Making the Most of Two Heuristics: Breaking Transposition Ciphers with Ants

Matthew D. Russell
Department of Computer Science
The University of York
York, YO10 5DD
matthew@cs.york.ac.uk

John A. Clark
Department of Computer Science
The University of York
York, YO10 5DD
jac@cs.york.ac.uk

Susan Stepney
Department of Computer Science
The University of York
York, YO10 5DD
susan@cs.york.ac.uk

Abstract- Multiple anagramming is a general method for the cryptanalysis of transposition ciphers, and has a graph theoretic representation. Inspired by a partially mechanised approach used in World War II, we consider the possibility of a fully automated attack. Two heuristics based on measures of natural language are used — one to recognise plaintext, and another to guide construction of the secret key. This is shown to be unworkable for cryptograms of a certain difficulty due to random variation in the constructive heuristic. A solver based on an ant colony optimisation (ACO) algorithm is then introduced, increasing the range of cryptograms that can be treated; the pheromone feedback provides a mechanism for the recognition heuristic to correct the noisy constructive heuristic.

1 Introduction

Cryptography has had a long and colourful history. The earliest schemes, now termed the classical ciphers, were designed to be carried out with pen and paper rather than by electronics. Many were transpositions: algorithms which rearrange the order of letters in a message. Classical cryptography became obsolete after the advent of computers; more complex ciphers could be used, and older ciphers broken with greater ease. Nonetheless, modern analogues of classical schemes can still be found as components of larger ciphers. In particular, some iterated block ciphers, such as the Data Encryption Standard [Nat99], incorporate transpositions to provide diffusion.

The cryptanalyst's tactic when presented with a transposition was to exploit particular statistical features of the ciphertext, as well as to rely upon intuition, luck and trial-and-error, to find the correct decryption. As this was sometimes too slow a process, mechanised aids were used as early as World War II [Bau97] by which frequencies of letter pairs (known as *bigrams*) were automatically examined in order to narrow down the space of possible keys. The remaining few keys could then be checked exhaustively by hand to recover the plaintext.

We consider the possibility of fully automating this procedure. A straightforward implementation turns out to be incapable of decrypting harder cryptograms due to random variation in the bigram heuristic. We quantify which cryptograms are hard for this algorithm. It will be shown that the pheromone feedback mechanism of an Ant Colony System is capable of overcoming some random variation and decrypting a wider variety of messages.

A preliminary version of this result was summarised in [RCS03].

The rest of the paper is organised as follows: first, we review transposition ciphers, Section 2, and standard attacks on transpositions, Section 3. In Section 4, the fully automated attack is presented. A dictionary-based score is used to recognise plaintext. However, difficulties with the bigram heuristic limit its application. The remedial Ant Colony System is then given in Section 5, with experimental results in Section 6. Previous approaches are contrasted in Section 7, and finally conclusions are given in Section 8.

2 Transposition Ciphers and Bigrams

In this section, two forms of the transposition cipher (see [Gai39]) are introduced and their cryptanalysis shown to be equivalent. The first is known as *columnar transposition*. In this cipher, the plaintext is written into a grid of fixed width, padded with dummy characters if necessary. The columns of the grid are numbered according to a permutation — the permutation forms the secret key — and read off column-wise in that sequence. As an example, consider the plaintext "MYANTALGORITHMSARERIDDLEDWITHBUGS", encrypted using the key (31524):

3	1	5	2	4
M	Y	A	N	T
A	L	G	O	R
I	T	H	M	S
A	R	E	R	I
D	D	L	E	D
W	I	T	H	B
U	G	S	X	X

Ciphertext: YLTRDIG NOMREHX MAIADWU TRSIDBX AGHELTS

Decryption is simply a matter of writing the ciphertext back into the grid using the same ordering of the columns.

The second form is termed *complete-unit transposition*. The plaintext is divided into a series of blocks (units) of a fixed length w, again padding if necessary. A permutation of size w is applied to each block in turn, rearranging the letters. The sequence of permuted blocks is then used as the ciphertext. Here is an example with the same key and plaintext as before:

```
      31524 31524 31524 31524 31524...
      MYANT ALGOR ITHMS ARERI DDLED...
  ⇒   YNMTA LOARG TMISH RRAIE DEDDL...
```

In a sense this latter form of the transposition is also the most general, as any transposition can be recast as a complete-unit transposition with key size set to the length of the plaintext.

Both of these forms of the transposition cipher are susceptible to an attack known as *multiple anagramming*. The key size w is assumed known (there are statistical tests for this purpose), and the ciphertext is written into a grid with w columns. For columnar transpositions, the ciphertext must be written into the grid column by column from left to right, and dually for complete-unit transpositions the ciphertext is entered row by row from top to bottom. The columns then have to be rearranged to form readable plaintext in every row. For example,

Y	N	M	T	A
L	O	A	R	G
T	M	I	S	H
R	R	A	I	E
D	E	D	D	L
I	H	W	B	T
G	X	U	X	S

$\xrightarrow{\text{rearrange}}$

M	Y	A	N	T
A	L	G	O	R
I	T	H	M	S
A	R	E	R	I
D	D	L	E	D
W	I	T	H	B
U	G	S	X	X

Finding the correct rearrangement is clearly equivalent to finding the key. Certain patterns inherent in natural language can be exploited in order to do this efficiently.

3 Historical Cryptanalysis of Transposition Ciphers

One property of written natural language is that the distribution of pairs of letters, known as *bigrams*, is not uniform. In English, for example, '`TH`' is common and '`QZ`' is rare. Using some large sample of text, stripped of numbers, punctuation, whitespace and other non-letters, a standard probability for each bigram can be obtained[1]. For other texts, the observed frequencies will tend to be close to these probabilities. Two columns placed next to each other form several bigrams, one for each row. The *bigram adjacency score*, $Adj_{(I,J)}$, is defined as the average probability of the bigrams created by juxtaposing columns I and J, i.e.,

$$Adj_{(I,J)} \stackrel{\text{def}}{=} \frac{1}{h} \sum_{r=1}^{h} P_{\text{std}}(I_r J_r), \tag{1}$$

where I_r and J_r denote the rth letter in the column I or J respectively, $P_{\text{std}}(\texttt{xy})$ is the standard probability of the bigram '`xy`', and h is the number of rows in a column. The score will be higher for two correctly aligned columns, because the bigrams will be from the plaintext. If they are incorrectly aligned, the pairs will be much more random, and likely to score lower. This distinction is examined more rigorously in Section 4.

From the bigram adjacency score, a pen-and-paper cryptanalyst infers the top candidates for each column's neighbour. Together with other statistical clues, it is usually straightforward to reassemble the columns correctly.

[1]For the experiments described in this paper, these probabilities were taken to be the same as the relative frequencies of bigrams observed in a 140 000 character novel [Wel98].

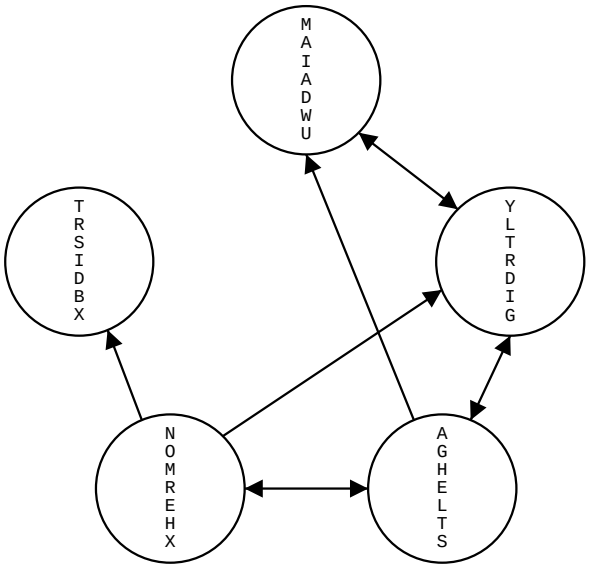

Figure 1: The anagramming graph produced by the cryptogram used as the running example. Arcs with bigram adjacency score below $\alpha = 0.0035$ have been pruned.

A more general method is also known [Bau97] that is less reliant on *ad hoc* exploitation of particular features of the cryptogram. This method has been partially automated, as mentioned in the introduction, and will now be considered in more detail.

We will first need some preliminaries: by $I \parallel J$, we mean that when columns I and J are adjacent in that order they form bigrams from the plaintext. If this is not the case, then we write $I \nparallel J$. The multiple anagramming problem can be represented as a graph, as has been done in Figure 1. We will call such a graph the *anagramming graph* of the problem. Each node denotes a column. A directed arc from column I to column J indicates that $I \parallel J$ has not been ruled out.

Since the transposition key is a permutation, a candidate key can be represented as some path through the column nodes which does not pass through the same column twice. Normally, this would specify $w!$ possible keys, where w is the width of the grid; even for small w this precludes an exhaustive search. In the historical attack, arcs on the anagramming graph are pruned to restrict the number of paths. The number of keys is hopefully reduced to a point where each can be checked by hand to see which produces a comprehensible plaintext. To prune the arcs, a cutoff value α is chosen; an arc is included from node I to J if and only if $Adj_{(I,J)} > \alpha$.

In the following section we attempt to convert this into a fully automated attack.

4 A Fully Automated Attack

Human interaction is still required for the recognition of plaintext. This can be automated by a heuristic score based on the number of dictionary words in a candidate decryption. Longer words are given more weight than shorter

words — they are less likely to appear by chance and so give stronger evidence of a correct decryption. Let N_d be the number of d-letter substrings from the text which are present in a standard dictionary. The dictionary heuristic used in this paper is defined as:

$$Dict(M) \stackrel{\text{def}}{=} \frac{1}{L} \sum_{d=3}^{10} d^2 N_d \tag{2}$$

where L is the length of the text. A list of 40 000 words, derived from [Atk03], is used as the standard dictionary.

In practice, we find that $Dict(M)$ is maximum whenever M is the correct plaintext, considered over all the possible decryptions M of a cryptogram. Hence, one way to recover the plaintext is to consider the maximisation problem with $Dict$ as the objective function. Specifically, all the paths through the pruned anagramming graph are evaluated using $Dict$, and the maximum is identified as the plaintext.

It remains to select the cutoff value α. The choice for α is critical both to the time complexity of the algorithm and the possibility of finding the correct decryption at all. If α is chosen too high, then it is likely that the arc linking two columns I and J, where $I \parallel J$, will be pruned. This would be disastrous, as the correct key is removed from the search space. Conversely, if α is too low, though the confidence in retaining the right solution increases, the time taken to find it rises too because there are more paths through the graph. In this algorithm, there is a direct tradeoff between computational efficiency and probability of success.

We will attempt to derive some guidance on the choice of α. When juxtaposing two columns I and J, consider for the moment just the bigram produced in the top row, I_1J_1. We can define a random variable on I_1J_1,

$$S(I_1J_1) \stackrel{\text{def}}{=} P_{\text{std}}(I_1J_1) \tag{3}$$

where $P_{\text{std}}(I_1J_1)$ is the standard probability of the bigram I_1J_1 appearing in natural language text. These probabilities are assumed to be known, determined using a standard corpus, and are used below to calculate expected values.

If $I \parallel J$, the bigram will occur with probability

$$Prob(I_1J_1) = P_{\text{std}}(I_1J_1) \tag{4}$$

The expected value of $S(I_1J_1)$ is therefore

$$E[S(I_1J_1)] = \sum_{I_1J_1} P_{\text{std}}(I_1J_1)^2 \tag{5}$$

$$\approx 0.008 \tag{6}$$

However, if $I \nparallel J$, we can use the standard frequencies of single letters to estimate the probability of the bigram.

$$Prob(I_1J_1) = P_{\text{std}}(I_1)P_{\text{std}}(J_1) \tag{7}$$

where $P_{\text{std}}(X_1)$ is the probability of the single letter X_1 appearing at some position. The expected value of $S(I_1J_1)$ then becomes

$$E[S(I_1J_1)] = \sum_{I_1J_1} P_{\text{std}}(I_1)P_{\text{std}}(J_1)P_{\text{std}}(I_1J_1) \tag{8}$$

$$\approx 0.004 \tag{9}$$

h	5	10	15	20
α	0.0014	0.0034	0.0042	0.0047

Table 1: α cutoff values which give a 95% probability of the correct solution being in an anagramming graph for a cryptogram of fixed width $w = 15$ and varying height h.

The variance of $S(I_1J_1)$ can also be calculated in a similar fashion.

Two juxtaposed columns form several bigrams, one for each row. $Adj_{(I,J)}$ is the mean value of S over all the rows. Assuming that the bigram on each row is independent, $Adj_{(I,J)}$ is a sample mean with a sample size of h, the height of the column ($h = L/w$). By the Central Limit Theorem, this can be approximated by a normal distribution, obtaining:

$$Adj_{(I,J)} \sim N\left(0.008, \sqrt{\frac{3 \times 10^{-5}}{h}}\right) \text{ if } I \parallel J. \tag{10}$$

$$Adj_{(I,J)} \sim N\left(0.004, \sqrt{\frac{6 \times 10^{-5}}{h}}\right) \text{ if } I \nparallel J. \tag{11}$$

For any cutoff value α, and some pair of columns I and J where $I \parallel J$, the probability p_c that the arc between those columns will be in the anagramming graph can be calculated using (10). $w - 1$ correct arcs are needed in total, and, again assuming independence, the probability that the anagramming graph contains the correct solution is given by p_c^{w-1}. The reasoning can also be performed backwards; for a fixed probability, e.g. 95%, an α value can be calculated, dependent on h and w. This has been done in Table 1 for $w = 15$.

Unfortunately, (11) suggests that using a low value for α will result in few arcs being pruned, increasing the computational cost. It is also a time-consuming task just to identify all the legal paths in the graph even before they can be checked. If the automated cryptanalysis is to be tractable, a more subtle method is needed.

A cursory investigation using local search techniques indicated that maximising $Dict$ is made difficult by the existence of many local optima. It might be helpful to make use of the information provided by $Adj_{(I,J)}$, despite some noise, to give guidance towards the promising regions of the search space. For this task, we examine an Ant Colony Optimisation approach.

5 Ants for Cryptanalysis

Ant Colony Optimisation (ACO) [DS03] is a search metaheuristic inspired by the behaviour of real ant colonies. Ants lay trails of a chemical known as *pheromone* which can be used for communicating information about, for example, path finding. By analogy, artificial ants use artificial pheromone to represent knowledge gained about a search problem based on prior experience. Pheromone modifies ants' subsequent choices when constructing solutions.

	ATSP	MA
Constructs a...	cycle	simple path
Next node heuristic...	distance	$Adj_{(I,J)}$
Solution evaluated by...	total distance	$Dict(M)$

Table 2: Differences between using ACO for the Asymmetric Travelling Salesman Problem (ATSP) and Multiple Anagramming (MA)

ACO was originally applied to the $\mathcal{NP}$-hard Travelling Salesman Problem (TSP). Finding solutions in the anagramming graph bears many similarities to an Asymmetric TSP (ATSP). A specific ant algorithm with known success on the TSP is Ant Colony System (ACS). The reader is referred to [DG97] for a full description; here we will only highlight differences and specifics in the adaptation for this problem.

Ants construct paths on the anagramming graph of a cryptogram; each complete path is a permutation of the nodes corresponding to a key. It is not necessary to prune arcs, as ACS makes use of a heuristic η_{ij}, which represents some *a priori* knowledge of the desirability of choosing node j while at node i. For this, $Adj_{(I,J)}$ is used. In each iteration, after the ants have constructed their paths, pheromone is deposited on arcs in the graph corresponding to the best path found since the start of the algorithm. A completed path, which is a decryption key, is evaluated using the $Dict$ heuristic on the plaintext produced.

The essential difference between the multiple anagramming problem and the ATSP is the use of one heuristic for constructing a solution, $Adj_{(I,J)}$, and another for evaluating it, $Dict$; see Table 2. The two heuristics are not disparate — both are statistics of an underlying cryptogram — but $Adj_{(I,J)}$ varies in how useful its guidance to the $Dict$ maximum is.

6 Computational Results

Figure 2 shows an overview result of cryptanalysis trials for a variety of cryptograms of differing heights (h) and widths (w). A sample plaintext of the implied length ($L = wh$) is obtained as a random subsection of a long text [Car65], and encrypted using a fixed key. The ACS solver returns a key candidate within 5000 $Dict$ evaluations. We recorded only whether the key was exactly equal to the correct decryption key.

The results show that the ACS algorithm is most likely to succeed on cryptograms with a small width and large height. This is consistent with what we know: a large height provides many bigram samples for any column pair, meaning $Adj_{(I,J)}$ is a more reliable statistic. A small width gives a small key-space over which the algorithm must search.

Figure 2 also indicates that cryptograms can sometimes be solved even when the key size is as large as 40. In this case the total number of possible keys is $40! \approx 10^{48}$, but the ACS solver examines (using $Dict$) only 5000 of them in order to locate the correct key.

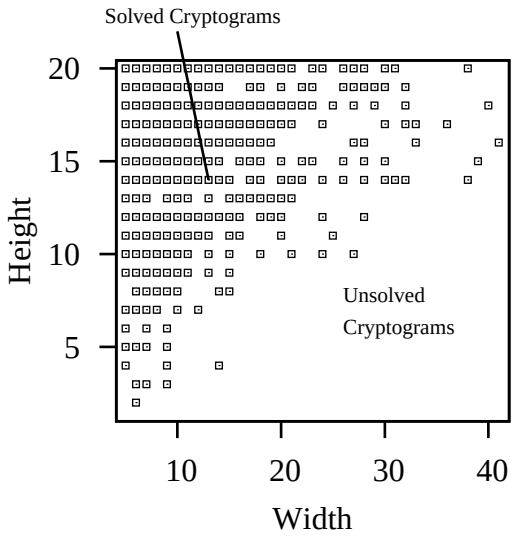

Figure 2: Solvability: a point is plotted at a given width, $5 \leq w \leq 42$, and height, $1 \leq h \leq 20$, if and only if the ACS algorithm solved a sample cryptogram of length $L = wh$ within 5000 $Dict$ evaluations. Only one trial at each point was attempted.

Sometimes, ACS returns solutions that are close to the key. For example, two elements in the permutation might be in the incorrect order, or the permutation may start on the wrong column. Despite this, the plaintext remains mostly readable. To investigate this property, we use a success metric which counts the number of columns which are placed before their correct successor. Figure 3 shows how, for a fixed width, the accuracy of the solution gradually improves with increasing height.

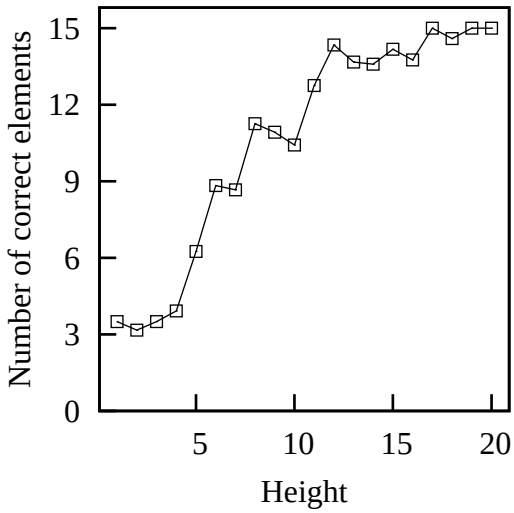

Figure 3: Partial solvability: Number of correct key elements found by ACS (5000 $Dict$ evaluations) for cryptograms with a fixed width $w = 15$ and varying height. An average over 12 trials is given for each height.

Method	Width (w)	Length (L)
GA [Mat93]	7	184
	9	184
SA [GSN94]	10	150
	20	1000
	25	1000
GA [Cla98]	18	1000
SA [Cla98]	15	600
	21	1000
Tabu Search [Cla98]	18	1000
ACS [this paper]	15	300
	20	400
	25	625

Table 3: An overview of some typical results in automated cryptanalysis of transposition ciphers. GA = Genetic Algorithm; SA = Simulated Annealing

7 Previous Work on Automated Cryptanalysis

In this section, we examine previous work on automated cryptanalysis using metaheuristics, and provide a limited comparison.

An early paper considering transposition ciphers is [Mat93], which applies a genetic algorithm to the problem. The transpositions are of a somewhat harder type, permitting uneven column lengths. In a subsequent paper [GSN94], a different metaheuristic is employed — simulated annealing. In his PhD thesis [Cla98], Andrew Clark provides a comparative study of simulated annealing and genetic algorithms, together with a third metaheuristic: tabu search. For transpositions, simulated annealing solves the most difficult cryptograms, but is also the slowest algorithm tested.

Analogous results for a different type of classical encryption, substitution ciphers, can be found in [FSN93, SJNK93, CD98].

Table 3 lists typical results from previous work on transpositions, together with some from this paper. The various approaches involve differing heuristics, processing time and success criteria. Despite this, a comparison does give a rough indication of the regions of difficulty in the problem space. It also shows that the ACS algorithm presented in this paper can decrypt cryptograms which are significantly shorter (up to a factor of about a half) than those tackled by previous metaheuristic methods.

The ability of ACS to successfully decrypt cryptograms of shorter length can be attributed to the use of a dictionary heuristic in addition to bigrams. The earlier approaches used only bigrams to evaluate the fitness of candidate solutions. This is susceptible to noise in shorter messages for similar reasons to the problems demonstrated in the bigram adjacency score. In a local search method, this leads to a situation where locating the global optimum does not guarantee the correct decryption of the cryptogram; this was noted in [GSN94].

Local search algorithms could potentially be modified to use the dictionary score. One pitfall is that the dictionary heuristic does not define a very smooth search space; e.g. an interchange of just two columns would in all likelihood destroy a long and high scoring word. It would be desireable to combine it with the bigram heuristic in some way; however, it is not immediately obvious as to how this could be done robustly. A constructive algorithm, like ACS, allows both heuristics to be integrated in a natural way.

A recent result indicates that metaheuristic techniques may be applicable to modern cipher cryptanalysis: in [HSIR02], statistical flaws in a reduced-round version of the block cipher TEA are found using a genetic algorithm.

8 Conclusions and Further Work

It has been shown how the cryptanalysis of transposition ciphers can be completely automated in a ciphertext-only attack.

Two heuristics are used: one for recognising plaintext using a dictionary, $Dict$, and another for indicating adjacent columns using bigrams, $Adj_{(I,J)}$. Taken by themselves, each is insufficient to easily decrypt a difficult cryptogram. Constructing the key solely using the bigram scores becomes infeasible when random variation causes the heuristic to be deceptive. Equally, $Dict$ defines a search space that is hard to maximise. However, in ACS both heuristics can be used in a complementary fashion to rapidly find the solution. The bigram scores provide hints to the right region of the $Dict$ search space. The pheromone feedback, calculated using $Dict$, compensates for any noise in the $Adj_{(I,J)}$ scores.

The capability of maximising $Dict$ allows much shorter cryptograms to be automatically decrypted compared to previous techniques. When 5000 $Dict$ evaluations are used, the length can be reduced by a factor of around two.

We believe this is the first application of an Ant Colony Optimisation approach to cryptanalysis. More difficult transposition schemes, such as irregular and double columnar transpositions, might also be vulnerable to this approach. However, it is unlikely that these methods will pose a threat to modern ciphers in direct key-recovery attacks such as those just presented — the key spaces are designed to be as discontinuous as possible. Nonetheless, much of modern cryptanalysis is concerned with extracting information reliably from noisy statistics; perhaps ants could improve existing techniques.

Bibliography

[Atk03] Kevin Atkinson. The spell checking oriented word lists (SCOWL). http://wordlist.sourceforge.net/, 2003.

[Bau97] F. L. Bauer. *Decrypted Secrets*. Springer, second edition, 1997.

[Car65] Lewis Carroll. *Alice in Wonderland*, volume 11 of *Project Gutenberg*. Project Gutenberg, 1865.

[CD98] Andrew Clark and Ed Dawson. Optimisation heuristics for the automated cryptanalysis of clas-

sical ciphers. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 28:63–86, 1998.

[Cla98] Andrew Clark. *Optimisation Heuristics for Cryptology*. PhD thesis, Queensland University of Technology, 1998.

[DG97] Marco Dorigo and Luca Maria Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, April 1997.

[DS03] M. Dorigo and T. Stützle. The ant colony optimization metaheuristic: Algorithms, applications and advances. In F. Glover and G. Kochenberger., editors, *Handbook of Metaheuristics*, pages 251–285. Kluwer Academic Publishers,, 2003.

[FSN93] W. S. Forsyth and R. Safavi-Naini. Automated cryptanalysis of substitution ciphers. *Cryptologia*, 17(4):407–418, October 1993.

[Gai39] Helen Fouché Gaines. *Elementary cryptanalysis: a study of ciphers and their solution*. Dover, New York, USA, 1939.

[GSN94] J. P. Giddy and R. Safavi-Naini. Automated cryptanalysis of transposition ciphers. *The Computer Journal*, 37(5):429–436, 1994.

[HSIR02] J. C. Hernández, J. M. Sierra, P. Isasi, and A. Ribagorda. Genetic cryptoanalysis of two rounds TEA. *Lecture Notes in Computer Science*, 2331:1024–1031, 2002.

[Mat93] Robert A. J. Matthews. The use of genetic algorithms in cryptanalysis. *Cryptologia*, 17(2):187–201, April 1993.

[Nat99] National Institute of Standards and Technology (NIST). Data Encryption Standard (DES). Federal Information Processing Standards Publication (FIPS PUB) 46-3, October 1999.

[RCS03] Matthew Russell, John A. Clark, and Susan Stepney. Using ants to attack a classical cipher. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, volume 2723 of *Lecture Notes in Computer Science*, pages 146–147, 2003. Poster paper.

[SJNK93] Richard Spillman, Mark Janssen, Bob Nelson, and Martin Kepner. Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers. *Cryptologia*, 17(1):31–44, January 1993.

[Wel98] Herbert George Wells. *The Time Machine*, volume 35 of *Project Gutenberg*. Project Gutenberg, 1898.