Decrypting Classical Cipher Text Using Markov Chain Monte Carlo

Jian Chen and Jeffrey S. Rosenthal*
Department of Statistics, University of Toronto

(May 2010; revised November 2010.)

We investigate the use of Markov Chain Monte Carlo (MCMC) methods to attack classical ciphers. MCMC has previously been used to break simple substitution ciphers. Here, we extend this approach to transposition ciphers and to substitution-plus-transposition ciphers. Our algorithms run quickly and perform fairly well even for key lengths as high as 40.

1 Introduction

Cryptography (e.g. [16]) is the study of algorithms to encrypt and decrypt messages between senders and receivers. And, Markov chain Monte Carlo (MCMC) algorithms (e.g. [19, 6, 13]) are popular methods of approximately sampling from complicated probability distributions. Traditionally these two subjects have been quite distinct.

However, recently MCMC algorithms have been used to iteratively converge to solutions which allow us to break simple substitution codes. This approach was first introduced by Marc Coram and Phil Beineke in the Stanford statistical consulting service (see Diaconis [3]), and later studied more systematically by Connor [1]. Table 1 shows output from a typical run of this algorithm (in this case, decrypting the first line of the Project Gutenberg [12] copy of *Oliver Twist*).

Iteration #	First Line of Decrypted Text
0	LIW PSKMWCL YNLWRDWSY WDKKJ KH KGUXWS LAUEL DQ CIVSGWE FUCJWRE
200	RAS KINJSBR MDRSEHSIM SHNNV NW NGUZSI RPUOR HY BATIGSO LUBVSEO
400	ARS HUNJSPA GDASEBSUG SBNNV NW NMIKSU AFIOA BY PRTUMSO LIPVSEO
600	ARE HLNJEPA KOAESBELK EBNNW NG NMIVEL AFIDA BY PRULMED TIPWESD
800	IME KNSJEPI HUIETBENH EBSSG SW SLOVEN ICODI BY PMANLED ROPGETD
1000	IME GNOJEPI HUIETBENH EBOOK OF OLSVEN ICSDI BY PMANLED RSPKETD
1200	SME GNOJECS HUSETBENH EBOOK OF OLIVEN SPIDS BY CMANLED RICKETD
1400	SME PNOJECS HUSETBENH EBOOK OF OLIVEN SWIDS BY CMANLED RICKETD
1600	SHE MROJECS GUSETBERG EBOOK OF OLIVER SWIDS BY CHARLED NICKETD
1800	SHE PROJECS GUSETBERG EBOOK OF OLIVER SWINS BY CHARLEN DICKETN
2000	SHE PROJECS GUSELBERG EBOOK OF ONIVER SWITS BY CHARNET DICKELT
2200	THE PROJECT GUTENBERG EBOOK OF OLIVER TWIST BY CHARLES DICKENS

Table 1: A sample run of a simple MCMC decryption algorithm.

*Supported in part by NSERC of Canada.

We see that the algorithm begins with encrypted text that looks like gibberish, and then gradually (in this case, after 2200 iterations) breaks the code and recovers the correct original text.

In this paper, we significantly extend the use of MCMC in decryption, from simple substitution ciphers to transposition ciphers and to substitution-plus-transposition product ciphers. While these new cases still correspond to "classical" ciphers, they are generally regarded as being significantly more complicated than simple substitution ciphers (see Section 1.1). To successfully attack these ciphers, we develop various innovations including combining multiple independent runs, cycling between different cipher attacks, and (for substitution-plus-transposition ciphers) using a uni-gram attack as an initialization point for a sequence of bi-gram attacks.

Extensive computer simulations indicate that our algorithms run quickly, and work quite well even for key lengths as large as 40. These results appear to improve upon existing decryption methods [4], and suggest that MCMC algorithms can be of genuine use for decrypting encoded text.

This paper is organized as follows. We present background on cryptography and on MCMC below. The use of MCMC for decryption is outlined in Section 2. We then present detailed algorithms and simulation results for attacking substitution ciphers (Section 3), transposition ciphers (Section 4), and substitution-plus-transposition ciphers (Section 5). All of the software used for our simulations is freely available at: `probability.ca/decipher`

1.1 Background on Cryptography

In cryptography, the original text is called the plain text, and the encrypted text is called the cipher text. The algorithms to perform encryption and decryption are referred to as ciphers. Usually a cipher contains one or two keys. In a symmetric key algorithm (e.g. DES), the decryption key is the same as the encryption key (or just the inverse function of it). In an asymmetric key algorithm (e.g. RSA), two different keys are used. The public key is used for encryption and a private key is for the decryption.

Ciphers can also be categorized in a different way, as classical ciphers and modern ciphers. Classical ciphers, such as the substitution and transposition ciphers considered herein, perform encryption and decryption text manipulations at the byte level. Modern ciphers, such as DES (symmetric key) and RSA (asymmetric key), perform encryption and decryption at the bit level, and are correspondingly more complicated and secure than the classical ciphers, and we do not consider them here. (Note, however, that a simplified version of DES, called SDES [11, 5], can be regarded as a special case of substitution cipher and is thus included in our results below.)

A *simple substitution cipher* works by replacing each letter with another one. In this paper, we only substitute alphabetic letters; spaces are left untouched and all other non-alphabetic characters are removed. So, the number of the possible keys is equal to $26! \doteq 4 \times 10^{26}$. Table 2 illustrates an encryption and decryption example of a simple substitution cipher. For the encryption, all 'A's in the plain text are replaced by letter 'X', 'B's replaced by 'E', etc. For the decryption, all 'A's in the cipher text are replaced by 'I', 'B's replaced by 'C' etc. Note that the encryption key is the inverse function of the decryption key.

plain text	`THE PROJECT GUTENBERG EBOOK OF OLIVER TWIST`
encryption key	XEBPROHYAUFTIDSJLKZMWVNGQC
cipher text	`MYR JKSURBM HWMRDERKH RESSF SO STAVRK MNAZM`
decryption key	ICZNBKXGMPRQTWFDYEOLJVUAHS
decrypted text	`THE PROJECT GUTENBERG EBOOK OF OLIVER TWIST`

Table 2: A simple example of a substitution cipher encryption and decryption.

Another classical cipher is the *transposition cipher* (also called the *permutation cipher*). The letters in the plain text stay the same but their positions are rearranged in a different order. A simple transposition cipher works by splitting the plain text into fixed sized blocks. The length of the key (also called the period) is the same as the size of the block. Letters in each block are permuted according to a same pattern(the key). Table 3 illustrates an example of encryption and decryption by a transposition cipher with key length 10. Note the encryption key is the inverse function of the decryption key.

plain text	T	H	E		P	R	O	J	E	C
encryption key	1	9	3	7	0	4	5	8	6	2
cipher text	H	C		J	T	P	R	E	O	E
decryption key	4	0	9	2	5	6	8	3	7	1
decrypted text	T	H	E		P	R	O	J	E	C

Table 3: A simple example of a transposition cipher encryption and decryption.

Transposition ciphers are generally regarded as much more difficult to decrypt than simple substitution ciphers. For example, Matthews [8] (p. 190) notes that:

> the automatic breaking of [transposition] ciphers is notoriously difficult. In contrast to substitution ciphers, for which a number of statistical tools aiding automated breaking have been developed ... cryptanalysis of transpositions is usually highly interventionist and demands some knowledge of the likely contents of the ciphertext to give an insight into the order of rearrangement used.

A *product* cipher combines a sequence of simple transformations such as substitution, transposition and other arithmetic. SP-network (Substitution-permutation network) is an example of a product cipher, involving repeated applications of substitutions (S-box) and permutations (P-box), which is very common in the design of modern ciphers such as DES. Herein, we consider the special case of a simple substitution-plus-transposition (or, substitution-transposition) cipher, in which the plain text is encrypted by a substitution cipher followed by a transposition cipher, causing additional challenges.

Such product ciphers have long been commonly used to make decryption more difficult. Shannon [17] (p. 671) noted:

> Product encipherment is often used; for example, one follows a substitution by a transposition.

Stinson [18] (Section 2.7) agreed, writing:

> [C]ombining cryptosystems by forming their "product" ... has been of fundamental importance in the design of present-day cryptosystems ... Taking the product of substitution-type ciphers with permutation-type ciphers is a commonly used technique.

Menezes et al. [9] further emphasise how the combination of substitution and transposition ciphers can lead to "strong ciphers", writing (p. 20):

> Simple substitution and transposition ciphers individually do not provide a very high level of security. However, by combining these transformations it is possible to obtain strong ciphers. As will be seen in Chapter 7 some of the most practical and effective symmetric-key systems are product ciphers. One example of a product cipher is a composition of $t \geq 2$ transformations $E_{k_1}, E_{k_2}, \ldots, E_{k_t}$, where each E_{k_i}, $1 < i < t$, is either a substitution or a transposition cipher.

All of these quotations indicate that transposition ciphers, and especially product ciphers, are generally regarded as being significantly more difficult to decrypt than simple substitution ciphers.

Frequency analysis (e.g. [17]) is the study of the frequencies of letters or combination of letters in the cipher text. In a particular language (e.g. English) certain letters and there combinations occurs more frequently than others. The frequencies of letters are also called n-gram, i.e. uni-gram stand for single letter frequencies, bigram for combination of 2 letters, trigram for 3 letters, etc. For example, in English 'E' is the most used single letter while 'Z' is the least used single letter. 'TH' and 'ER' are pairs which arise frequently. Simple classical ciphers like substitution ciphers are often broken by comparing the letter frequencies of the cipher text to a reference text (usually a large text such as *War and Peace*).

When faced with a simple substitution cipher, the simplest form of frequency analysis is a uni-gram attack, which involves simply replacing the most frequent letter in the cipher text by the most frequent occurred letter in the reference text, and the second-most-frequent letter in the cipher text by the second-most-frequent letter in the reference text, and the third-most by the third-most, and so on. As a first experiment, we tried this simple uni-gram attack on the novel *Oliver Twist* after a random simple substitution cipher was applied to it. This attack does not succeed very well, revealing just 16 out of 26 letters in the cipher text (Table 4). This is because some letters (e.g. 'R' and 'S', or 'C' and 'W') have similar frequencies and are thus likely to be interchanged in such an attack.

plain text	`THE PROJECT GUTENBERG EBOOK OF OLIVER TWIST`
decrypted text	`THE PSOJEWT FUTEIBESF EBOOK OG OLNVES TCNRT`

Table 4: Attempted decryption of *Oliver Twist* with a uni-gram attack.

Because of results like this, more complicated attacks involving pair frequencies have to be employed – even for simple substitution ciphers, but especially for more complicated ciphers such as transposition and product ciphers. Since pairs cannot be simply "substituted in" as is done with uni-gram attacks, this leads to more complicated algorithms, as we discuss herein.

1.2 Background on MCMC

MCMC algorithms have long been used by physicists and statisticians to sample from complicated high-dimensional probability distributions. Let $\pi(\cdot)$ be an important possibly-unnormalised density (for example, the posterior distribution from a Bayesian inference problem) on a state space $\mathcal{X}$. (In statistical inference problems, usually $\mathcal{X}$ is an open subset of $\mathbf{R}^d$, but in this paper $\mathcal{X}$ will be a finite set.) MCMC proceeds by defining an iterative sequence $X_0, X_1, X_2, \ldots$ of $\mathcal{X}$-valued random

variables which converge in distribution to $\pi(\cdot)$. For example, the following result is well-known (see e.g. [15], Theorem 8.3.10; or [19], Theorem 1).

Proposition 1. *If a Markov chain $\{X_n\}$ on a finite or countable state space $\mathcal{X}$ is irreducible and aperiodic, with stationary distribution $\pi(\cdot)$, then for every subset $A \subseteq \mathcal{X}$,*

$$\lim_{n\to\infty} \mathbf{P}(X_n \in A) \;=\; \int_A \pi(x)\, dx\,. \tag{1}$$

It follows from this proposition that for large n, the value X_n is approximately a "sample" from $\pi(\cdot)$. Repeating or continuing this process leads to multiple samples, which can then be used to estimate probabilities and expected values with respect to $\pi(\cdot)$.

The simplest version of MCMC is the full-dimensional Metropolis algorithm [10], which proceeds as follows:

- Choose an initial state $X_0 \in \mathcal{X}$.
- For $n = 1, 2, 3, \ldots,$
  - Propose a new state $Y_n \in \mathcal{X}$ from some symmetric proposal density $q(X_{n-1}, \ldots)$.
  - Let $U_n \sim \text{Uniform}[0, 1]$, independently of $X_0, \ldots, X_{n-1}, Y_n$.
  - If $U_n < (\pi(Y_n)/\pi(X_{n-1}))$, then "accept" the proposal by setting $X_n = Y_n$, otherwise "reject" the proposal by setting $X_n = X_{n-1}$.

Thus, the acceptance probability of each proposal is equal to $\min(1, \pi(Y_n)/\pi(X_{n-1})$. This probability is chosen precisely so that the resulting Markov chain $X_0, X_1, X_2, \ldots$ will be *reversible* with respect to $\pi(\cdot)$, so that $\pi(\cdot)$ is a stationary distribution, and under the mild assumptions of irreducibility and aperiodicity, the probabilities will converge to those of $\pi(\cdot)$ as in (1). (It is not essential the the proposal density $q(x, \cdot)$ be symmetric, but if it is not then the acceptance probability must be appropriately modified, so for simplicity we do not consider that case here.)

It is also possible to replace $\pi(x)$ by a power, $(\pi(x))^p$, so that the acceptance condition above is replaced by $U_n < ((\pi(Y_n)/\pi(X_{n-1}))^p$. This is a *tempering* modification in which p plays the role of *inverse temperature*, and we shall refer to p as a *scaling parameter*. Such a modification changes and flattens (for $0 < p < 1$) the density $\pi(\cdot)$, potentially changing the corresponding probabilities and expected values, but leaving the *mode* (argmax) of $\pi(\cdot)$ unchanged. Thus, such tempering modifications can help the chain escape from local modes, while preserving the same mode; we shall make use of them herein.

2 Using MCMC to Break Classical Ciphers

We now discuss the use of MCMC for breaking classical ciphers.

For this application, the relevant Markov chain has state space $\mathcal{X}$ consisting of all possible decryption keys (a large but finite state space). That is, each possible decryption key is a possible state of the Markov Chain. Following [3, 1], we make use of a long reference text such as *War and Peace*. For each pair of characters β_1 and β_2 (e.g. β_1 =T and β_2 =H), we let $r(\beta_1, \beta_2)$ record

the number of times that specific pair (e.g. "TH") appears consecutively in the reference text. Similarly, for a putative decryption key $x \in \mathcal{X}$, we let $f_x(\beta_1, \beta_2)$ record the number of times that pair appears when the cipher text is decrypted using the decryption key x. To avoid problems from zeroes, we also add one to each of $r(\beta_1, \beta_2)$ and $f_x(\beta_1, \beta_2)$.

For a particular decryption key x, we then define its *score function* as follows:

$$\pi(x) \;=\; \prod_{\beta_1,\beta_2} r(\beta_1, \beta_2)^{f_x(\beta_1,\beta_2)}\,. \tag{2}$$

This function can be thought of as multiplying, for each consecutive pair of letters in the decrypted text, the number of times that pair occurred in the reference text. Intuitively, the score function is higher when the pair frequencies in the decrypted text most closely match those of the reference text, and the decryption key is thus most likely to be correct. (In our computer programs, we compute (2) on a log scale for easy calculation and to avoid numerical errors.)

In terms of this score function, we use the following general MCMC algorithm to break the classical ciphers:

- Choose an initial state (initial decryption key), and a fixed scaling parameter $p > 0$.
- Repeat the following steps for many iterations (e.g. 10,000 iterations).
  - Given the current state x, propose a new state y from some symmetric density $q(x, y)$.
  - Sample $u \sim \text{Uniform}[0, 1]$ independently of all other variables.
  - If $u < (\frac{\pi(y)}{\pi(x)})^p$ then accept the proposal y by replacing x with y, otherwise reject y by leaving x unchanged.

By the usual Markov chain convergence theorem, this Markov chain will converge in probability to its stationary distribution, which in this case means it will converge to the distribution with density proportional to $(\pi(x))^p$ with $\pi(\cdot)$ as in (2). So, intuitively, after many iterations, the algorithm is likely to be at a decryption key which gives decryption text pair frequencies close to those of the reference text, and is thus more likely to be correct.

2.1 Previous related work

The use of MCMC algorithms to break simple substitution codes was introduced by Marc Coram and Phil Beineke in unpublished work (summarised in the Introduction to [3]) that they undertook for the Stanford Statistics Department's drop-in statistical consulting service. A psychologist from the California state prison system had presented them with a collection of coded messages. They correctly guessed that the messages were encrypted using a simple substitution cipher. They then ran an algorithm very similar to that introduced above. It quickly decoded the messages and discovered that they were written mostly in English, but with some Spanish words and other "jargon" also included. This early success indicated the possibility of using MCMC algorithms for decryption purposes.

These algorithms were then studied more systematically by Connor [1]. He provided precise definitions and framework and background for studying substitution ciphers using MCMC, putting the proposal choices in the more general context of random walks on the symmetric group. He then ran careful simulations to do the decoding, while varying a number of parameters such as the text being decoded (a variety of English-language novels), the length of text used for the decryption, the number of MCMC iterations used, and the starting state used by the algorithm.

Connor [1] generally achieved quite high decryption success rates, sometimes as high as 99%, especially when using an "intelligent" starting state (defined as the one which matches up the character frequencies in the cipher and reference texts). He also tried decoding more unusual texts such as Welsh writing and a Biochemistry textbook, though with less success (which was not surprising since the character pair frequencies of *War and Peace* are less relevant for these texts). In any case, his results were all restricted to simple substitution ciphers, not to the more challenging transposition and product ciphers considered herein.

In a different direction, Connor [1] also considered the issue of the running time (i.e., rate of convergence) for the MCMC algorithms he was using. However, as that was a very difficult problem, he instead considered the algorithm consisting of just the *proposed* moves, ignoring the accept/reject step, thus corresponding to a pure random walk for the uniform distribution on the symmetric group without regard to the actual text data being decrypted. For this simplified algorithm, he applied stopping time and group representation arguments of Diaconis [2] to provide concrete quantitative bounds on distance to stationarity after k iterations. These results are quite interesting mathematically. However, they are not of direct relevance for the original decryption algorithms which we study herein.

Another approach was taken by Matthews [8]. He designed an algorithm GENALYST for decrypting transposition ciphers using genetic algorithms which postulate multiple possible solutions and then deletes and permutes them using a "survival of the fittest" procedure. He achieved some promising results. However, his success was limited by a number of factors. For example, he measured the "fitness" success of his solutions by computing the resulting frequencies of just 10 fixed letter patterns (TH, HE, IN, ER, AN, ED, THE, ING, AND, EEE; see Table 1 on his p. 192), rendering the assessment of his results unclear and incomplete. Also, his algorithm does not in general produce a final solution but merely narrows down the plausible solutions to a smaller set (of size 24, in the case of key length 11; see the top of his p. 200), and still requires "the final breaking of the cipher into English being achieved by ... the human brain", i.e. it requires additional human intervention to completely decrypt the text. By contrast, the algorithms considered herein are designed to produce a single final answer with no further human intervention, and are assessed in terms of actual comparison of the decrypted text with the original text.

2.2 Testing methodology

To test our algorithms, we shall primarily use the four texts listed in Table 5. During the programming and initial testing we used *War and Peace* as the reference text and *Oliver Twist* as the plain text. All four texts were then used to test our final attack algorithms. (A systematic investigation of MCMC decryption results with many different choices of texts was undertaken by Connor [1], so we do not repeat that here.)

Text	Author	Publication Date
War and Peace	Leo Tolstoy	1869
Oliver Twist	Charles Dickens	1838
Pride and Prejudice	Jane Austen	1813
Ice Hockey (Wikipedia Page) [7]	Wikipedia	2010

Table 5: Cipher texts and reference texts used in our attacks.

For simplicity, we first convert all letters to upper case, and remove or convert to spaces all non-alphabetic characters. So in total we have 27 characters (26 upper case English alphabet letters plus one space character), which we number from 0 to 26.

For each attack algorithm we consider, we run the encryption and the decryption process 100 separate times. In each such run, a random key is generated to encrypt the plain text, and the attack is then performed on the cipher text. At the end of the attack, we compare the decrypted text with the plain text. If the decrypted text is the same as the cipher text, it is a successful run. Obviously, the more successful runs out of 100, the better has our algorithm performed.

Even if a run is not completely successful, it is still true that if we successfully guessed "most" of the letters, i.e. our decryption was "mostly" successful, then this may still be helpful because the remaining cipher text can probably then be determined by human intervention. For this reason, we also want a definition of "accuracy" to measure how close the decrypted text is to the plain text.

For a substitution ciphers, the accuracy is defined as $\frac{m_s}{n_s}$, where m_s is the number of letters correctly revealed, and n_s is the number of available letters in the plain text (usually n_s is 26, but it may be less than 26 for short cipher text). A letter is said to be correctly revealed if the position of its first appearance in the plain text is the same as that of the decrypted text.

(Of course, it would be possible to modify this definition to weight the value of the letters according to their frequency within the text. For example, perhaps it is more important to correctly decode 'E' than to correctly decode 'Z'. In fact, such a modification would probably make our algorithms appear even *more* successful, since more frequent letters are easier to decode correctly. On the other hand, different texts have different letter frequencies, so such a modified definition would vary from text to text making between-text comparisons less meaningful. Overall we have decided for simplicity to stick with the simple $\frac{m_s}{n_s}$ definition above, while recognising that other definitions are possible though they will probably not affect our results very much.)

For a transposition ciphers, we define accuracy as $\frac{m_t}{n_t}$, where n_t is equal to the key length minus 2, and m_t is the number of letters correctly placed in one period (the key length). A letter is said to be correct positioned if it has the same neighbors in the decrypted text as in the plain text. We do not count the letters in the start and end positions as they only have one neighbor.

We also measure how long it takes for our attacks to run, since a good attack should finish within a reasonable time. Our program is written in C++ and was run on a MacBook Pro with the system configuration as in Table 6.

CPU	2.26 GHz Intel Core 2 Duo
Memory	4GB 1067 MHz DDR3 Memory
OS version	Mac OS X Version 10.6.3
Compiler	g++ i386-apple-darwin10-g++-4.2.1

Table 6: System configuration of the machine running the attacks.

3 Attacks on Substitution Ciphers

We now consider attacks on substitution ciphers, in which an unknown permutation is applied to the 26 letters of the English alphabet. Following [3, 1], we use MCMC algorithms as in the previous section, and find good results.

Our Markov chain state space now consists of all the $26! \doteq 4 \times 10^{26}$ possible permutations of 26 letters. We let the initial state be the identity permutation 'ABCD...XYZ' (so the decrypted text using this key is identical to the cipher text itself).

A key part of the MCMC algorithm is to define a proposal so the chain is detailed balanced and guaranteed to converge to its stationary distribution. Similar to [3, 1], we propose a new key by swapping 2 randomly selected letters in the current key. So, each such swap has proposal probability $1/n^2$. Note that these proposals are symmetric. (For simplicity, and to guarantee aperiodicity, we allow our program to propose swapping a letter with itself, e.g. swapping 'A' and 'A', even though such proposals will not change the chain's state.)

It follows easily that this algorithm will converge to solutions with "approximately" maximal score functions, in the following sense (which will be improved in Theorem 3 below):

Theorem 2. *Let $\{X_n\}$ be the sequence of decryption keys produced by the above algorithm, using the score function $\pi(x)$ from (2). Let $M = \max_{x \in \mathcal{X}} \pi(x)$. Then the Markov chain $\{X_n\}$ is irreducible and aperiodic, and for any $\epsilon > 0$,*

$$\lim_{n \to \infty} \mathbf{P}(\pi(X_n) > M - \epsilon) \;=\; \sum \{\pi(y) : y \in \mathcal{X},\ \pi(y) > M - \epsilon\}\,. \tag{3}$$

In particular, if $\sum\{\pi(y) : y \in \mathcal{X},\ \pi(y) > M - \epsilon\}$ is close to 1, then after many iterations the score functions produced by the algorithm will probably be within ϵ of being maximal.

Proof. Since we added one to each $r(\beta_1, \beta_2)$, it follows from (2) that $\pi(x) > 0$ for all decryption keys x. This implies that we always have $(\pi(y)/\pi(x))^p > 0$, so that every proposed swap has positive probability of being accepted. Hence, since every permutation can be obtained from every other by a sequence of pairwise transpositions, this implies that the Markov chain $\{X_n\}$ is irreducible. In addition, since we allow the algorithm to propose swapping a letter with itself (or, since some proposed swaps have positive probability of being rejected), the Markov chain has positive holding probability, and hence is aperiodic. The conclusion (3) then follows from Proposition 1, with $A = \{y \in \mathcal{X} : \pi(y) > M - \epsilon\}$. ∎

For the above decryption algorithm, we next try adjusting various parameters to see which tuning allows the algorithm to perform optimally.

3.1 Number of Iterations

Table 7 shows that by increasing number of iterations, we improve the accuracy and the number of successful runs. But the accuracy doesn't change much after 10,000 iterations. At this point, although the accuracy is quite high (greater than 90%) which mean most of the runs were very close to the correct result, the number of completely successful runs is fairly low (around 50–60 out of 100). Next we try to improve the algorithm by tuning different parameters.

iterations	accuracy	no. of successful runs
1,000	0.5196	0
2,000	0.7732	19
5,000	0.9060	47
10,000	0.9064	51
20,000	0.9348	59
50,000	0.8932	54

Table 7: Results from initial attempt to decrypt substitution ciphers using bi-grams, with different numbers of MCMC iterations.

3.2 Tuning the Scaling Parameter

The scaling parameter can be very important in the MCMC algorithm. Larger scaling parameters give lower acceptance rates. But if the acceptance rate is too low, the chain is moving too slowly, and it will take too long to converge. Smaller scaling parameters gives higher acceptance rates. But if acceptance rate is too high, the chain will move too often and may not always stay in the stationary distribution.

To investigate this question, we ran simulations with various choices of the scaling parameter p. The results are in Table 8.

scaling parameter	acceptance rate	accuracy	no. of successful runs
0.05	0.27	0.2664	0
0.1	0.12	0.6184	0
1	0.04	0.9064	51
10	0.04	0.8520	80
20	0.04	0.8920	85
50	0.04	0.9156	87
100	0.04	0.8288	74

Table 8: Results of bi-gram attacks for substitution ciphers after 10,000 iterations, for different choices of the scaling parameter.

We see from Table 8 that in this case, certainly the scaling parameter should be at least 1. If we increase the acceptance rate by lowering the scaling parameter, the chain will not converge well (e.g. with $p = 0.05$, on average it only revealed 26.6% of the cipher text after 10,000 iterations). Larger values of p do lead to a larger number of successful runs, but not to significantly greater accuracy. (This is because large p forces the algorithm to remain right at a local mode, while smaller p allows for greater flexibility of the algorithm to remain "near" a mode.) We shall see later that for transposition ciphers the choice $p = 1$ is preferable. Thus, to avoid confusion and incompatibility when we try to combine substitution and transposition ciphers in Section 5, for simplicity we leave the scaling parameter set to 1, leading to an acceptance rate of 0.04.

3.3 Remembering the Best Score Function

The above runs had our algorithm return the *final* decryption key from the run, i.e. whatever key the Markov chain ends up at after a full run of (say) 10,000 iterations.

However, we found that many of our runs revealed the plain text (e.g. "THE PROJECT") in the middle of the run, but then later jumped away from it (e.g. "THE PROZECT"). We know that larger score functions usually indicate better solutions. So, instead of having our algorithm return the *final* decryption key from the run, we have it return *whichever* decryption key from whichever iteration which gave the largest log score function.

For this modified algorithm, a stronger optimality theorem follows:

Theorem 3. *Let $\{X_n\}$ be the sequence of decryption keys produced by the above algorithm, using the score function $\pi(x)$ from (2), with the modification of remembering the best score function. Let $M = \max_{x\in\mathcal{X}} \pi(x)$. Then if $B_n := \max(\pi(X_1), \pi(X_2), \ldots, \pi(X_n))$ is the best score function from the first n iterations, then*

$$\lim_{n\to\infty} B_n \;=\; M \qquad \text{with probability 1}\,.$$

That is, the algorithm produces score functions which converge in the limit to the maximal possible value.

Proof. Since the Markov chain $\{X_n\}$ is irreducible and aperiodic (by Theorem 2), it follows from the Markov chain law of large numbers (see e.g. Theorem 3 of [19]) that for any subset $A \subseteq \mathcal{X}$,

$$\lim_{n\to\infty} \frac{1}{n} \#\{i : 1 \leq i \leq n,\ X_i \in A\} \ = \ \pi(A) \qquad \text{with probability 1}\,.$$

In particular, if $\pi(A) > 0$, then with probability 1, $\#\{i : 1 \leq i \leq n,\ X_i \in A\} > 0$ for sufficiently large n, i.e. the chain will eventually enter the subset A.

To continue, let $\epsilon > 0$, and let $A = \{y \in \mathcal{X} : \pi(y) > M - \epsilon\}$ as before. Then $\pi(A) > 0$ by definition of M. Hence, with probability 1, the chain will eventually enter A, at some random iteration n. At this iteration, we will have $B_n > M - \epsilon$ by the definition of A. Furthermore, since $\{B_n\}$ is a non-decreasing sequence by construction, it then follows that we will have $B_n > M - \epsilon$ for all subsequent iterations n as well. Since this is true for all $\epsilon > 0$, this implies that $B_n \to M$. (In fact, since the state space $\mathcal{X}$ is finite, it is possible to instead let A be the set on which the score function actually attains its maximum. But we prefer the above argument since it is still valid on continuous state spaces as well.) ∎

Remark. Of course, even Theorem 3 gives no indication of *how large* the iteration number n has to be before B_n is close to M. Such "non-asymptotic convergence rate" issues are in general quite challenging, see e.g. [14] and references therein. As noted above, Connor [1] did attempt some convergence rate analysis related to this algorithm, but only for the very special case of no data, i.e. assuming that all proposed moves always get accepted. The general case is a much harder problem, which we leave to possible future work. ∎

We tried re-running our algorithm with this new modification (of remembering the best score function). The results are presented in Table 9. Comparison with Table 7 shows that the new modification leads to significantly better results.

iterations	accuracy	no. of successful runs
1,000	0.5300	1
2,000	0.7716	20
5,000	0.9172	87
10,000	0.9312	90
20,000	0.9148	87
50,000	0.9488	93

Table 9: Results of bi-gram attacks on substitution ciphers, when we return whichever key maximizes the score function.

3.4 How Much Cipher Text is Needed

Usually we use the entire cipher text when computing the score function (2) at each iteration. We can ask whether it is more efficient, and of comparable accuracy, to compute the score function at each iteration using just a (random) subset of the cipher text. Table 10 indicates that in this case, the time spend on the decryption is essentially independent of the length of the cipher text used. On the other hand, the accuracy is already quite high (over 93%) when using just 2,000 characters of cipher text.

cipher text	accuracy	no. of successful runs	duration (in seconds)
1,000	0.7143	0	0.4441
2,000	0.9312	90	0.4442
full cipher text	0.9831	97	0.4381

Table 10: Results from attacks on substitution ciphers using bi-gram, when using different amounts of cipher text. Each run uses 10,000 iterations, and returns whichever key maximize the log score.

These results suggest that for simple substitution ciphers, it does not much matter (for either speed or accuracy) whether we use just 2,000 characters of cipher text, or the entire cipher text. However, since our main interest is in transposition-related ciphers for which speed is much more effected (see below), for our final attack we use just 2,000 characters of cipher text for simple substitution ciphers as well.

3.5 Independent Repetitions

Experimentation indicates mixed result when using just 2000 randomly-chosen consecutive characters from the cipher text for the attack (Table 11). That is, some selections from the cipher text are better for decryption than others.

cipher text starting position	accuracy	no. of successful runs	duration (in seconds)
574798	0.9492	0	0.3906
416031	0.9488	90	0.3933
243158	0.9840	97	0.3932
551774	0.9452	0	0.3940
223511	0.9596	94	0.3939

Table 11: Results of attacks on substitution ciphers using bi-gram, depending on the position of cipher text used in the attack. Each run uses 10,000 iterations, and 2000 characters of cipher text, and returns whichever key maximize the log score.

This suggests that our final attack should, instead of using just one run, use several independent repeated runs, and return whichever final result has the largest score function (from (2) computed using the *entire* cipher text). We use this approach in our final attack below.

3.6 Tri-gram Attack

As a final check, we tried modifying the previous MCMC algorithm to use tri-grams (triple letters frequencies) instead of bi-grams. That is, we replace the score function (2) by:

$$\pi(x) = \prod_{\beta_1,\beta_2,\beta_3} r(\beta_1,\beta_2,\beta_3)^{f_x(\beta_1,\beta_2,\beta_3)},$$

where β_1, β_1, β_3 are all possible three-characters combinations, and where $r(\beta_1,\beta_2,\beta_3)$ and $f_x(\beta_1,\beta_2,\beta_3)$ are now the corresponding triple letter frequencies of the reference text and the decrypted text respectively.

This new tri-gram attack also works (Table 12), but the result is not as good as the attack using the bi-grams. Therefore, we stick with bi-grams for the final version of our attack.

iterations	accuracy	no. of successful runs	duration (in seconds)
1,000	0.5336	1	0.9089
2,000	0.7652	46	1.6892
5,000	0.7896	75	4.0435
10,000	0.8920	87	7.9283

Table 12: Results of attacks on substitution ciphers by tri-gram, for different numbers of iterations. Each run returns whichever key maximize the log score.

3.7 Attack for Substitution Ciphers – Preliminary Version

Based on the above experimentation, we take the preliminary version of our attack to involve 10,000 iterations, with scaling parameter 1, and with cipher text length 2,000.

To investigate how our program works, we apply this attack to different combinations of cipher text and reference text. The results are presented in Table 13.

cipher text	reference text	accuracy	no. of successful runs
Oliver Twist	*War and Peace*	1.0000	100
Pride and Prejudice	*War and Peace*	1.0000	100
Ice Hockey (Wikipedia Page)	*War and Peace*	1.0000	100
Pride and Prejudice	*Oliver Twist*	1.0000	100
War and Peace	*Oliver Twist*	0.9977	97
Ice Hockey (Wikipedia Page)	*Oliver Twist*	0.9869	83
War and Peace	*Pride and Prejudice*	0.9977	97
Ice Hockey	*Pride and Prejudice*	0.9977	97
Oliver Twist	*Pride and Prejudice*	0.9985	98
Pride and Prejudice	*Ice Hockey (Wikipedia Page)*	1.0000	100
War and Peace	*Ice Hockey (Wikipedia Page)*	0.9938	92
Oliver Twist	*Ice Hockey (Wikipedia Page)*	0.9750	74

Table 13: Results of our preliminary attack on substitution ciphers with key length 20, for different choices of cipher text and reference text. Each run uses 5 repetitions of 10,000 iterations each, with

2000 characters of cipher text, and scaling parameter 1, and returns whichever key gives the highest score function.

We see from the table that our final attack algorithm performed very well, often achieving perfect or near-perfect scores. The only sub-par performances arose when using the *Ice Hockey* Wikipedia page, which is much shorter than the three novels (less than 8,000 words) and thus provides insufficient text for our algorithm to perform well. (Furthermore it was written in the modern era so it may have somewhat different language usage as well.)

To further investigate the sub-par performance when using the *Ice Hockey* Wikipedia page, we consider some additional improvements. Firstly, we try increasing the number of repetitions from 5 to 10. Secondly, we consider the possibility of *first* using a uni-gram attack, as a "starting point" for the later bi-gram attacks, since such a uni-gram attack is not sufficient on its own but is still a quick and easy way to get closer to a true solution before beginning. (This innovation was also employed by Connor [1], to find an "intelligent" starting state before running the more complicated algorithms; it is of only minor importance here but will be much more important when attacked substitution-transposition product ciphers in Section 5 below.) The results are presented in Table 14.

repetitions	uni-gram?	accuracy	no. of successful runs
1	N	0.8608	26
5	N	0.9750	74
10	N	0.9962	96
1	Y	0.9081	33
5	Y	0.9838	84
10	Y	0.9981	98

Table 14: Results of attacks on substitution ciphers with key length 20, using *Oliver Twist* as cipher text, and *Ice Hockey (Wikipedia Page)* as reference text. Each run uses a certain number of bi-gram attack repetitions of 10,000 iterations each, either with or without an initial uni-gram attack. Each run uses 2000 characters of cipher text, and scaling parameter 1, and returns whichever key gives the highest score function.

We see from Table 14 that for this challenging case (with *Ice Hockey (Wikipedia Page)* as reference text), it does indeed help to increase the number of repetitions from 5 to 10. It also helps *slightly* to begin with a uni-gram attack. Thus, to maximise the power and flexibility of our algorithm, we use both of these improvements in our final version of the algorithm.

3.8 Attack for Substitution Ciphers – Final Version

Based on the above investigations, our final algorithm to attack substitution ciphers is:

- Run the uni-gram attack for substitution cipher on the original cipher text.
- Randomly select 2000 cipher text from available cipher text
- Run a bi-gram attack (with scaling parameter 1) for 10,000 iterations.
- Apply the decode function to the full cipher text, to calculate the score function for the full text, remembering which decode function gives the highest score function.

- Repeat the above procedure 10 times.
- The final key is the key which gives the highest score function.

We ran this final algorithm on all the same cipher/reference pairs as in the previous section. Our results were very successful, and are presented in Table 15.

cipher text	reference text	accuracy	no. of successful runs
Oliver Twist	*War and Peace*	1.0000	100
Pride and Prejudice	*War and Peace*	1.0000	100
Ice Hockey (Wikipedia Page)	*War and Peace*	1.0000	100
Pride and Prejudice	*Oliver Twist*	1.0000	100
War and Peace	*Oliver Twist*	1.0000	100
Ice Hockey (Wikipedia Page)	*Oliver Twist*	0.9977	97
War and Peace	*Pride and Prejudice*	1.0000	100
Ice Hockey	*Pride and Prejudice*	1.0000	100
Oliver Twist	*Pride and Prejudice*	1.0000	100
Pride and Prejudice	*Ice Hockey (Wikipedia Page)*	1.0000	100
War and Peace	*Ice Hockey (Wikipedia Page)*	0.9992	99
Oliver Twist	*Ice Hockey (Wikipedia Page)*	0.9981	98

Table 15: Results of our final attack on substitution ciphers with key length 20, for different choices of cipher text and reference text. Each run uses a uni-gram attack followed by 10 repetitions of a bi-gram attack of 10,000 iterations each and scaling parameter 1, using 2000 characters of cipher text, and returns whichever key gives the highest score function.

4 Attacks on Transposition Ciphers

We now turn our attention to Transposition Ciphers. Since Transposition Ciphers only move letters around, there is no change to the frequencies of single letters, so we certainly can't use a uni-gram attack to break it. Instead, we concentrate on bi-gram attacks (i.e., again using the frequencies of pairs of letters).

The state space depends on the key length of the transposition cipher. For key length k, there are $k!$ possible decryption keys, corresponding to all possible permutations of $0, 1, 2, \ldots, k-1$. We again choose the initial decryption key to be the identity permutation, so the decrypted text using this key is identical to the cipher text.

The score function is thus again the same as in (2), and the algorithm and acceptance rate are still the same as in Section 2. The only potential difference from the bi-gram attack for the substitution cipher concerns the proposal distribution, as we now discuss.

4.1 Swap Moves versus Slide Moves

For the proposal, first we tried the same swap moves as in our substitution cipher attacks. But we found the swap moves are not very efficient in some cases. For example, suppose $k = 7$ and a typical block of plain text is "PROJECT", and the current decryption key gives a decrypted text

"ROJECTP". Then this is very close to the correct answer, but we need at least 6 swap moves find the correct decryption key from here.

Instead of proposing individual swaps, we can instead propose a "slide move" of randomly taking out one decryption position and inserting it back to a random location among the remaining decryption positions. For example, suppose the key length $k = 8$, and decryption position 3 is taken out and inserted back in position 6. Then the decryption key "01234567" will become "01245637" by this slide move.

Sliding moves of a single decryption position work pretty well for small key lengths. But as the key length gets larger, these moves become less efficient. For example, suppose $k = 12$ and a typical block of plain text is "THE_PROJECT_" (where "_" indicates a space), but the current decryption key generates corresponding decrypted text "_PROJECT_THE". To get the correct key using single letter slide moves, we need at least 3 moves (move each character in "THE_" to the left of "_PROJECT"). But each such move may lower the score function since we are breaking the word "THE", so it will more likely be rejected by the algorithm, making this a difficult feat for our algorithm to perform.

This can be solved by using slide move involving entire blocks of decryption positions. That is, we select a random contiguous sequence of decryption positions, which we remove and insert back somewhere within the remaining decryption positions. Thus, for the "_PROJECT_THE" example, the word "THE" can be moved to the left of "_PROJECT" by just one move.

Formally speaking, for a key length k, the new proposal is to slide move a block of n decryption positions from position k_1 to k_2, where $n \sim \text{Uniform}\{0, \ldots, k-2\}$, $k_1 \sim \text{Uniform}\{0, \ldots, k-n+1\}$, and $k_2 \sim \text{Uniform}\{0, \ldots, k-n+1\}$. For example, with key length $k = 8$, we might choose $n = 2, k_1 = 3, k_2 = 6$, corresponding to a proposal to move 2 letters from position 3 to position 6, i.e. to change "01234567" to "01256347" (since "34" is moved to after "6").

Tables 16 and 17 each compare algorithms using swap moves, single letter slide moves, and block slide moves. With key length $k = 10$ and 1,000 iterations as in Table 16, we see some improvement of the accuracy and success rate by switching from swap move to slide move algorithm: the addition of block letter slide move increases the accuracy of from 66% to 96%, and the complete successes from 17 to 90 out of 100 runs. With key length $k = 20$ and 5,000 iterations as in Table 17, the benefit of using slide moves is also apparent: swap moves and single letter slide moves can't achieve a single successful run, but block letter slide moves still perform fairly well with an accuracy of 90% and complete success in 59 out of 100 runs.

move	accuracy	no. of successful runs
swap move	0.6550	17
single letter slide move	0.9525	83
block letter slide move	0.9587	90

Table 16: Comparison of results for attacks on transposition ciphers of key length 10 when the proposals are swap moves, single-letter slide moves, and block-letter slide moves. Each run uses 1,000 iterations and 1,000 characters of cipher text, with scaling parameter 1.

move	accuracy	no. of successful runs
swap move	0.4383	0
single letter slide move	0.6333	0
block letter slide move	0.8961	59

Table 17: Comparison of results for attacks on transposition ciphers of key length 20 when the proposals are swap moves, single-letter slide moves, and block-letter slide moves. Each run uses 5,000 iterations and 1,000 characters of cipher text, with scaling parameter 1.

4.2 The Scaling Parameter and the Best Score Function

We again experimented with different choices of the scaling (inverse temperature) parameter. We found that small choices of this parameter lead to poor performance, while values equal to or greater than 1 lead to approximately equally good performance (and acceptance rates around 0.44), see Table 18. We again choose our final scaling parameter to be 1 since that gives the highest accuracy and produces the highest percentage of successful runs.

power	acceptance rate	accuracy	no. of successful runs
0.01	0.975	0.013	0
0.1	0.690	0.084	0
1	0.439	0.9694	87
10	0.436	0.9650	85
20	0.436	0.9644	85
50	0.432	0.9572	83
100	0.425	0.9583	83
1,000	0.431	0.9489	81
100,000	0.440	0.9489	81

Table 18: Results of attacks on transposition ciphers with key length 20, for different choices of the scaling parameter. Each run uses 10,000 iterations, and 1,000 characters of cipher text.

Recall that with substitution ciphers, we improved our attack algorithm by remembering whichever key gave the highest score function. For transposition ciphers, this turns out to be less important, since (with scaling parameter 1) it is very rare for the chain to ever jump from higher to lower score functions. Indeed, in each of our runs above, the last key was also the key which gives the highest score. However, we still choose to remember the highest score, since this doesn't cost much overhead and it guarantees that we will always return the key which gave the best result.

With this modification, since the original chain is still irreducible and aperiodic, we have a precise analog of Theorem 3:

Theorem 4. *If* $B_n := \max(\pi(X_1), \pi(X_2), \ldots, \pi(X_n))$ *is the best score function from the first* n *iterations of the above algorithm, and* $M = \max_{x \in \mathcal{X}} \pi(x)$*, then*

$$\lim_{n \to \infty} B_n \;=\; M \qquad \text{with probability 1}\,.$$

4.3 Amount of Cipher Text Needed

We next investigated the extent to which the accuracy and success rate of the algorithm are affected by the length of the cipher text used to compute the score function. This question is more relevant here than for substitution ciphers, since now the speedup from using less cipher text is much more significant.

We found (Table 19) that we certainly need at least 500 characters of cipher text to break a transposition cipher with key length 20 in 10,000 iterations. More precisely, it appears that the 2000 characters of cipher text is the best choice, since that leads to very high accuracy (over 99%) and successful runs (95%), and using more cipher text requires significantly more time to process but leads to very marginal benefits.

cipher text length	accuracy	no. of successful runs	duration (in seconds)
100	0.0789	0	0.4640
200	0.2844	0	0.5093
500	0.9250	72	0.6526
1,000	0.9694	87	0.8856
2,000	0.9933	95	1.3500
5,000	0.9917	96	2.7557
10,000	0.9861	92	5.079

Table 19: Results of attacks on transposition ciphers with key length 20, when using different numbers of characters of cipher text. Each run uses 10,000 iterations, with scaling parameter 1.

We also found that using a different section of cipher text of the same length leads to very similar results (Table 20), showing a certain stability of this approach.

cipher text starting position	accuracy	no. of successful runs
830080	0.9917	97
254640	0.9933	96
568780	0.9906	96
634220	0.9972	98
366660	0.9928	96

Table 20: Results of attacks on transposition ciphers with key length 20, when using 2000 characters of cipher text starting from different positions in the text. Each run uses 10,000 iterations, with scaling parameter 1.

For a different perspective, we also considered the amount of cipher text required to achieve at least 95% accuracy for a fixed number (10,000) of iterations, but with different key lengths.

key length	cipher text length	accuracy	no. of successful runs	duration (in seconds)
5	20	1.0000	100	0.4076
10	500	0.9700	96	0.6332
20	1,000	0.9694	87	0.8856
30	5,000	0.8971	27	2.7245
30	10,000	0.9025	30	5.0573

Table 21: Amount of cipher text required to achieve at least 95% accuracy with 10,000 iterations, for various key lengths.

We see from this table that if the key length is only 5, then we only need 20 characters of cipher text to break it! On the other hand, for key length as large as 30, simply including more cipher text does not help, and in fact more iterations would be required to achieve success.

4.4 Number of Iterations

With the above optimal block slide move proposal, and scaling parameter 1, and cipher text size 2000 for transposition key length 20, we next investigate the extent to which we can increase the accuracy by running more iterations. Table 22 shows that the accuracy and success rate increase steadily as we use longer runs up to about 20,000 iterations, after which there is little further gain. So, we choose 20,000 iterations as our optimal run length.

no. of iterations	accuracy	no. of successful runs
1,000	0.6878	3
2,000	0.7944	17
5,000	0.9544	73
10,000	0.9911	95
20,000	1.0000	100
50,000	1.0000	100

Table 22: Results of attacks on transposition ciphers with key length 20, when using 2000 characters of cipher text and scaling parameter 1, for different numbers of iterations.

More generally, for different transposition key lengths, we tried increasing the number of iterations to get a reasonable accuracy rate. Our results are presented in Table 23.

key length	no. of iterations	accuracy	no. of successful runs	duration (in seconds)
10	2,000	0.9962	97	0.3424
20	10,000	0.9911	95	1.3736
30	50,000	0.9957	98	6.4034
40	50,000	0.9613	70	6.3969
50	100,000	0.9648	68	12.8497

Table 23: Results of attacks on transposition ciphers with different key lengths, when using 2000 characters of cipher text and scaling parameter 1, when using different numbers of iterations.

On the other hand, increasing the number of iterations alone is not sufficient to overcome all difficulties. We illustrate this more precisely using just 1000 characters of cipher text. In this case, we already know that the accuracy will not be great. However, it is also true that this accuracy will not increase very quickly as we run more iterations (Table 24).

no. of iterations	accuracy	no. of successful runs
1,000	0.6333	1
2,000	0.7583	12
5,000	0.8961	59
10,000	0.9694	87
20,000	0.9872	95
50,000	0.9856	93

Table 24: Results of attacks on transposition ciphers with key length 20, when using just 1000 characters of cipher text for different numbers of iterations (still with scaling parameter 1), indicating poor performance even after many iterations.

For a different perspective, we next consider how many iterations are needed to achieve at least 95% accuracy for different cipher text lengths.

cipher text length	iterations	accuracy	no. of successful runs
100	200,000	0.2722	0
200	100,000	0.9450	83
500	17,000	0.9559	88
1,000	10,000	0.9694	87
2,000	6,000	0.9411	71
5,000	7,000	0.9567	78
10,000	7,000	0.9533	76

Table 25: Number of iterations required to achieve at least 95% accuracy, for various cipher text lengths.

4.5 Independent Repetitions

As with substitution ciphers, we may wish to use several independent repeated runs of our attack, and return as our final answer whichever of the results has the largest score function (from (2) computed using the *entire* cipher text).

To consider the extent to which multiple independent repetitions might help with this problem, we compare a single long run of 50,000 iterations, with 5 repetitions of a run of 10,000 which returns the decryption key which gives the highest score. Our results are presented in Table 26. We see that multiple shorter runs are consistently better than one very long run, increasing the percentage of successful runs from 93% to 100%. This makes sense since we have already seen that the success rate with 1000 characters of cipher text for 1 run of 10,000 iterations is about 87%. So, if we use 5 independent such runs, then the probability not getting a correct answer is only $(1 - 0.87)^5 = 0.0037\%$ which is very low.

no. of repetitions x no. of iterations	accuracy	no. of successful runs
20 x 2,500	0.9972	98
10 x 5,000	1.0000	100
5 x 10,000	1.0000	100
2 x 25,000	1.0000	100
1 x 50,000	0.9856	93

Table 26: Results of attacks on transposition ciphers with key length 20, when using just 1000 characters of cipher text (still with scaling parameter 1), when dividing up the 50,000 total iterations into multiple independent repetitions.

4.6 Attack for Transposition Ciphers – Final Version

Based on the above investigations, we propose the following as our final algorithm for the attack of the transposition cipher (when the key length equals 20).

- Randomly select 2000 cipher text from the available cipher text.
- Attack the selected cipher text with the bi-gram score function, using block slide proposal moves and parameter value 1, for 10,000 iterations.
- Apply the decryption key to the full cipher text and calculate the log score for full decrypted text.
- Repeat the above procedure 5 times; the final result is whichever iteration from whichever repetition which gives the highest score.

To investigate how our program works, we apply the method to different cipher text and reference text. The results are presented in Table 27, which shows that the results are very good, leading to perfect runs in every case.

cipher text	reference text	accuracy	no. of successful runs
Oliver Twist	*War and Peace*	1.0000	100
Pride and Prejudice	*War and Peace*	1.0000	100
Ice Hockey (Wikipedia Page)	*War and Peace*	1.0000	100
Pride and Prejudice	*Oliver Twist*	1.0000	100
War and Peace	*Oliver Twist*	1.0000	100
Ice Hockey (Wikipedia Page)	*Oliver Twist*	1.0000	100
War and Peace	*Pride and Prejudice*	1.0000	100
Ice Hockey (Wikipedia Page)	*Pride and Prejudice*	1.0000	100
Oliver Twist	*Pride and Prejudice*	1.0000	100
Pride and Prejudice	*Ice Hockey (Wikipedia Page)*	1.0000	100
War and Peace	*Ice Hockey (Wikipedia Page)*	1.0000	100
Oliver Twist	*Ice Hockey (Wikipedia Page)*	1.0000	100

Table 27: Results of our final attack on transposition ciphers, with key length 20, when using 2000 characters of cipher text and scaling parameter 1, with 5 repetitions of 10,000 iterations each. A perfect result is obtained in every case.

4.7 Unknown Transposition Key Lengths

The above transposition cipher attacks all assumed that the key length k was known in advance. In a real decryption situation this might not be the case. So, we now consider using MCMC for decryption of transposition ciphers when the key length itself is unknown.

To attack such ciphers, we use the obvious extension of our previous algorithm. That is, for each possible key length k, we run the entire above algorithm to obtain the best score function for that k, say S_k. We then regard as the "true" key length k_* whichever value of k leads to the best score function, i.e. $k_* = \text{argmax}_k\, S_k$, and declare the "true" decryption solution to be the optimal solution for that key length k_*.

More formally, if the keylength k is known to be between $k_{\min}$ and $k_{\max}$, then our algorithm is as follows:

- For $k = k_{\min}, k_{\min}+1, k_{\min}+2, \ldots, k_{\max}$:
  - Randomly select 2000 cipher text from the available cipher text.
  - Repeat the following procedure 5 times, and let the preliminary result S_k be the highest score function out of all iterations in all 5 repetitions:
    - Attack the selected cipher text with the bi-gram score function, using block slide proposal moves and parameter value 1, for the appropriate number of iterations (2,000 if $k \leq 10$; 10,000 if $11 \leq k \leq 20$; 50,000 if $k > 20$).
    - Apply the decryption key to the full cipher text and calculate the log score for full decrypted text.
- Let $k_* = \text{argmax}_k\, S_k$.
- The final result is the solution which gives the score S_{k_*}.

We tested this algorithm on text which was encrypted using an unknown key length k which was generated (independently for each test run) from the uniform distribution on $(k_{\min}, k_{\min}+1, k_{\min}+2, \ldots, k_{\max})$, for appropriate fixed choices of $k_{\min}$ and $k_{\max}$.

We first ran this 100 times with $k_{\min} = 5$ and $k_{\max} = 20$, corresponding to key lengths randomly selected between 5 and 20. We found that the algorithm successfully broke the cipher every single time, i.e. in all 100 runs.

We then ran this 100 times with $k_{\min} = 20$ and $k_{\max} = 30$, corresponding to key lengths randomly selected between 20 and 30. Once again, we found that the algorithm successfully broke the cipher every single time, i.e. in all 100 runs.

We conclude from this that an unknown key length k requires additional computation, to consider separately all possible key lengths, it does not make the decryption fundamentally more difficult, and our algorithm is able to correctly identify k in every case. Thus, in the next section we return to considering fixed key lengths only.

5 Attacks on Substitution-Transposition Ciphers

Substitution-Transposition ciphers have 2 different keys. First the letters are switched using a substitution cipher. Then, the characters are moved around using a transposition cipher. The length of the substitution key is 26 as usual. We consider different lengths of the transposition key, k, as in the previous section. To break Substitution-Transposition ciphers, we shall reuse and

combine the MCMC attack algorithms previously developed for the substitution cipher and the transposition cipher. In particular, we shall again attempt to maximize the same score function (2).

5.1 First Attempt

For illustrative purposes, we first try to break a substitution-transposition cipher with transposition key length 10.

As our initial algorithm, we simply combine our two previous algorithms directly, by first running a bi-gram attack as for a substitution cipher, and then running a bi-gram attack as for a transposition cipher. We use the optimal values of parameters from our previous attacks: specifically, we use 2000 cipher text characters, with scaling parameter 1, and run 10,000 iterations of the bi-gram attack on a substitution cipher, followed by 2,000 iterations of the bi-gram attack on a transposition cipher.

The results are presented in Table 28. We see that we are not able to get good results, even by increasing the number of iterations. And, switching the sequence of the two attacks also does not help.

no. of iterations (substitution/transposition)	accuracy	no. of successful runs
10,000/2,000	0.0600	3
10,000/5,000	0.0587	3
10,000/10,000	0.0338	2

Table 28: Results of our first attack on substitution-transposition ciphers with transposition key length 10, using 2000 characters of cipher text, with a bi-gram substitution cipher attack followed by a bi-gram transposition cipher attack.

Of course, it is not surprising that these results are poor. The basic problem is that our first attack is attempting to break a substitution cipher, but it is working with text which also had an unknown transposition applied. Thus, there is no particular reason that the pair frequencies of the transposed text should in any way match those of the reference text. So, the first attack is doomed from the start. And, of course, if the first attack fails completely, then the second attack is similarly handicapped.

5.2 Multiple Cycles

We have previously seen that with MCMC cipher attacks, sometimes several shorter runs are better than one longer run. Inspired by this, we let our algorithm run for several cycles (Table 29). Each cycle consists of a bi-gram attack on substitution cipher, followed by a bi-gram attack on transposition cipher. We use the result of one cycle as the starting point for next one. Table 29 shows that our results do improve upon increasing the number of cycles. However, the improvement does not continue much beyond 3 cycles: we get just 63% accuracy on average and 62 success out of 100 runs even after 10 cycles.

cycles	accuracy	no. of successful runs	duration (in seconds)
1	0.0600	3	0.7446
2	0.3713	35	1.4919
3	0.5462	52	2.5222
5	0.5250	52	3.6850
10	0.6300	62	7.4400

Table 29: Results of attacks on substitution-transposition ciphers with transposition key length 10, for various numbers of cycles. Each cycle uses 2000 characters of cipher text, and consists of a 10,000-iteration bi-gram substitution cipher attack followed by a 2,000-iteration bi-gram transposition cipher attack.

It seems that, even with multiple cycles, the problem remains that if the substitution attack fails massively, then the transposition attack has little chance of success, and vice-versa. This is a sort of "chicken and the egg" problem: if *one* of the attacks were successful, or even nearly successful, then the other attack would perform well and the problem would quickly be solved. The question remains, how can we get initial near-success? We consider that next.

5.3 Using a Uni-gram Attack for Initialization

Recall that the standard uni-gram attack on substitution ciphers is quick and simple, but it is not terribly accurate, i.e. it tends to only partially reveal the original text. We can make use of this in the attack for breaking the substitution-transposition cipher. Even though the uni-gram attack does not reveal all the letters, it can quickly provide a very good starting point.

We saw earlier that for simple substitution ciphers, beginning with a uni-gram attack was helpful, but only a little bit so. However, since substitution-transposition ciphers are so much more challenging, the additional benefit of starting with a uni-gram attack could be much more significant.

Inspired by this, we modify our algorithm to *first* run the uni-gram attack for substitution ciphers, and *then* run multiple cycles of bi-gram attacks. The results are presented in Table 30 (for transposition key length 10, up to 2 cycles) and Table 31 (for transposition key length 20, up to 3 cycles), and indicate very high success rates in both cases.

cycles	accuracy	no. of successful runs	duration (in seconds)
1	0.7937	68	0.8369
2	1.0000	100	1.5800

Table 30: Results of attacks on substitution-transposition ciphers with transposition key length 10, after initializing with a uni-gram attack, for various numbers of cycles. Each cycle uses 2000 characters of cipher text, and consists of a 10,000-iteration bi-gram substitution cipher attack followed by a 2,000-iteration bi-gram transposition cipher attack.

cycles	accuracy	no. of successful runs	duration (in seconds)
1	0.2394	0	1.8440
2	0.9133	82	3.6105
3	0.9906	98	5.3200

Table 31: Results of attacks on substitution-transposition ciphers with transposition key length 20, after initializing with a uni-gram attack, for various numbers of cycles. Each cycle uses 2000 characters of cipher text, and consists of a 10,000-iteration bi-gram substitution cipher attack followed by a 10,000-iteration bi-gram transposition cipher attack.

5.4 Remembering the Best Score Function

In each of the above cases, we again used the modification of always remembering the best score function so far. With this modification, since these chains are again irreducible and aperiodic, we again have the exact analog of Theorems 3 and 4:

Theorem 5. *If* $B_n := \max(\pi(X_1), \pi(X_2), \ldots, \pi(X_n))$ *is the best score function from the first* n *iterations of the above algorithm, and* $M = \max_{x \in \mathcal{X}} \pi(x)$*, then*

$$\lim_{n \to \infty} B_n \;=\; M \qquad \text{with probability 1}\,.$$

5.5 Attack for Substitution-Transposition Ciphers – Final Version

Putting the above together, we propose the following MCMC algorithm to attack the substitution-transposition cipher.

- Randomly select 2000 cipher text from the available cipher text.
- Run the uni-gram attack for substitution cipher on the original cipher text.
- Then, for several cycles (3, for key length 20):
  - Run the bi-gram attack for transposition cipher on the resulting text, for an appropriate number of iterations (10,000, for key length 20).
  - Run the bi-gram attack for substitution cipher on the resulting text, for an appropriate number of iterations (10,000, for key length 20).
- The final result is whichever iteration from whichever repetition which gives the highest score.

We ran this final algorithm on randomly-generated substitution-transposition ciphers with transposition key length 20, with different combinations of cipher text and reference. The overall results are presented in Table 32. Once again, we find that runs using the short and modern text *Ice Hockey (Wikipedia Page)* are worse than using other text. However, for experiments using the classic novels, the accuracy is always above 80% and the number of successful runs is always above 70, indicating quite good performance for this challenging problem.

cipher text	reference text	accuracy	no. of successful runs
Oliver Twist	*War and Peace*	0.8894	86
Pride and Prejudice	*War and Peace*	0.8433	80
Ice Hockey (Wikipedia Page)	*War and Peace*	0.6811	58
Pride and Prejudice	*Oliver Twist*	0.9100	89
War and Peace	*Oliver Twist*	0.8367	78
Ice Hockey (Wikipedia Page)	*Oliver Twist*	0.7211	62
War and Peace	*Pride and Prejudice*	0.8189	74
Ice Hockey (Wikipedia Page)	*Pride and Prejudice*	0.6811	56
Oliver Twist	*Pride and Prejudice*	0.8244	81
Pride and Prejudice	*Ice Hockey (Wikipedia Page)*	0.7961	74
War and Peace	*Ice Hockey (Wikipedia Page)*	0.6761	64
Oliver Twist	*Ice Hockey (Wikipedia Page)*	0.7778	71

Table 32: Results of our final attacks on substitution-transposition ciphers with transposition key length 20, after initializing with a uni-gram attack, for various texts. Each cycle uses 2000 characters of cipher text, and consists of a 10,000-iteration bi-gram substitution cipher attack followed by a 10,000-iteration bi-gram transposition cipher attack.

Further experimentation using the classic novels indicates that with enough iterations and cycles, the accuracy and success rates remain quite high even with transposition keys up to size 40 (Table 33).

transposition key length	no. of iterations (subst./trans.)	cycles	accuracy	no. of successful runs	duration
10	10,000/2,000	3	1.0000	100	3.93
20	10,000/10,000	3	0.8894	86	5.32
30	10,000/50,000	5	0.8618	85	34.08
40	10,000/100,000	5	0.7645	73	65.51

Table 33: Results of our final attacks on substitution-transposition ciphers with various transposition key lengths, for *Oliver Twist*, using *War and Peace* as the reference text. Each attack first initializes with a uni-gram attack, and then repeats the specified number of cycles. Each cycle uses 2000 characters of cipher text, and consists of a bi-gram substitution cipher attack followed by a bi-gram transposition cipher attack, each of the number of iterations specified.

Overall this indicates quite good performance, even for the difficult substitution-transposition cipher, achieving accuracies and success rates above 70% even with key length 40.

6 Summary

In this paper, we successfully applied MCMC algorithms to break substitution ciphers, transposition ciphers, and even substitution-transposition ciphers. The attacks are based on the frequency analysis of the cipher text together with a reference text, and primarily consist of bi-gram attacks.

We have experimented significantly with such issues as number of MCMC iterations, scaling (inverse temperature) parameter, amount of cipher text to use, number of independent repetitions, swap moves versus slide moves versus block-slide moves, etc., in an attempt to optimise our choices. For substitution-transposition ciphers, we required additional innovations such as repeatedly cycling between substitution-type and transposition-type attacks, and using a simple uni-gram substitution attack as an initialization point.

Overall, our simulations indicate good success of our algorithms. In particular, we are able to break the simple substitution-transposition cipher with accuracy and success rates above 70%, even with transposition key length up to 40. This indicates the potential for MCMC algorithms to provide significant help in deciphering challenging encryptions.

Acknowledgements. We are very grateful to the editors and referees for insightful comments which significantly improved the manuscript.

References

[1] S. Connor (2003), *Simulation and solving substitution codes*, Master's thesis, Department of Statistics, University of Warwick.

[2] P. Diaconis (1988), *Group Representations in Probability and Statistics*. IMS Lecture Series volume **11**, Institute of Mathematical Statistics, Hayward, California.

[3] P. Diaconis (2008), *The Markov Chain Monte Carlo Revolution*. Bull. Amer. Math. Soc., Nov. 2008.

[4] A. Dimovski and D. Gligoroski (2003), *Attacks on the Transposition Ciphers Using Optimization Heuristics*. Proceedings of the XXXVIII International Scientific Conference on Information, Communication & Energy Systems & Technologies, Heron Press, Birmingham, U.K.

[5] P. Garg (2009), *Cryptanalysis of SDES via Evolutionary Computation Techniques*, IJCSIS **1(1)**, May 2009.

[6] W.R. Gilks, S. Richardson, and D.J. Spiegelhalter, ed. (1996), *Markov chain Monte Carlo in practice*. Chapman and Hall, London.

[7] Ice Hockey (Wikipedia Page). `http://en.wikipedia.org/wiki/Ice_hockey`

[8] R.A.J. Matthews (1993), The use of genetic algorithms in cryptanalysis. Cryptologia **17(2)**, 187-201.

[9] A. Menezes, P. van Oorschot, and S. Vanstone, eds. (1996), *Handbook of Applied Cryptography*. CRC Press, Boca Raton, Florida.

[10] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller (1953), *Equations of state calculations by fast computing machines*. J. Chem. Phys. **21**, 1087–1091.

[11] K.S. Ooi and B.C. Vito (2002), *Cryptanalysis of S-DES*. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.79.6617

[12] Project Gutenberg. `http://www.gutenberg.org/`

[13] G.O. Roberts and J.S. Rosenthal (2004), *General state space Markov chains and MCMC algorithms*. Prob. Surv. **1**, 20–71.

[14] J.S. Rosenthal (2002), Quantitative convergence rates of Markov chains: A simple account. Elec. Comm. Prob. **7(13)**, 123–128.

[15] J.S. Rosenthal (2006), *A First Look at Rigorous Probability Theory*, 2nd ed. World Scientific Publishing Company, Singapore.

[16] B. Schneier (1996), *Applied Cryptography, Second Edition*. John Wiley & Sons, New York.

[17] C. E. Shannon (1949), *Communication Theory of Secrecy Systems*. Bell System Technical Journal **28(4)**, 656–715.

[18] D.R. Stinson (2005), *Cryptography: Theory and Practice*, 3rd ed. Chapman & Hall / CRC Press, Boca Raton, Florida.

[19] L. Tierney (1994), Markov chains for exploring posterior distributions (with discussion). Ann. Stat. **22**, 1701–1762.