minimal value of C, i.e. the *global minimum*. The success of the algorithm in finding this global minimum, as opposed to the iterative search methods that can be easily trapped in a local minimum, is in the inclusion of a randomization element that allows probabilistic hill climbing in the search process. Simulated annealing is successfully used in a number of applications (van Laarhoven and Aarts, 1987). The algorithm works by defining a neighborhood relation N on the set of states in S, $N \subset S \times S$. If $s_j \in S$ is a neighbor of $s_i \in S$ then s_j is called *adjacent* to s_i and is said to be *reachable* from it in one *move*. The state space must be connected and there exists an integer $d(S, N)$, which specifies the maximum number of moves required to reach a state s_j starting from a state s_i, for all $s_i, s_j \in S$. At each step of the algorithm, when the algorithm is at state s_i, with the value of the cost function being $C(s_i)$, a generation mechanism is used to probabilistically generate a new state s_j. The probability of generating state s_j from state s_i at temperature T is denoted by $G_{ij}(T)$. T is a control parameter corresponding to the temperature of annealing in metals. The value of the cost function $C(s_j)$ at this new state is calculated and is compared with $C(s_i)$. A *transition* (move) to the new state will be made with probability $A_{ij}(T)$ where,

$$A_{ij}(T) = P(s_j \text{ acceptable from state } s_i)$$

$$= \begin{cases} 1, & C(s_j) \leqslant C(s_i), \\ \exp\left(\dfrac{C(s_i) - C(s_j)}{T}\right), & C(s_j) > C(s_i). \end{cases}$$

The basic step is known as a *Metropolis step*. T is a control parameter corresponding to the temperature in annealing of metals. The probability of transition from state i to state j is denoted by $P_{ij}(T)$, and is given by,

$$P_{ij}(T) = \begin{cases} G_{ij}(T)A_{ij}(T), & \forall\, i \neq j, \\ 1 - \sum\limits_{l=1, l \neq i}^{|S|} G_{il}(T)A_{il}(T), & i = j, \end{cases}$$

where $G_{ij}(T)$ and $A_{ij}(T)$ are generation and acceptance probabilities of state s_j from state s_i at temperature T.

At each temperature, the step is repeated for many times and then the temperature is decreased. Mathematically, simulated annealing can be modeled using homogeneous or inhomogeneous Markov chains (van Laarhoven and Aarts, 1987). In the homogeneous case, which we have considered for this implementation, the algorithm can be described as a sequence of Markov chains. The value of the control parameter is kept fixed over the whole chain and is decreased between two consecutive chains. Convergence of the algorithm to the global minimum is guaranteed if the matrix $A_{ij}(T)$ and the generation mechanism satisfy the conditions of van Laarhoven and Aarts (1987, p. 22, Theorem 2).

3. CRYPTANALYSIS OF TRANSPOSITION CIPHERS

A *plaintext* is a sequence of characters over an alphabet $\mathscr{A} = \{a_1, \cdots a_p\}$. A *transposition cipher* takes a block $(m_1, \ldots, m_n)$ of n consecutive characters of plaintext and permutes it according to an n-permutation P, to produce the ciphertext block $(c_1, \ldots, c_n)$. In cryptanalyzing the cipher we assume that the enemy has access to the ciphertext and attempts to find the corresponding plaintext without the knowledge of permutation P. It is assumed that the size of the permutation, n, is known. In the following we describe a simulated annealing algorithm for the cryptanalysis of transposition ciphers.

Let $(i_1, i_2, \ldots, i_n)$, $1 \leqslant i_j \leqslant n$, $1 \leqslant j \leqslant n$, denote the n-permutation that moves one to the i_1th place, two to i_2th place, etc.

The *configuration space*, S, consists of the set of all possible permutations of n objects, hence, $|S| = n!$. The *neighborhood* of a state s_i is the collection of all states s_j that are accessible from s_i in one *move*. A *move* from a state s_i reaches a state s_j and consists of an application of a *generalized transposition* (*G-transposition*), to s_i. A *G-transposition* $[\alpha, l, k]$, is an n-permutation that swaps two adjacent blocks in the sequence $(1, 2, \ldots, n)$. The first block starts at α and has length l and the second one starts at $\alpha + l$ and has length k. Hence the n-permutation representing $[\alpha, l, k]$ is

$$(1, 2, \cdots \alpha, \alpha+l+1, \ldots, \alpha+l+k, \alpha+1, \ldots, \alpha+l \ldots, \cdots n)$$

A move from state s_i using the G-transposition $[\alpha, l, k]$ ends in state s_j where $s_j = [\alpha, l, k] \circ s_i$ and $\circ$ denotes composition of n-permutations.

We note that for any G-transposition $[\alpha, l, k]$, there exists an inverse G-transposition $[\alpha, l, k]^{-1}$, that will *undo* the action of $[\alpha, l, k]$. In fact $[\alpha, l, k]^{-1} = [\alpha, k, l]$.

3.1. Cost function

The cost function attaches a real number $C(s)$ to a state $s \in S$. Let $p_{\alpha\beta}$ denote the probability of the bigram $\alpha\beta$ in the plaintext. To compute $C(s)$, the permutation defining state s is used to decrypt the ciphertext and the relative frequency of the bigram $\alpha\beta$ in the resulting output, denoted by $c_{\alpha\beta}$, is calculated. We define $C(s)$ as,

$$C(s) = N \sum_{\alpha=a}^{z} \sum_{\beta=a}^{z} \frac{|p_{\alpha\beta} - c_{\alpha\beta}|}{p_{\alpha\beta}}$$

where N is the length of the ciphertext. We note that the cost function is expected to have its minimum if s corresponds to the permutation used for encryption. States that are in the neighborhood of s will have at most six bigram frequencies (determined by the applied G-transposition).

A cost function based on bigram frequencies has the advantages of capturing a reasonable amount of plaintext structure and being relatively simple to compute.

3.2. Properties of state space and neighborhood function

The neighborhood relation determines the states that are immediately accessible from the current state. The choice of the neighborhood relation is probably the most important decision to be made for the success of the algorithm. In the following we review important properties of the neighborhood relation, as defined above, which have contributed to the efficiency of the algorithm and proving its convergence.

3.2.1. *The space is connected*

Proposition 1. $d(S, N) = n - 1$.

Proof. Consider two states $s_i = (p_1, \ldots, p_n)$ and $s_j = (q_1, \ldots, q_n)$ belonging to the configuration space S. We show that there exists a sequence of at most $n-1$ G-transpositions $\Pi_1, \Pi_2, \ldots, \Pi_k$ such that

$$(\Pi_k \circ \Pi_{k-1} \circ \cdots \circ \Pi_1) s_i = s_j$$

where $(\Pi_k \circ \Pi_{k-1} \circ \cdots \circ \Pi_1) s_i = (\Pi_k \circ \Pi_{k-1} \circ \cdots)(\Pi_1 s_i)$. We will construct Π_1 and use a recursive argument to construct the rest. Define Π_1 to be the G-transposition that generates state s_{i+1} from state s_i,

$$\Pi_1 \circ s_i = (q_1, p_{i_1}, \ldots, p_{i_{n-1}}) = s_{i+1}$$

Π_1 can be obtained by locating q_1 in s_i, say $p_l = q_1$, and swapping the two blocks $(p_1, \ldots, p_{l-1})$ and $(p_l, \ldots, p_n)$, i.e. $\Pi_1 = [0, l, n-l]$. We note that s_{i+1} has its first element equal to q_1 and hence at most $n-2$ G-transpositions are required to take s_{i+1} to s_j. This is true because the last element must be in place when the second last one is placed.

3.2.2. *The neighborhood relation is reflexive and symmetric*

G-transpositions $[0, 0, 0]$, $[0, 0, n]$, $[0, n, 0]$ and $[n, 0, 0]$ fix elements of the state space and hence the relation is reflexive.

If the neighborhood of a state $s_i = (p_1, \ldots, p_n)$ contains a state $s_j = (q_1, \ldots, q_n)$, there exists a G-transposition Π such that $\Pi \circ s_i = s_j$. Using Π^{-1} we have $\Pi^{-1} \circ s_j = s_i$ and hence the relation is symmetric.

3.2.3. *The landscape of state space is smooth*

A desirable property of the state space is that the terrain of the energy/cost landscape be as smooth as possible. 'Deep valleys should also be big valleys, while small valleys should be shallow' (Otten and van Ginneken, 1989). Since the cost function is defined in terms of bigram frequencies, moving to a neighboring state should have the least effect on these frequencies. The suggested neighborhood function causes at most six bigram frequencies per block to be changed in each move. This is a small change compared to the case that the two blocks are disjoint.

3.3. Probabilistic generation of neighboring states

To generate an element of the neighborhood probabilistically we need to have a probabilistic method of generating G-transpositions. As noted earlier a G-transposition is of the form $[\alpha, l, k]$ where $\alpha, l, k \in \{0, 1, \ldots n\}$. To generate a G-transposition we use a pseudorandom number generator (PN generator) that generates three pseudorandom numbers in the interval $[0, n]$. The numbers are sorted in increasing order and used to define a G-transposition. When the three outputs α, β and γ (in sorted form) of the PN generator are distinct, a unique G-transposition $[\alpha, \beta-\alpha, \gamma-\beta]$ can be defined. In the cases that at least two outputs are the same, the lowest value from the triple of outputs is set to zero and the highest value is set to n. Note that, if the middle value is 0 or n, the values will still not be distinct.

In Table 1, the first column lists the possible triple output patterns of the PN generator. The second column contains the probability of a pattern for specific values of a, b and c. The third column indicates the number of choices that exist for the values a, b and c. The fourth column shows the G-transposition produced by the three outputs of the PN generator. The fifth column shows the type of operation performed by the G-transposition. A *rotation* is a G-transposition in which the whole block of length n is broken into two subblocks and the two subblocks are exchanged. This type of G-transposition occurs more frequently because of the mapping of the outputs of the PN generator to a

TABLE 1. Probability of state transitions

PNG output	Output probability	Combinations	G-transposition	Action	Individual probability
$\{0, 0, 0\}$	P	1		no change	$(6n+2)P$
$\{0, 0, n\}$	$3P$	1	$[0, 0, n]$		
$\{0, 0, a\}$	$3P$	$n-1$			
$\{n, n, n\}$	P	1			
$\{0, n, n\}$	$3P$	1	$[0, n, 0]$		
$\{a, n, n\}$	$3P$	$n-1$			
$\{0, b, n\}$	$6P$	$n-1$		rotate around b	$(3n+6)P$
$\{0, b, b\}$	$3P$	$n-1$			
$\{a, b, b\}$	$3P$	$\binom{n-1}{2}$	$[0, b, n-b]$		
$\{b, b, c\}$	$3P$	$\binom{n-1}{2}$			
$\{b, b, n\}$	$3P$	$n-1$			
$\{b, b, b\}$	P	$n-1$			
$\{0, a, b\}$	$6P$	$\binom{n-1}{2}$	$[0, a, b-a]$	exchange	$6P$
$\{a, b, n\}$	$6P$	$\binom{n-1}{2}$	$[a, b-a, n-b]$		
$\{a, b, c\}$	$6P$	$\binom{n-1}{3}$	$[a, b-a, c-b]$		

$a, b, c \in \{1, 2, \ldots, n\} \quad a < b < c$

G-transposition described above. If after the mapping the values are not distinct no change of state is made. The sixth column is the probability of the G-transposition occurring for particular values of a, b and c. We have $P = 1/(n+1)^3$.

THEOREM 1. *The simulated annealing algorithm for transposition ciphers with neighborhood and cost function given above is convergent.*

Proof. We show that the following conditions, given in Theorem 2, p. 22 of van Laarhoven and Aarts (1987), are satisfied and hence the algorithm is convergent.

$$s_i, s_j \in S : G_{ij} = G_{ji} \tag{1}$$

$$s_i, s_j, s_k \in S : C(s_i) \leqslant C(s_j) \leqslant C(s_k),\ A_{ik}(T) = A_{ij}(T)A_{jk}(T) \tag{2}$$

$$s_i, s_j \in S : C(s_i) \geqslant C(s_j),\ A_{ij}(T) = 1 \tag{3}$$

$$s_i, s_j \in S,\ T > 0 : C(s_i) < C(s_j),\ 0 < A_{ij}(T) < 1 \tag{4}$$

As noted in Section 3 the generation mechanism is symmetric and reflexive so condition 1 is satisfied. Moreover, we have

$$\begin{aligned} A_{ij}(T)A_{jk}(T) &= \exp\left(-\frac{(C(s_j) - C(s_i))}{T}\right) \\ &\quad \times \exp\left(-\frac{(C(s_k) - C(s_j))}{T}\right) \\ &= \exp\left(-\frac{(C(s_k) - C(s_i))}{T}\right) = A_{ik}(T) \end{aligned}$$

which proves condition 2. Condition 3 is true because a cost decreasing transition is always allowed and finally condition 4 is true because

$$1 > A_{ij}(T) = \exp\left(-\frac{(C(s_j) - C(s_i))}{T}\right) > 0,$$

for $C(s_j) > C(s_i)$.

4. IMPLEMENTATION

4.1. Cost function

The cost function is a measure of *the number of characters in correct sequence.* If all characters are in the correct order, the text will have statistical characteristics similar to other plaintexts. The frequency with which bigrams occur in a plaintext is one such statistic. Forsyth (1992) uses a cost function where the cost of any state is the sum of differences between bigram frequencies in the decipherment of the ciphertext obtained by applying the permutation identifying the state and those in a sample plaintext.

We have used a weighted version of this cost function. In a typical 1000 character English message, we might expect 27 occurrences of the bigram 'TH' and zero occurrences of the bigram 'BZ'. Because 'BZ' is *so* unlikely to occur in English, it is reasonable to expect that a message containing 3 'BZ's is less likely to be a valid message than one containing 24 'TH's. The cost function from Forsyth (1992) gives equal weighting to these two differences. A simple weighting method is to divide the absolute difference by the expected value, giving the relative error. The use of relative errors was found to provide much better recognition of a valid message.

A table of expected frequencies of bigrams is generated from the sample text. The values in this table are scaled to contain the same number of bigrams as the ciphertext. For a bigram $\alpha\beta$, denote the *expected* count by $p_{\alpha\beta}$.

To calculate the cost of a state, a bigram table is created that contains the *actual* bigram counts, $c_{\alpha\beta}$, for the decipherment represented by the state. The partial cost of each bigram $\alpha\beta$ is $|p_{\alpha\beta} - c_{\alpha\beta}|/(p_{\alpha\beta} + \varepsilon)$. The parameter ε is a small value used to prevent division by zero.

The total cost of a state is the sum of the partial costs for all bigrams in the alphabet Λ^2. For a state s, the cost function is:

$$C(s) = \sum_{\alpha \in \Lambda} \sum_{\beta \in \Lambda} \frac{|p_{\alpha\beta} - c_{\alpha\beta}|}{p_{\alpha\beta} + \varepsilon} \qquad (5)$$

In this cost function, the least frequent components have a greater effect than the more frequent components on the final cost function. Because of this, the value of ε has a significant effect on the ability of the cost function to select the correct decipherment. $\varepsilon = 0.001$ places too much significance on the least frequent patterns. Good results can be obtained using $\varepsilon = 0.1$. However, $\varepsilon = 1$ works better for some ciphertexts, notably those containing dummy characters.

4.2. Cooling schedule

To achieve a finite-time implementation of the simulated annealing algorithm, we must specify a set of parameters that determine the convergence of the algorithm. These parameters are combined to form a cooling schedule.

A cooling schedule usually consists of four parameters

- start value of the cooling control parameter,
- number of moves generated at each value of the control parameter (length of the Markov chain),
- decrement of the control parameter,
- stop criterion.

Theoretical models are based on the infinite-time system. We try to model the behavior of an infinite-time system as closely as possible. We attempt to choose values for the parameters of the cooling schedule according to well-known heuristics, in order to obtain convergence in polynomial time.

4.2.1. Start temperature

Start temperature is determined by the method in Aarts and van Laarhoven (1985).

This involves starting at a random point in the state space. A possible move is generated and the cost difference is calculated. A value for the initial temperature is generated by the equation,

$$T_0 = \overline{\Delta C}^{(-)} \left(\ln \frac{m_2}{m_2\chi - (1-\chi)m_1} \right)^{-1}$$

where $\overline{\Delta C}^{(-)}$ is the average of all positive changes in cost, χ is the acceptance ratio, m_1 is the number of negative transitions and m_2 is the number of positive transitions.

The Metropolis acceptance criterion is then applied and if accepted the move is taken. These trials are repeated for a large number of moves (equal to the chain length).

4.2.2. Length of the Markov chain

The length of each chain must be long enough to allow quasi-equilibrium to be attained at each value of the control parameter. We use a chain length equal to the size of the neighborhood, as suggested in Aarts and van Laarhoven (1985, equation 64),

$$\begin{aligned} L &= |N| \\ &= 1 + \binom{n+1}{3} \end{aligned}$$

For a period of 10, the chain length is 121; for a period of 25, the chain length is 2601.

4.2.3. Temperature decrement

Between chains, the control parameter is decremented by multiplying it by a value between 0 and 1. A higher value causes the annealing to proceed at a slower rate. Typical values for the temperature decrement ratio are 0.8 and 0.9.

4.2.4. Stop criterion

A simple stop criterion is used. If the final states of four consecutive chains are the same, the algorithm is terminated. A more complex stop criterion may allow processing to stop earlier, but this method is efficient to calculate and is reasonably reliable.

4.3. Evaluation

The cryptanalysis program operates under the assumption that the global minimum of the cost function corresponds to the correct decryption. This is often true, as the cost function is a measure of a decipherment's conformity to the expected frequencies of bigrams in the plaintext language. However, there is no guarantee that the frequencies of the correct decipherment match the expected values perfectly. Meaningful messages may vary greatly in their statistical profiles, while it is possible to create garbage with excellent statistics.

The redundancy available because of the repetitive nature of a periodic cipher strongly inhibits the chances of a meaningless message being accepted as the decipherment. The more blocks of text that are available, the better chance the program has of picking the correct text. This is evident for smaller ciphertexts where the number of complete blocks is smaller, resulting in less correct decipherments.

The ability of the cost function to recognize a correct decipherment is essential to the success of the program.

There are several possible improvements to the cost function that build on the bigram table approach, while adding better plaintext recognition. One is to use a dictionary search to locate words in a decipherment. In keeping with standard cryptanalysis methods, a small dictionary containing probable words could be created by the cryptanalyst. The appearance of these words in a decipherment could have a weighted effect on the cost function. Another method is to include a trigram frequency comparison similar to the current bigram frequency comparison in the current algorithm.

Calculation of the cost function is fairly expensive since the move to a new state must actually be performed to allow the change in cost of moving to the new state to be calculated. In Forsyth (1992), the change in cost is calculated independently of performing a move, allowing faster generation of moves. The ability to do this is dependent on the problem.

The cost is calculated incrementally. Only the bigrams that are split or joined by the move are used in updating the bigram table. The most expensive part of the code is updating the bigram frequency table. The usual, and largest, number of updates to the bigram table for one move is $6c/n$, where c is the number of characters in the ciphertext and n is the period of the cipher.

The *time complexity* of the annealing algorithm with the current neighborhood function can be determined from Aarts and van Laarhoven (1985, equation 67). Where n is the period of the cipher, the execution time of the statistical cooling algorithm is proportional to

$$\begin{aligned}|N|\ln|S| &= \binom{n+1}{3}\ln n! \\ &< Cn^3 \ln n^n \\ &= Cn^4 \ln n\end{aligned}$$

Thus the theoretical time complexity is $O(n^4 \ln n)$. This does not take into account calculation of the cost function. The size of the calculation required for the cost function is indicated above.

Implementation is in C++. The current implementation is written for flexibility rather than optimal code generation. The use of classes simplifies the representation of individual states and allows efficient permutation of the ciphertext, while maintaining the appearance of dynamic text.

The ciphertext is stored in an allocated doubly-indexed array. The first index accesses a vector of pointers, each one pointing to a block of the ciphertext. The second index accesses an element of the block. To allow for fast permutation of the elements of all blocks, the second index is usually an element from an integer vector, representing the current permutation. This allows moves to be made by changing the integer vector instead of moving columns in the array. This structure is hidden inside a class, and is accessed through an iterator class that allows movement through the text in read order, and also down a column (i.e. the same position in each block).

5. RESULTS

Table 2 shows typical running times on a Sun Sparc station SLC for various cipher lengths and block sizes. These figures indicate a time complexity of $O(n^3 c)$, which is significantly better than expected.

The results in Table 3 are derived from a series of tests in which style, cipher length and block size were varied. The block sizes were 10, 15, 20, 25 and 30. The cipher lengths were 2000, 1000, 750, 500, 250, 200, 150 and 100. Cipher lengths slightly larger than these values indicate dummy characters added. The other parameters were at their default values. That is, the acceptance ratio was 0.95 and the temperature decrement ratio was 0.85. The value of ε used in the cost function (see Section 4.1) was 0.1. All ciphertexts consisted of alphabetic letters only. The sample text was a 8171 character UNIX manual entry.

The program was run on ciphertexts representing two different styles: prose and UNIX manuals. Three *different* ciphers of each combination of length, style and period were deciphered using the program.

Success of the decryption was measured as the percentage of bigrams in the decipherment that were in the correct order: 100% indicates that the cipher was completely solved. The *median* of the success rates for the three decipherments is shown.

We note that decipherments less than 100% are likely to provide valuable clues to the cryptanalyst. For example, for a period of 10, a success rate of 80% means every 10 characters contains two subsequences in correct order. This is a good result since it means that in every 10 characters there is one correct subsequence of at least five characters and only one other subsequence. It is easy for the cryptanalyst to see the correct decryption from this point. This also holds for larger periods. For example, the following text remains only 80% solved after an attempt to decrypt a cipher of period 20.

```
andsawkawkusercomman
gncannimeawkpatterns
galanguandprocessing
maprogresynopsisawkf
...
```

Although the left side is unsolved, the rightmost 13 columns appear to be in correct order. The third line can be used to solve the rest as the word ‘language’ stands out as a probable word. In general, a success rate of 80% can be considered to be decrypted satisfactorily. At lower values, messages are usually unreadable, although partial words may be detected, which may help in further analysis.

In general, a success rate of at least 80% can be

TABLE 2. Annealing times for decryption on a Sun Sparc station SLC

Period (n)	Cipher length (c)	CPU time (min:s)
10	500	0:29
10	1000	0:49
15	1000	2:31
20	1000	4:47
25	1000	8:14
30	1000	13:49

TABLE 3. Results for differing language types where the sampled plaintext consists of 8171 letters from Unix OS manuals

Cryptanalysis of different language styles

Cipher style	Period	Cipher length	Bigrams/ column	Successfully decrypted (%)
Prose	10	250	25	100
Unix manual	10	250	25	100
Prose	10	200	20	100
Unix manual	10	200	20	100
Prose	10	150	15	100
Unix manual	10	150	15	100
Prose	10	100	10	70
Unix manual	10	100	10	50
Prose	15	510	34	100
Unix manual	15	510	34	100
Prose	15	255	17	100
Unix manual	15	255	17	80
Prose	15	210	14	73
Unix manual	15	210	14	53
Prose	15	150	10	73
Unix manual	15	150	10	46
Prose	20	1000	50	100
Unix manual	20	1000	50	100
Prose	20	760	38	70
Unix manual	20	760	38	80
Prose	20	500	25	100
Unix manual	20	500	25	85
Prose	20	260	13	65
Unix manual	20	260	13	60
Prose	25	1000	40	100
Unix manual	25	1000	40	100
Prose	25	750	30	80
Unix manual	25	750	30	100
Prose	25	500	20	88
Unix manual	25	500	20	80
Prose	25	250	10	48
Unix manual	25	250	10	52

obtained if the ratio of the cipher length to the cipher period (c/n) is at least 20.

Most failures to decrypt correctly were the result of the global minimum not being the correct decipherment. This is especially true for low c/n ratios. In such cases, the simulated annealing algorithm finds the global minimum, but this minimum does not necessarily represent the correct decipherment. This is in contrast to many uses of simulated annealing, such as circuit wiring placement and the Traveling Salesman Problem, where the cost function directly measures a value to be minimized (distance) (Kirkpatrick *et al.*, 1983).

Ciphertexts containing dummy characters were also deciphered poorly. Increasing the ε parameter may improve the results for these ciphers.

The experiments described were repeated using a sample text of 26 000 characters of prose. The results for ciphers containing prose improved. Conversely, the ciphers containing manual pages were solved less successfully. This emphasizes the necessity of selecting a sample text of the same style as expected in the decipherment.

6. CONCLUSION

It has been shown that simulated annealing can be successfully applied to the problem of cryptanalyzing periodic transposition ciphers. Convergence of the algorithm with the specified neighborhood function has been proved.

The method has completely solved some periodic transposition ciphers with periods of 25 and cipher lengths of 500 characters. Where a complete solution is not found, the results can be examined for partially correct solutions. In most cases, solutions that are 80% correct can be found for ciphers whose length is at least 20 times the cipher period.

The ability of the cost function to differentiate valid text from a series of characters with 'good statistics' is essential to the program's ability to decrypt correctly. The cost function presented in this paper provides a significant improvement over the one used by Forsyth and Safavi-Naini (1993). The current system may be improved by use of a more complex cost function.

Further work may concentrate on the ability to recognize a valid message with a small sample, determining the blocksize of the cipher automatically, and using additional knowledge of the cipher (e.g. that it is a columnar transposition) to make the cryptanalysis more efficient.

ACKNOWLEDGMENTS

We wish to thank Bill Forsyth for providing the source code from his work. Much inspiration was derived from his work and programs. Also, several procedures (the annealing driver and the cooling schedule code) in the transposition cipher cryptanalysis program have directly evolved from Bill's code. Support for this work was provided in part by Australian Research Council grant A49030136.

REFERENCES

Aarts, E. H. L. (1992) Simulated annealing. *UMAP J.*, **13**, 79–89.

Aarts, E. H. L. and van Laarhoven, P. J. M. (1985) Statistical cooling: a general approach to combinatorial optimization problems. *Phillips J. Res.*, **40**, 193–226.

Carroll, C. and Robbins, L. (1987) The automated cryptanalysis of polyalphabetic ciphers. *Cryptologia*, **XI**, 193–205.

Carroll, C. and Robbins, L. (1989) Cryptanalysis of product ciphers. *Cryptologia*, **XIII**, 303–326.

Forsyth, W. (1992) *Solving Substitution Ciphers using the Method of Simulated Annealing*. Honors Thesis, Department of Mathematics, Statistics and Computing Science, The University of New England, March 1992.

Forsyth, W. and Safavi-Naini, R. (1993) Automated cryptanalysis of substitution ciphers. *Cryptologia*, **XVII**, 407–420.

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

van Laarhoven, P. J. M. and Aarts, E. H. L. (1987) *Simulated Annealing: Theory and Application*. Reidel, Dordrecht.

Matthews, R. A. J. (1993). The use of genetic algorithms in cryptanalysis. *Cryptologia*, **XVII**, 187–201.

Otten, R. H. J. M. and van Ginneken, L. P. P. P. (1989). *The Annealing Algorithm*. Kluwer, Dordrecht.

Peleg, S. and Rosenfeld, A. (1979) Breaking substitution ciphers using a relaxation algorithm. *Commun. ACM*, **22**, 598–605.

Shannon, C. E. (1949) Communication theory of secrecy systems. *Bell System Tech. J.*, **28**, 656–715.

Spillman, R., Janssen, M., Nelson, B. and Kepner M. (1993) Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers. *Cryptologia*, **XVII**, 31–44.