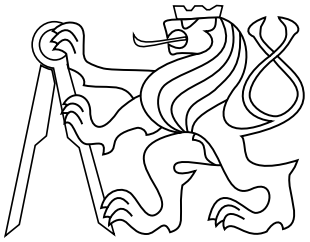
Bachelor's Thesis

MCMC Decryption

Tomáš Kocmánek



May 2013

Ing. Tomáš Kroupa, Ph.D.

Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Cybernetics

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Tomáš K o c m á n e k

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: MCMC Decryption

Guidelines:

1. The student will get familiar with the statistical method of cipher breaking that is based on the Markov chain-based sampling from a multidimensional distribution [2]. This MCMC method (Metropolis algorithm) [1] enables us to crack most of the usual text cipher (substitution, transposition).
2. The goal is to study further types of text ciphers (RSA or DES are thus excluded) and to develop a program for the cipher identification and the cipher attack.

Bibliography/Sources:

[1] Gilks W.R., Richardson S. and Spiegelhalter D.J.: Markov Chain Monte Carlo in Practice. Chapman & Hall/CRC, 1996.
[2] Diaconis, Persi: The Markov chain Monte Carlo revolution. Bull. Amer. Math. Soc. (2009)

Bachelor Project Supervisor: Ing. Tomáš Kroupa, Ph.D.

Valid until: the end of the winter semester of academic year 2013/2014

L.S.

prof. Ing. Vladimír Mařík, DrSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 10, 2013

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Tomáš K o c m á n e k

Studijní program: Otevřená informatika (bakalářský)

Obor: Informatika a počítačové vědy

Název tématu: Dešifrování pomocí MCMC

Pokyny pro vypracování:

1. Seznamte se se statistickou metodou luštění šifer popsanou v [2], která je založena na simulaci náhodného výběru z multidimenzionálního rozdělení pomocí markovského řetězce. Uvedená MCMC metoda (Metropolisův algoritmus) [1] umožňuje prolomit běžné textové šifry (substituční, transpoziční).
2. Nastudujte další druhy textových šifer (tedy nikoli binární šifry, jako je RSA nebo DES) a vytvořte program, který by druh šifry dokázal identifikovat a pokusil se ji prolomit užitím statistických metod.

Seznam odborné literatury:

[1] Gilks W.R., Richardson S. and Spiegelhalter D.J.: Markov Chain Monte Carlo in Practice. Chapman & Hall/CRC, 1996.
[2] Diaconis, Persi: The Markov chain Monte Carlo revolution. Bull. Amer. Math. Soc. (2009)

Vedoucí bakalářské práce: Ing. Tomáš Kroupa, Ph.D.

Platnost zadání: do konce zimního semestru 2013/2014

L.S.

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 10. 1. 2013

Acknowledgement

I would like to take this opportunity to thank my advisor Tomáš Kroupa for his support and professional guidance. Also I would like to thank my family for their support.

Declaration

I declare that I worked out the presented thesis independently and I quoted all the used sources of information in accord with Methodical instructions about ethical principles for writing academic thesis.

Prague, May 20, 2013

Abstract

Cílem této práce je prozkoumat možnosti využití metod Monte Carlo na markovských řetězcích k útoku na různé klasické šifry, jako substituční či transpoziční šifry. Vytvoříme program na lámání šifer, který může být použit při zájmovém luštění nebo na šifrovacích nočních hrách. Probereme možnosti, jak rozpoznat šifru, kterou byla zpráva zašifrována. Pomocí námi navržených metod dokážeme lámat i velmi krátké zprávy s délkou okolo 150 znaků.

Klíčová slova

Textová šifra, Monte Carlo, Markovský řetězec, dešifrování

Abstract

The goal of this thesis is to investigate the use of Markov Chain Monte Carlo (MCMC) methods to attack different classical ciphers, like substitution or transposition ciphers. We create a program for breaking specific ciphers, which can be used by hobby cipher breakers. We discuss the ways of cipher identification. We design and implement methods, which can break short encrypted messages with length around 150 letters.

Keywords

Text cipher, Monte Carlo, Markov chain, decryption

Contents

1 Introduction **1**

2 Cryptography **2**
2.1 Background on cryptography . . . 2
2.2 Cipher mechanisms . . . 3
2.3 Background on cryptanalysis . . . 4
2.3.1 Frequency analysis . . . 4

3 Background on MCMC **5**
3.1 Markov chain . . . 5
3.2 Metropolis algorithm . . . 5
3.3 Selecting new key . . . 6
3.3.1 Substitution, keyword and morbit cipher key change . . . 6
3.3.2 Pollux and Vigenère based ciphers key change . . . 7
3.3.3 Transposition ciphers key change . . . 7
3.3.4 Foursquare key change . . . 8
3.3.5 Playfair key change . . . 8

4 Identification of a cipher **10**
4.1 Ciphertext examination . . . 10
4.2 The use of unigram distribution . . . 11
4.2.1 Transposition ciphers recognition . . . 11
4.2.2 Substitution ciphers recognition . . . 12

5 Adjusting the breaking performance **14**
5.1 Problems with the Morse code . . . 14
5.2 Conditional probability versus sum of n-grams . . . 15
5.3 Which n-grams are the best . . . 16
5.3.1 Convex combination of n-grams . . . 17
5.4 Scaling parameter . . . 17
5.5 Number of iterations . . . 18
5.6 Further improvements . . . 19
5.7 Breaking foursquare cipher . . . 20
5.8 Breaking Playfair cipher . . . 21

6 Results **22**
6.1 Time complexity . . . 23

7 Summary **24**

Appendices

A Ciphers **25**
A.1 Caesar cipher . . . 25
A.2 Substitution cipher . . . 25
A.3 Keyword cipher . . . 26
A.4 Vigenère cipher . . . 26
A.5 Beaufort cipher . . . 27

A.6 Gronsfeld cipher 27
A.7 Autokey cipher 28
A.8 Playfair cipher 28
A.9 Foursquare cipher 29
A.10 Pollux cipher 30
A.11 Morbit cipher 31
A.12 Simple transposition cipher 31
A.13 Columnar transposition cipher 31

B Tables **33**

Bibliography **35**

1 Introduction

Cryptography is the study of algorithms that encrypts and decrypts messages sent between two parties. Markov chain Monte Carlo (MCMC) algorithms are popular methods of sampling from complicated probability distributions. Traditionally these two subjects have been quite distinct. Recently in [1] it was proposed how could these two fields be connected together. Later work [2] uses MCMC algorithms in order to break simple substitution and transposition ciphers.

In this work we consider only ciphers that can be encrypted with pen and paper. These ciphers are no longer in the use by any party to encrypt any secrets, because they are breakable. Due to this, many hobby breakers solves these ciphers and organize challenges in solving cipher puzzles.

The goal of this work is:

- Significantly extend the use of the MCMC on breaking various ciphers.
- Implementing methods and ciphers, and creating a program that could be used by hobby breakers.
- Improving the program to be able to break the shortest messages possible.
- Identifying cipher based only on the gibberish of encrypted message.

In the work [2], they estimated the optimal length of ciphertext to be 2000 letters, at the end of this work, we shows that it can be reduced to much shorter messages.

In the following two chapters we introduce the field of cryptography, the methods for breaking ciphers and MCMC algorithms, also the ways how it can be used on breaking ciphers. In the chapter 4 we are discussing ways of distinguishing ciphers. And in the chapter five, we try to minimize length needed for breaking ciphers. We experiment with different settings and test different approaches.

2 Cryptography

In this chapter we introduce notions of cryptography and cryptanalysis. We show how a cipher works and how it can be decrypted without knowing the key. Also we mention modern ciphers such as a DES or RSA.

2.1 Background on cryptography

During the course of history there were always secrets that needed to be hidden and parties, who were trying to get those secrets. The first cipher was used about 1900 B.C. (according to [3]); it was a symbolic substitution of hieroglyphs. Old ciphers used simple mechanisms, which could be easily broken. It was safe to use them, as long as nobody knows, what mechanism were used. For example, the keyspace of the Caesar's cipher is only 25 different keys. Later the mechanism started to be more complex, especially for the military purposes, in order to make breaking more difficult. Some ciphers we are going to break in this work were used even in World War II during the tactical operations.

Another approach besides encrypting messages is *steganography*, it is the art of writing hidden messages in such a way that no one, besides the sender and the intended recipient, suspects the existence of the message. For example, the use of invisible ink, messages written on the back of a stamp or writing the message on the shaved head of a slave and then wait until the hair grow back, before sending the message. The advantage of the steganography over the cryptography is that messages do not attract any attention. The messages encrypted with the classic ciphers only arouse suspicion and some parties try to break them.

Ciphers can be divided into two groups: *classical ciphers* and *modern ciphers*. Classical ciphers, such as the substitution and the transposition ciphers, perform encryption at the byte level, in other words changing one letter by another, or mixing their position. They are also called "pen-and-paper" ciphers, since only a paper and a pen is needed to apply them.

Since these ciphers are not considered secure, there is many hobby breakers, who try to solve the ciphers and takes the challenge of breaking them. Usually someone encrypt a message and publish only its encryption, without saying which algorithm was used. Participants then try to be the first one, who can break the code. There also exists many outdoor games based on breaking different kinds of codes. Participants usually get a code and if they break it they find out a location with a new code. It is like a race, where you try to be the first in the finish, which position is unknown. There is a lot of these games, for example the hardest game in the Czech Republic is TMOU [1]: 250 teams are on the start and after 20 hours only few of them find the way to the finish.

[1] http://tmou.cz/

Figure 1 Example of a cipher from night-long game TMOU

Another group of cipher are the modern ciphers, such as a DES or RSA. They operate at a bit level and use many different approaches during the encryption. For example, DES consists of substitution parts, transposition parts and many others; they change the text beyond recognition. They are much more complicated and secure than the classical ciphers. And they cannot be broken as easily as classical ones, but the goal of this work isn't to break any of them.

2.2 Cipher mechanisms

Ciphers are used for the secret communication between two parties without revealing the secret to any third party. In cryptography some information (called *plaintext*) is transformed into the unintelligible gibberish (called *ciphertext*) with the information known only by communicating parties (called *key*). The process of transformation is called *encryption*. *Decryption* is the converse. The transformation performing encryption and decryption is referred to as a *cipher*.

A cipher is a method of concealing a plaintext, where letters are substituted or transposed for other letters, pairs of letters or symbols (e.g. hieroglyphs, numbers, emoticons, etc.). In general, cipher operates on an alphabet of letters, usually English alphabet with 26 letters. There are countless number of different algorithms of encrypting the message. However only some of them are well known and were used by some important parties to hide their secrets.

In this thesis we work with 13 different ciphers. There were selected well-known ciphers representing different approaches to the encrypting (e.g. substitution, transposition, Vigenère, Polybius square, Morse code etc.). The complete list of ciphers is in the Appendix A, where you can find their brief description.

2.3 Background on cryptanalysis

Cryptanalysis is the science of analyzing crypto-systems in order to study the hidden aspects of them. Cryptanalysis is used to breach encrypted systems and gain an access to hidden information even if the key is unknown. The methods changed in course of history, but the goal has been always the same.

The first known recorded explanation of cryptanalysis was given by 9th-century Arabian polymath, Al-Kindi in A Manuscript on Deciphering Cryptographic Messages. He describe method called *frequency analysis*, which was for a long time the best approach to breaking ciphers.

2.3.1 Frequency analysis

Frequency analysis is a study of frequencies of letters or combinations of letters in the ciphertext. In a particular language (e.g. English) some letters occur more likely than the others. For example, in English "E" is the most used single letter, while "Z" is the least used. Analogously, we can study pairs of letters, triplets of letters etc. They are called *n-grams*, e.g. *unigrams, bigrams, trigrams*, etc. The easiest way to break classical ciphers is to compare the unigrams of the ciphertext and the unigrams of some long text (called *reference text*). Result of this type of attack on the substitution cipher is shown in Table 2, where we can see that lot of the letters aren't correctly decrypted. This method isn't very sophisticated and doesn't correctly decode the whole text. To get better results we have to use the pair frequencies and more complex algorithms.

Plaintext	`THE PROJECT GUTENBERG EBOOK OF OLIVER TWIST`
Decrypted text	`THE PSOJEWT FUTEIBESF EBOOK OG OLNVES TCNRT`

Figure 2 The result of frequency analysis attack on substitution cipher

3 Background on MCMC

Markov chain Monte Carlo (MCMC) methodology provides enormous space for possible statistical modeling. Monte Carlo methods draw samples from the required distribution and sample the expectation distribution. MCMC draws the samples from Markov chain, which is usually easily constructed with desired properties. And sampling it for a long time until we get expected value. MCMC algorithms are generally used for sampling from the high-dimensional distributions.

3.1 Markov chain

Markov chain (MC) is a mathematical system that describes the transition from one state to another, between a finite or a countable number of states. Each state depends only on the previous one and not on the sequence of states that preceded it. This is called the *Markov property*. A Markov chain is usually represented by a transitional matrix, where each of its entries $m_{ij} \in S$ is a probability that state i precedes state j. Suppose that the Markov chain is represented with the matrix $[m_{ij}]$. Then the vector π is called a *stationary distribution* if its entries π_j are non-negative and it satisfies:

$$\pi_j = \sum_{i \in S} \pi_i m_{ij}$$

You can find the formal definition and further information about Markov chains in [4].

The important part is that we will be using high-order matrices in this work. The Markov chain described above basically shows bigram distribution. When we are using trigrams, the we are considering Markov chain, where each state depends on two previous, equally tetragrams (each state depends on three previous states) or pentagrams (each state depends on four previous states).

3.2 Metropolis algorithm

The Metropolis algorithm can draw samples from any distribution P_X. The main idea of the algorithm is to generate a series of samples from a MC associated with P_X, here any sample depends just on its predecessor. After a sufficient amount of time the distribution of samples matches P_X distribution. This is guaranteed by the Fundamental Theorem for MC [5, Chapter 11.4].

This algorithm uses Markov chain, which can be estimated, when we count conditional probabilities of all pairs of letters in reference text. In this way, we get a stochastic matrix $M = [m_{ij}]$ of transition probabilities, where m_{ij} is a conditional probability that i precedes j.

The decryption of cipher is a function that maps the ciphertext to the plaintext:

$$f : \textit{encrypted space} \rightarrow \textit{decoded space}$$

At first we define the *score function* of the decryption f:

$$\pi(f) = \prod_i M(f(s_i), f(s_{i+1}))$$

where s_i runs over consecutive letters in ciphertext. The higher the score is, the more correctly decrypted the scored text is. The score function can be basically any function, which returns the higher value the more likely the text appears to be written in English.

The Metropolis algorithm proceeds as follows [6]:

First generate random key and then repeat the next steps sufficiently many times:

1. Choose a new key that depends on the last one, for example, it swaps two letters in the key.
2. Calculate the acceptance ratio a_t, where t represent the number of iteration

$$a_t = \frac{\pi(f_{t+1})}{\pi(f_t)}$$

3. Sample u_t from the uniform distribution over the interval $< 0, 1 >$.
4. If $a_t \geq u_t$, then accept the proposed key, otherwise reject it.

The algorithm proceeds by randomly attempting to move in the key space to more correct state, sometimes accepting the moves and sometimes remaining in the state. The randomness allows accepting the less plausible keys and prevents from getting stuck in local maximum. Note that the acceptance ratio a indicates how probable is the new proposed key with respect to the current key.

It is also possible to establish *scaling parameter* p and change the equation of a_t to the following

$$a_t = \left(\frac{\pi(f_{t+1})}{\pi(f_t)}\right)^p$$

This modification changes a density of $\pi(.)$ and can help the algorithm to escape from the local maxima.

3.3 Selecting new key

We need to define a way of changing a key k. The change of a key should make the smallest change in the decryption as possible. Therefore the dependency of these two keys must be maximal. In other words, we are doing the smallest change of the key as possible.

Ciphers represent the key in different way, therefore we discus picking a new key for each group of ciphers individually.

3.3.1 Substitution, keyword and morbit cipher key change

The key of the substitution and the keyword cipher is a permuted English alphabet, where each letter appears in the key exactly once. The key of morbit cipher is the same, but it consists of permutation of numbers 1-9. The smallest change we can make is to

swap two letters in the key. First idea for swapping letters was to swap two letters with uniform probability, so each swap have probability:

$$\frac{1}{n^2}$$

But with this configuration there is probability

$$\frac{1}{n}$$

that the key won't change, because it swaps one letter with itself. For substitution cipher it is $\frac{1}{26}$, so every 26th iteration in average is without effect.

To prevent this, we forbid swapping one letter with itself. Then the new probability of swapping two letters is:

$$\frac{1}{n(n-1)}$$

The swapping can be understand like moving from one state in Markov chain (the current key) to the following state (a proposed key).

3.3.2 Pollux and Vigenère based ciphers key change

This category contains following ciphers: Autokey, Beaufort, Gronsfeld, Vigenère and pollux cipher. These ciphers use a key which can have different length (except of pollux cipher, which have fixed length) and there aren't any constrains on the maximum number of appearances of some letter. This means that even key "KKKKKK" is a valid one.

The smallest change of the key is to change one letter in the key. Replacing a letter with the same letter is now allowed.

Notice that with this strategy used on the pollux cipher, we can get key that isn't correct, because each key must contain each symbol at least once. But the total number of the incorrect keys is $3 \cdot 2^9 = 1536$, which is insignificant in the comparison with the whole keyspace. However, in this work we are breaking the pollux cipher with a brute force of trying all possible keys, because the keyspace is relatively small (55977 keys). And in comparison with how many iterations would MCMC algorithm had to take, in order to get correct solution, it is faster to evaluate all keys.

3.3.3 Transposition ciphers key change

Both simple and columnar transposition ciphers uses a string of unique letters as a key, which determines the order in which are letters in plaintext swapped. The smallest change of the key is to swap two letters, which is the substitution cipher change. Even though that this more or less worked, the algorithm usually got stuck in local maximum. For example if you would have decryption "ROJECTP", you would have to make 6 changes to get the correct decryption "PROJECT", but usually another change of the key changes the decryption too much and the proposed key is rejected.

As proposed in [2], it is better to use *slide moves* of the letters in the key. To change a key by slide moves, one would take a random letter from the key and put it in the random position. Thus other letters would slide one position right or left. In this way you

need only one change of the key to get correct decryption of the ciphertext "ROJECTP".

Second improvement is that sometimes you need to move more than one letter. For example if you have decryption "WAR PEACE AND", which is a correct solution in bigrams point of view, but it isn't the correct solution. However, if you allow *block slide moves* you can get the correct decryption with only one change of the key. Block slide moves works basically as a letter slide moves, but instead of one letter you are moving block of letters.

3.3.4 Foursquare key change

The foursquare cipher uses two keys, which are independent on each other. Both keys are used to create Polybius square, which is 5x5 table of alphabet permutation. Therefore the smallest change is to select one of the keys and swap two letters in it.

3.3.5 Playfair key change

The Playfair cipher was the toughest one from all ciphers used in this work. It uses combination of three rules and encrypt message by pair of letters, so it makes the breaking much harder. You can change the key in many ways and there is none of them, which would be sufficient and which could replace the others. I describe three different types of changing the key. For each of them I have counted, how much one modification of the key change the plaintext.

The first possible change of the key is swapping two rows or columns in Polybius square. There is different 10 ways of permutation columns (rows respectively) and I have counted that the one change of the key alter with 11.7 % of possible pairs of letters in message.

The second possible change is to swap two letters. There is 600 different pairs which can be swapped and each swap alter with 20.1 % of the pairs. Therefore one swap of the key change $\frac{1}{5}$ of letters in the decryption.

The last change is the transposition. It was proposed by Jan Stumpel. The main idea is to transpose the key around some axis, therefore the decryption which uses rule 3 doesn't change and only pairs of letter that are decrypted with the rules 1 and 2 are changed. There are 4 different axis around which you can transpose the key: main diagonal (NW-SE), side diagonal (SW-NE), horizontal (W-E), vertical (N-S), all swaps are shown in the Figure 3. The size of text that is altered by these swaps differs. The first swap alter 66.6 % of the text, the second 33.3 % and the last two only 16.6 % of the text.

ABCDE	AFLQV	ZYXWV	VWXYZ	VQLFA
FGHIK	BGMRW	UTSRQ	QRSTU	WRMGB
LMNOP	CHNSX	PONML	LMNOP	XSNHC
QRSTU	DIOTY	KIHGF	FGHIK	YTOID
VWXYZ	EKPUZ	EDCBA	ABCDE	ZUPKE

Figure 3 The second Polybius square shows the change of the first square around axis NW-SE, the third around axis SW-NE, the fourth around axis W-E and the last one around axis N-S

The other problem associated with this cipher was to find out the ratio between these swaps. It would be irrational to select these swaps uniformly. I have selected the ratio based on the number of different possible swaps for each rule and on their percentage of unchanged pairs of letters. The ratio is following: in 5.4 % cases choose rows or columns swap, in 93.8 % choose the letters swap and in 0.8 % choose the transposition.

In addition the last rule differs on the selection of axis. Therefore I have divided the ratio between them the in following way: 13 % main diagonal, 25 % side diagonal and 31 % for horizontal and vertical transpositions.

4 Identification of a cipher

In this chapter we are going to discuss ways how to distinguish the ciphers. We consider only the ciphers that are implemented in this work, but this approach can be extended to any other list of ciphers. However some ciphers are very similar and cannot be distinguished. In this situation we try to break the ciphertext with all possible ciphers and then return the decryption with the highest score.

Because some ciphers use the same or similar mechanism of encryption, they can be broken with other ciphers. In this work there are three with this property: Caesar, keyword and Gronsfeld cipher. The first two mentioned can be broken as a substitution cipher, the latter as a Vigenère cipher. Therefore we can exclude them from the analyzed list of ciphers.

4.1 Ciphertext examination

Only by the examination of the ciphertext you can tell, that some ciphers cannot be used to decipher the text. First feature is that if the ciphertext contains only numbers or letters. If it contains only numbers, then the cipher that was used to encrypt the message is either pollux or morbit cipher. Furthermore, if the ciphertext contains at least one zero, you can say that it was encrypted with pollux, because morbit doesn't encode zeros.

Another feature is that the ciphertext contains other symbols than letters. Some ciphers can reflect for example punctuation or spaces into the ciphertext and others cannot. Therefore, if there are any other symbols, you can exclude autokey, foursquare and Playfair ciphers, because these cannot encrypt anything than the alphabet. Furthermore, if the ciphertext contains only letters and contains at least one letter J, you can exclude foursquare and Playfair. They don't use the letter J.

The next significant feature is the length of the ciphertext. Foursquare and Playfair encrypt message by pairs of letters therefore they must have the ciphertext with the even length. On the other hand, transposition ciphers need to have length of the ciphertext divisible by length of a key. Unfortunately the length of a key isn't limited, but it is beyond our capabilities to try every length of the key. Therefore we try only every possible key with length shorter than 26 letters.

The last feature is a minor one and can exclude only Playfair cipher. This cipher encrypt message by pairs of letters and cannot encrypt any pair consisted of doubled letter. If the ciphertext contains double letters in one its pairs, it cannot be encrypted by Playfair.

4.2 The use of unigram distribution

Since some ciphers during encryption doesn't change the unigram distribution in the ciphertext, you can use this information to distinguish among them. You can divide ciphers into three categories, which can be later distinguished. The first category are the ciphers, which doesn't change the unigram distribution at all. Those are the transposition ciphers. They only change the position of letters in the message. The second category are the monoalphabetic substitution ciphers. These ciphers substitute each letter with another, although we don't know which letter correspond to which, the distribution is still intact. And the third category are the remaining ciphers, which change the unigram distribution completely and we cannot tell anything about them.

4.2.1 Transposition ciphers recognition

Our first goal is to distinguish the first category, the transposition ciphers, from the others. We are going to use the fact, that each letter in the ciphertext should have similar a priori probability as the same letter in the reference text. We find the difference between the ciphertext and reference text, and establish the threshold, which distinguish that two groups.

To find out the difference we sort the alphabet based on the a priori probability of letters in ciphertext (and the same for the reference text). Therefore the first letter is the one with highest probability (for English it should be letter E). So we have two arrays of letters c and r. To compute the *difference* we are using following function:

$$difference(c, r) = \sum_{i}^{26} |c(i) - r(i)|$$

where functions $c(i)$ and $r(i)$ return position of letter i in the array c (and $r(i)$ respectively). In other words, if the uniform distributions are similar, the difference will be low. With this function we can evaluate every ciphertext. Now we must establish the threshold for detection. To find out its value we have proposed a test. For each cipher 1000 messages with different content and different length (50-400 letters long) was encrypted, and we measured theirs *difference*. We have divided the results into two groups, the first correspond to the first category and in the second group are ciphers from second and third category [1]. The distribution of *difference* values fits the normal distribution. You can see the result in the Figure 4.

[1] Note that we doesn't consider pollux and morbit ciphers, because they use numbers instead of letters.

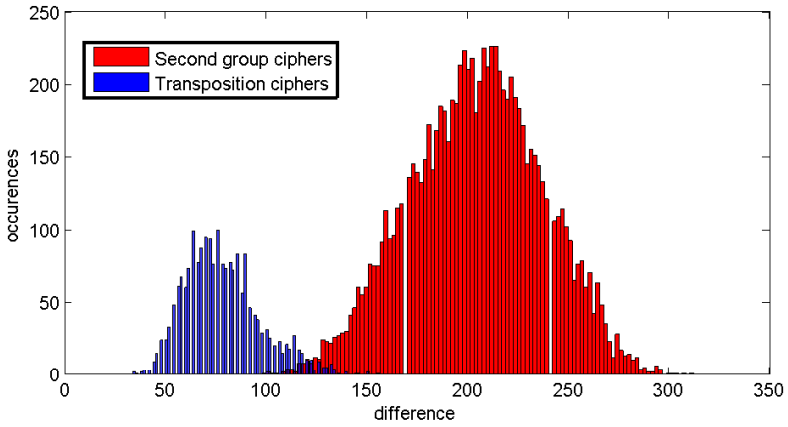


Figure 4 The results of the test, it shows that these two groups are easily distinguishable.

In the graph above, you can see that these two groups can be distinguished. In order to minimize an error from incorrect identification, we are going to use two threshold. If the *difference* is lower than 119, it is encrypted by the cipher from the first group e.g. transposition ciphers. And if it is higher than 126, it is encrypted by some cipher from the second group. Therefore we have zone between these two threshold, which doesn't say anything about the used cipher.

4.2.2 Substitution ciphers recognition

Now we can tell, if the ciphertext was encrypted by the transposition cipher or not. Further, we would like to identify the ciphers from the second category, e.g. the substitution ciphers. We know about this category of ciphers, that if we sort the array of a priori probabilities of letters in the ciphertext, lets call it $\vec{Pc}$, and the same for array of a priori probabilities of reference text, lets call it $\vec{Pr}$, they will be similar. In other words, the unigram distributions are alike, regardless the letter substitution. This is basically the frequency analysis attack on the substitution cipher. Although it isn't very successful attack, see the Figure 2 in the Chaper 2, it can be used to distinguish ciphers.

In order to distinguish different ciphertext, we define the following function:

$$contrast(c, r) = \sum_{i}^{26} |Pc_i - Pr_i|$$

where Pc_i indicates a priori probability of the i-th most probable letter in $\vec{Pc}$ (the array is sorted) and Pr_i analogously. Therefore if the unigram distributions of ciphertext and reference text are similar, the function *contrast* returns small values. And if the ciphertext is encrypted by ciphers, which disturbs unigram distribution, it returns high values. Furthermore we need to select the threshold for identifying ciphers. To find out this value we have performed the same experiment as previously. You can see the result in the following Figure:

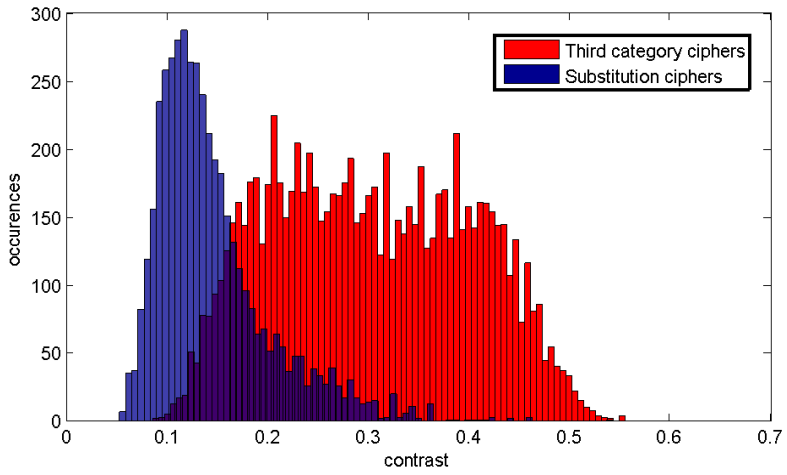


Figure 5 The results of the test. These two groups much more permeates each other

As you can see, the two groups of ciphers permeates each other much more than the transposition ciphers in previous section, so it cannot be so easily distinguished. To minimize the error of incorrect recognition, we have selected the two threshold in following way. If the *contrast* is lower than 0.1, it is encrypted by the substitution ciphers. And if it is higher than 0.3, it isn't encrypted by substitution ciphers. Unfortunately these two thresholds are far apart, therefore it usually doesn't tell anything about the used cipher, so we need to try to break all ciphers from both groups.

5 Adjusting the breaking performance

In this chapter we are discussing problems that have occurred during development of the program and how to handle them. And also how to adjust the breaking process to get the best performance. Our goal is to be able to break the shortest ciphertext possible. The time consumption isn't our primary goal, however breaking one ciphertext is done within few seconds for some ciphers dozens of seconds, which is a sufficient result.

As a reference text we are using books in text files with total size of 20MB. There are novels, history books, fact books or poems and also texts from different centuries. As a plain text we are using the book *Oliver Twist* by *Charles Dickens*. Note that all books are available for free in project Gutenberg [1].

5.1 Problems with the Morse code

The Morse code is a method of transforming text information written in 26-letter alphabet into a 3-letter alphabet (dot, dash and slash). Each letter is represented by a unique sequence of dots and dashes, and between each letter is one delimiter and between each word are two delimiters. It is also possible to encode numbers or others special symbols, which is not considered in this work.

Morse code has one inconvenient property, which makes breaking ciphers based on the Morse code (as a morbit or pollux ciphers) more difficult. The point is that you cannot decode some apparently wrong messages in order to get any gibberish. In other words, Morse code is really strict and therefore for example among every 5 symbols must be at least one delimiter or that some codewords don't have decoding at all (like `----`, `.-.-` or `..--`).

This property makes breaking these ciphers with MCMC more difficult. Because we cannot evaluate each decryption created by slight change of the key, so we cannot tell if we are making progress and accept proposed key or not. In order to avoid this problem, we come up with an easy and efficient solution of decoding invalid Morse codes, based on a greedy algorithm.

Under normal circumstances the decoding algorithm read symbols of the Morse code until it finds delimiter, then it compares the read symbols with the Morse decoding table and returns a single letter. If an error occurs, the algorithm ends. Our improvement works the same way until it reaches the error. The error can be either incorrect codeword or missing delimiter. In both cases the algorithm returns the decoding of the longest possible word and then continue from the last unused symbol. For example, if the message is `..--..-.` at first it tries to decode codeword `..--` (longer codewords are not possible), it finds out that this cannot be decoded, so it return the decoding of codeword `..-` which is `U`. After that it decode `-..-` as a `X` and at the end it decodes `E`.

[1] http://www.gutenberg.org/

Therefore the decoding of the `..--..-.` is `UXE`.

More methods were tested. One of them was to return the most likely letter, based on the frequency analysis, but this method was returning string mostly consisted of letters `E` and `T`, because both have high probability of occurrence and in Morse code are represented as one symbol codeword. Other approach was to read symbols until it finds delimiter and then equally divide symbols into the same size groups. For example, consider the previous Morse code `..--..-.`, would be at first divided into two groups of length 4, the first codeword couldn't be decoded, therefore it would split into 2 groups of length two. The decryption of the `..--..-.` would be `IMF`. However this method has proved to be less efficient than the chosen method.

5.2 Conditional probability versus sum of n-grams

As described earlier we count the score function as a product of the conditional probabilities of n-grams in the text. We determine the conditional probabilities as follows:

$$P(X|S) = \frac{\text{sum of appearances of string } SX \text{ in reference text}}{\text{sum of appearances of pairs starting with } S \text{ in reference text}}$$

where X is one letter and S is $(n-1)$-long string, that precedes X.

Chen and Rosenthal [2] proposed not to use conditional probability in the score function and instead use only the sum of appearances of n-grams in reference text, in other words only the numerator from the fraction shown above. They don't comment it at all and even doesn't mention the possibility of the use of conditional probability, which was proposed by [1]. This made me thinking which score function is better. The difference is only that in first case, we are normalizing it so the probabilities adds up to one.

In order to find out, which score function is better, we have proposed an experiment. We have encrypted random parts of text with various lengths and various ciphers. For this test we have selected 4 ciphers (substitution, transposition, Vigenère and autokey cipher) and for each one 100 messages of length between 50 and 400 letters were encrypted. After that we have tried to break each ciphertext twice, each time with both score functions separately (one that uses the conditional probability and the second that uses the sum of n-grams). Then we have count how many letters was decrypted incorrectly. The result is shown in the Table below.

Score func.	Letters correctly decryp.	Letters incorrectly decryp.	Correctly
Cond. prob.:	82610	9790	89.4 %
Sum of n-grams:	81940	10460	88.6 %

Table 1 Results of the test

As you can see both function are comparable and neither one is significantly better than the other.

5.3 Which n-grams are the best

During breaking the ciphers you can use various n-grams in the score function. But which n-grams are the best for breaking the ciphers? The bigrams are definitely better than unigrams, but what about trigrams or higher? The longer n-grams doesn't always mean the better solutions. From some point you have too long n-grams, that they couldn't break the cipher, because they won't give any reasonable score to the gibberish at the start of the Metropolis algorithm. Therefore the algorithm cannot start to climb up towards the solution.

To find out which n-gram is the best for each of the ciphers, we have proposed a test. We have encrypted 100 different messages with length uniformly distributed between 50-250 letters for each of the ciphers and then try to break them with the use of different n-grams. I have tried unigrams, bigrams, trigrams, tetragrams and pentagrams, the longer doesn't seem to make any improvement.

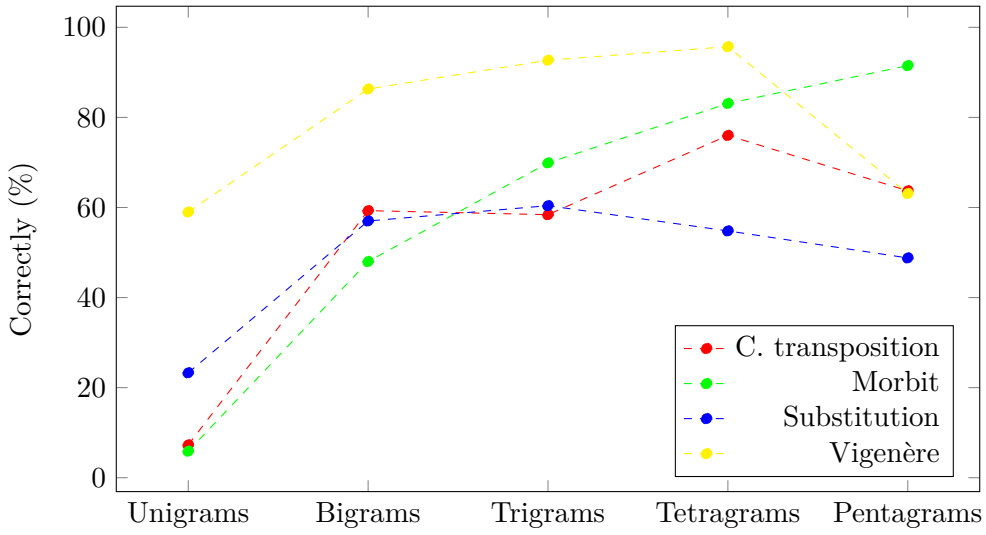


Figure 6 Graph comparing the use of the different n-grams on 4 selected ciphers

In the Figure 6 are shown results of this test on 4 selected ciphers. The graph shows how many percent of the text has been correctly broken from 100 different encrypted messages. The complete result of this test is in the Appendix B in the Table 19.

In the graph above, you can see that trigrams and tetragrams have usually better results than higher n-grams. As suggested earlier, it is because of that the algorithm cannot start climbing towards the solution, so any slight change of the key doesn't improve the score of the ciphertext.

If you examine Table 19 more closely, you notice, that the foursquare and Playfair ciphers had very poor results in this test, their best result was around 10% of correctness. Therefore we adjust them individually at the end of this chapter. They need longer ciphertexts and different approach to be broken.

5.3.1 Convex combination of n-grams

Can we use as a score function some mixture of different score functions, for example those which uses unigrams, bigrams or trigrams? We use basic method from Natural Language Processing the convex combination of n-grams. With this method you combine unigram, bigram and trigram models in the following way:

$$q(w_i|w_{i-1}, w_{i-2}) = \lambda_1 * P(w_i|w_{i-1}, w_{i-2}) + \lambda_2 * P(w_i|w_{i-1}) + \lambda_3 * P(w_i),$$

where $\lambda_1+\lambda_2+\lambda_3 = 1$, $\lambda_i > 0$ for all $i = 1, 2, 3$, and $P(.|.)$ is the conditional probability. To get the score of whole text, we use this function in the same way as Markov chains, therefore:

$$\pi(\mathbf{w}) = \prod_{i}^{textLength-2} q(w_i|w_{i-1}, w_{i-2}),$$

where w_i is the i-th letter in the ciphertext $\mathbf{w}$. We have defined score function, and we have to determine the λ_i. We have to maximize the following function:

$$L(\lambda_1, \lambda_2, \lambda_3) = \sum_{i}^{textLength-2} log(q(w_i|w_{i-1}, w_{i-2})),$$

where $\mathbf{w}$ is some validation text. The maximization of L isn't difficult task, we use an iterative method based on the EM algorithm, as described in lectures [7]. Some other methods of smoothing Language models, can be found in [8].

To find out if this method is better than simple n-grams, we run the same test as previous. The results can be seen in last column of Table 19 in the Appendix B. If we examine the results we can see, that this method is much better for the most of the ciphers. Therefore we set up our program to use the function that had the best performance in this test.

5.4 Scaling parameter

The next experiment is with choices of the scaling parameter. This parameter is very important in the Metropolis algorithm. Larger scaling parameters give lower acceptance rates, but if it is too low it converge towards the stationary distribution very slowly. On the other hand smaller scaling parameters give higher acceptance rates, but the changes are too often that algorithm won't always find the stationary distribution. To find the best scaling parameter I have made an experiment. As before I have encrypted 300 messages with the length between 20 and 180 letters, and tried to break them with different scaling parameters.

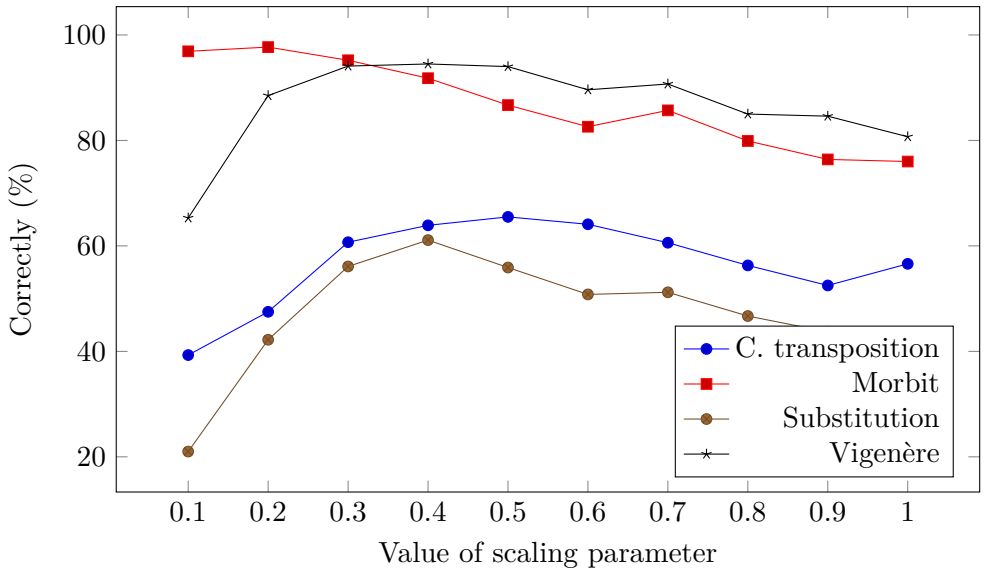


Figure 7 Results of the test of choosing the scaling parameter

Figure 7 shows results for selected ciphers (complete results for all ciphers can be found in the appendix B in the Table 20 [2]). As we can see, there is a best scaling parameter for each cipher usually between 0.2 and 0.5, therefore we choose these values as our scaling parameters (for each cipher its own).

5.5 Number of iterations

We could let the Metropolis algorithm run over a few millions iterations, but after some number of iterations it gets stuck in the maximum (either local or global) and the rest of the iterations would be useless. So how many iterations is needed to get the best performance?

We run a test, where for each cipher 300 messages with the lengths between 20 and 180 letters is encrypted and then we try to break them with various number of iterations. In the Figure below you can see the results of some selected ciphers. The complete results can be seen in the Appendix B in the Table 21.

[2]Note that we are not testing Caesar and Pollux ciphers, because we are breaking them with brute force, i.e. trying every possible key.

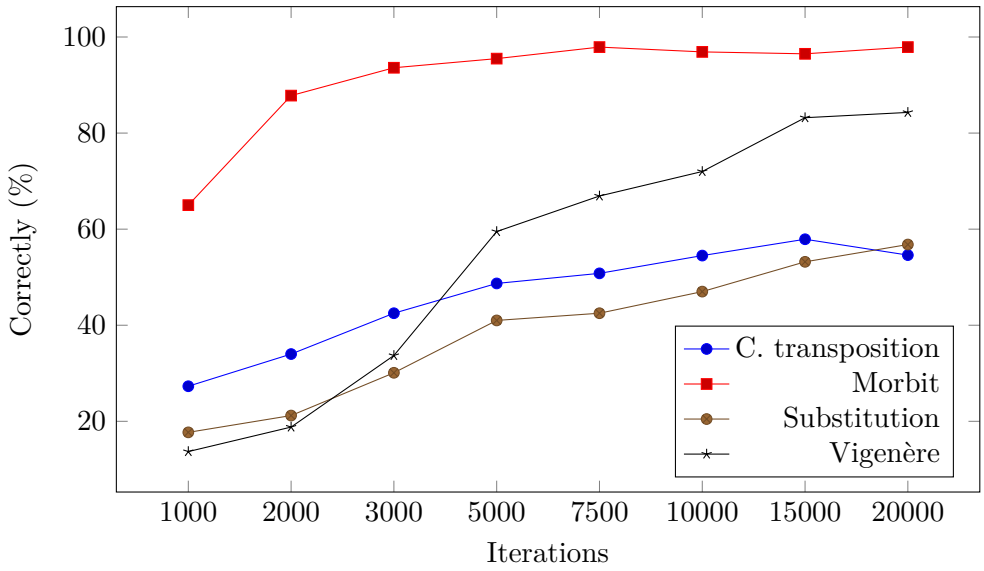


Figure 8 The results of the test showing how much of iterations break most of the ciphertext.

Figure 8 shows that by increasing number of iterations, we improve the accuracy of breaking. But the accuracy is changing very slowly after 5000 or 10 000 iterations. With this knowledge we adjust the program to run the Metropolis algorithm for 10 000 iterations (in case of some ciphers for 5000 iterations). We could make the number of iteration for example 50 000 and get better results, but we will instead improve Metropolis algorithm, to adaptively adjust number of iterations, when it is needed.

5.6 Further improvements

In some cases the algorithm gets into the local non-global maximum and cannot get out of it, because each change of the key make the score function too low to be accepted. We solve this problem by making the Metropolis algorithm to run more than once. To test this we encrypt 300 messages of length 20-120 letters and try to break them with modified Metropolis algorithm. As shown in the Figure below, if the algorithm runs 3 or 5 times we get much better results. In the Figure 9 are results of this test for some selected ciphers (complete results can be found in the Appendix B in the Table 22).

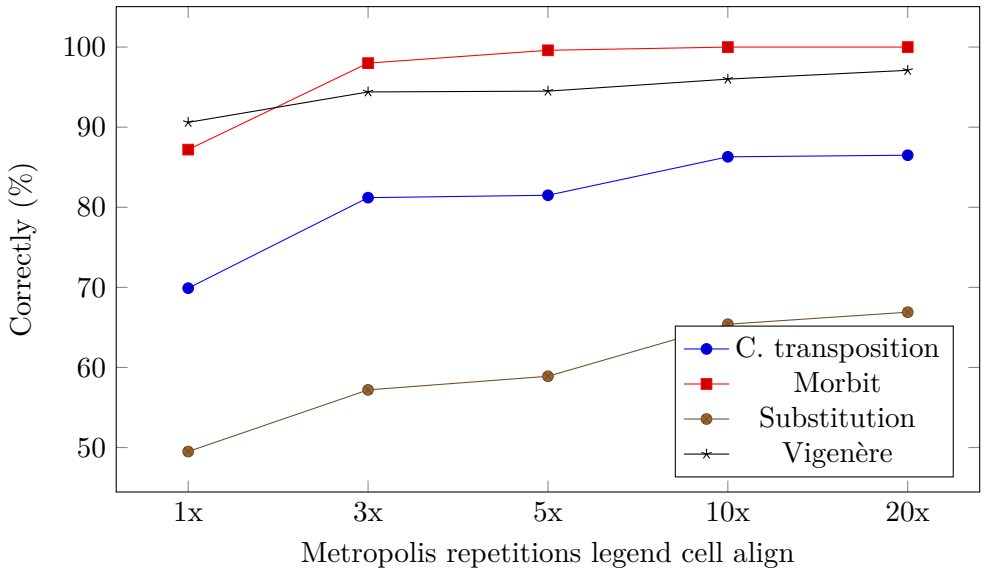


Figure 9 Results of the test of repeatedly running the Metropolis algorithm

Next problem was that sometimes the algorithm was cut off after the fixed number of iterations, even when he was near the solution, and if he would have a few more iterations, he could break it. This problem could be solved by either increasing the number of iteration, which would also increase the time of breaking the cipher or to adaptively change the number of iterations.

We select the second method. Where the algorithm run for fixed number of iterations and after that as long as the changes occurs. So, if in the last X iterations the change of key doesn't occur, then stop the algorithm. We choose the X to be a quarter of the fixed number of operations for the specific cipher. Then I have tried how much better it is in contrast to running Metropolis algorithm once with fixed number of iterations. The results are in the Table 2 (complete results can be found in the Appendix B in last column of the Table 22). For some ciphers it is improvement and for others it doesn't change the solution.

	fixed iterations	Extended iterations
Columnar transposition	69.9	74.6
Morbit	87.2	86.9
Substitution	49.5	50.8
Vigenère	90.6	91.1

Table 2 Results of the test showing how many percent of the text can be broken

5.7 Breaking foursquare cipher

As you could notice in previous experiments, foursquare had bad results on the text of length around 500. This can be caused because of its use of two keys. Thus the keyspace is squared, compared to ciphers that uses only one key. As well that the cipher encrypt the message by pairs of letters, which makes the number of substitutions much higher. Therefore to get better performance you need the longer ciphertext.

To adjust breaker of this cipher we repeat all previous tests with a text of length 5000 letters. It has been experimenting with the length and 5000 letters was sufficient for getting useful results.

The first test was which n-grams are the best for the foursquare cipher. After running 100 different ciphers it turned out, that with bigger n, the better results were made, until the tetragrams, which correctly broken 79.4 % of texts. After tetragrams any longer n-grams made worse results. This outcome is not surprising, because of that the cipher encrypt message by pairs of letters, then you need to used the n-grams, which take two adjacent pairs into consideration.

The second test was to established number of iterations that are needed. This test resulted that 10 000 iterations are sufficient and with more iterations the results doesn't improve much.

The last test was to find out the best scaling parameter. After running this test, it turned out that with the scaling parameter of 0.1 the algorithm had best performance with score 86 % of correctly decrypted messages.

The last adjustment was to run Metropolis 15 times, to avoid cases, when the algorithm gets stuck in local maxima. This adjustment had a huge impact on the score and the length of needed ciphertext rapidly dropped.

5.8 Breaking Playfair cipher

Same as the foursquare, the Playfair cipher wasn't easy to break. Unfortunately, we couldn't solve this issue the same way as in the foursquare cipher, therefore increasing length of the ciphertext, because breaking the Playfair cipher wasn't successful with the original Metropolis algorithm. After some experimenting with this cipher we come up with modification of Metropolis algorithm, which can break it. The modification works as follows:

Instead of letting the algorithm to climb towards the solution, sometimes reject the proposed key, sometimes accept even a worse key, we run the algorithm for many times, but only for restricted number of iterations. Therefore with every repetition of the algorithm, it resets the starting position and starts in different distance from the solution and tries to reach the solution. This way the algorithm generates many attempts, where usually none of them is the correct solution. After that we choose the best attempt among them and run it through original Metropolis algorithm.

We repeat the same experiments to adjust this improvement. Every times we have encrypted 100 messages and tried to break them with various settings. It turns out that this algorithm have the best results, when are used tetragrams in the score function. This is not a surprise, because the cipher encrypt the message by pairs of letters, the score function have to take it into account. The number of iterations, which restricts expansion of attempts, was established at 800. If one use more iterations, the algorithm would run much longer and the results wouldn't be much better. And the number of resets of Metropolis algorithm was established to 125.

6 Results

In this chapter we are going to estimate the shortest length of a ciphertext needed to successfully break presented ciphers. We have designed a test, which determine the minimal length for each cipher. During the test, 100 messages with the same length are encrypted and tested. The length of ciphertexts was gradually increased. As the shortest length of ciphertext correctly encrypted is the one with more than 90 % of successfully decoded letters. We are assuming that the rest of letters can be guessed right afterwards.

All tests were carried on personal computer with the following cPC configuration: Intel Core i5-2410M 2.3 GHz, with 4 GB memory RAM and operation system Microsoft Windows 7 64x.

Length	Auto.	Beau.	Caes.	C.T.	Gron.	Key.	Morb.	Pol.	Subs.	S.T.	Vige.
10	5.9	5.6	**100**	9.7	23	7.9	34.3	89.4	9.3	8.5	8.1
20	9.7	6.1	100	12.1	26	11.7	**96.9**	**96.1**	13	10.9	8.5
30	18.3	18.1	100	26.3	40	14	98.3	99	15	17.8	16.4
40	37.7	41.3	100	38	47	22.4	99.1	99.8	21	27	31.5
50	51.2	43.4	100	47.1	69	30.8	100	99.6	33	44.1	48.3
60	56.7	63.2	100	49.8	77	43.4	100	99.8	41	49.2	63.4
70	78.1	69.1	100	57.3	87	54.2	100	100	58	50.7	70
80	80.7	84.4	100	60.8	89	68.5	100	100	68	61.5	82.8
90	78.8	81.9	100	58.2	**94**	77.9	100	100	79	76	87.6
100	85.1	88.2	100	70.9	97	84.3	100	100	84	88	88.2
110	**91.8**	**95.4**	100	82	98	**91.8**	100	100	89	82.3	**97.7**
120	88.9	94	100	79.4	99	93.9	100	100	**94**	88.4	95.7
130	94.4	98	100	85.6	100	95.8	99.1	100	96	**94.4**	96.8
140	94.3	98.2	100	86	100	96.8	100	100	97	96.5	96.4
150	94.5	100	100	**93.2**	100	96.3	100	100	97	95	97.1
160	92.5	98	100	93	100	98.3	100	100	97	93.3	99
170	95.3	100	100	96.4	100	99.2	100	100	98	99.1	99

Table 3 Results of the test showing, how many percent of the text can be decrypted with fixed ciphertext length

The results of breaking ciphers with MCMC are remarkable, for the most ciphers it is sufficient to have a text with the length shorter than 150 letters. And this is the average length of ciphers in various cipher challenges, therefore it can be easily used. If you compare our results of breaking substitution and transposition cipher with [2], where they estimated optimal length of a ciphertext to be 2000 letters, we can see that our solution can break more than ten times shorter messages with the same accuracy.

Length	50	150	250	350	450	550	650	750	850
Foursquare	7.6%	10.4	16.8	63.6	**96.2**	98	99.5	99.8	99.8

Table 4 How many percent of ciphertext can be broken with foursquare cipher and fixed length

The results for the foursquare cipher are in Table 4. As you can see the foursquare cipher needs to have longer messages than others ciphers (except of Playfair cipher). This is apparently because this cipher uses much bigger keyspace than others ciphers. Even though this result is good enough.

Length	200	400	600	800	1000	1200	1400	1600
Playfair	8.25	27.41	81.73	88.82	**90.81**	92.94	94.35	95.4

Table 5 How many percent of ciphertext can be broken with Playfair cipher and fixed length

The last cipher is the Playfair. It is the most complicated cipher among all the ciphers considered in this work. We cannot improve breaking performance for this cipher much more, therefore the shortest text that is much likely to be decrypted is around 1000 letters long. It is not as good as with others ciphers, but still it could be used for breaking longer ciphertexts. Another MCMC method, Simulated Annealing, which was used in [9] can break even shorter messages.

6.1 Time complexity

Breaking ciphers take some time, usually it is a several seconds, but it mainly depends on the length of a ciphertext. Only foursquare and playfair ciphers takes a longer time. The foursquare needs around 30 seconds and the Playfair takes few dozens second up to a few minutes. These results are for this purpose of breaking sufficient. The goal of this work wasn't to get the fastest breaker, although it could be optimized.

7 Summary

In this thesis, we successfully applied MCMC algorithms to break various ciphers. We have implemented even some ciphers that aren't well examined for breaking on the computer. The attacks are based on the frequency analysis of the ciphertext compared with a reference text. To get the best results one must know what could be the content of the encrypted message and select the proper reference text. We have experimented with such issues as choosing the optimal n-grams, number of iterations, scaling parameter, number of repetitions of the whole algorithm, different changes of keys, etc. Our goal was to minimize the length of text needed to break the ciphers. This work indicates the potential of MCMC algorithms in cryptanalysis.

We have discussed basics of recognition ciphers based on theirs ciphertexts, by examination of symbols of their unigram distribution. This area isn't much explored and there are many objectives that would be worth of researching. Besides distinguishing ciphers, one could find out, if there are ciphertext, which can be broken with different ciphers and keys to get valid messages. Or finding out how much are some ciphers different and if they can be even distinguishable.

Further work should involve implementing more ciphers and extending the possibilities of the program. Also I would like to move the implementation on smartphone devices, so that it is usable during outdoor ciphering games.

Appendix A

Ciphers

We introduce some of the ciphers, how they work and some basic information about them. There are many sources about "pen and paper" ciphers, mostly written by the hobby breakers. The American Cryptogram Association occupy itself with the ciphers, codes and cryptograms, that can be solved by hand and they publish the regular journal called The Cryptogram. They also published the largest collection of manual ciphers and theirs mechanisms[1].

We have selected representatives out of many different approaches of encryption and also we tried to select the most used and the well-known ciphers, in order to make this work less specialized.

A.1 Caesar cipher

The Caesar cipher is simple cipher, which is often used inside others more difficult ciphers. It is named after Julius Caesar, who used it in his private correspondence.

During the encryption each letter in the plaintext is shifted by fixed number of places in the alphabet.

Because shifting plaintext by zero (or by letter A) doesn't change anything, therefore the size of the keyspace is only 25 different keys. Which means that this cipher can be easily broken with pen and paper, if you try every possible combination.

Key:	DDDDDD DDDDDD
pt:	caesar cipher
CT:	FDHVDU FLSKHU

Table 6 Example of encryption using Caesar cipher

A.2 Substitution cipher

In cryptography the substitution cipher is a method of changing letters or groups of letters of plaintext by theirs substitution. It can be letters from alphabet, numbers or even strange symbols. Even that the substitution ciphers are easily broken, they are used inside bit-oriented block ciphers (e.g., DES, or AES). Block ciphers often include smaller substitution tables called S-boxes.

[1] http://cryptogram.org/

In this work we talk about *simple substitution cipher*, where each letter is substituted by another one. To represent the key, we use mixed alphabet of 26 letters. Where the A is substituted by the first letter of key, the B is substituted by the second letter of the key and so on.

Because the key have length 26 and there each letter must be exactly once. Therefore the size of the keyspace is equal to *26!*.

Key:	RAVBHXGPTMUIQJFELNDSKYOWCZ
pt:	mixed alphabet
CT:	QTWHB RIEPRAHS

Table 7 Example of encryption using simple substitution

A.3 Keyword cipher

The keyword cipher is a simple substitution cipher, where instead of using key of 26 letters, you can use any word or even phrase. To create key, you remove all symbols except letters of alphabet and you remove all duplicates of letters that are already in the key. And finally you add all letters that aren't in the key at the end of it. This way you get shifted alphabet, which is then used during encryption by simple substitution cipher. For example, the keyword "EXTRAORDINARY" creates a key "EXTRAODINYBCFGHJKLMPQSUVWZ".

The size of the keyspace depends on the length of the key. We aren't considering repeated letters, because they don't change the key, therefore the size of the keyspace is equal to $\frac{26!}{(26-k)!}$, where k is length of the key.

A.4 Vigenère cipher

The Vigenère cipher is a method of encrypting plain text by using several different Caesar ciphers based on the letters of a keyword. This is well known cipher and in course of the history it earned the description *le chiffre indéchiffrable* (French for "the indecipherable cipher").

This Cipher was formulated by Leon Battista Alberti around 1467, but it is named after Blaise de Vigenère, who published his description of a similar, but stronger autokey cipher. Last time this cipher was used for the militarily purpose was in American Civil War around years 1860.

Gilbert Vernam tried to improve this cipher, but his Vernam-Vigenère cipher was still vulnerable to the cryptanalysis. His work, however, eventually led to the creation of one-time pad, a provably unbreakable cipher.

To encrypt with Vigenère cipher you can use a table of alphabets called *a tabula recta*. It is table with 26 rows and 26 columns. On each of the rows is an alphabet, which is shifted by one according to the previous one. Therefore it correspond to the 26 possible Caesar ciphers. The alphabet used at each point of encryption depends on

a repeating keyword.

The size of the keyspace depends on the length of the keyword k. And it equals to 26^k.

Key:	VIGENEREVIGENE REVIGENEREVI
pt:	polyalphabetic substitution
CT:	KWRCNPGLVJKXVG JYWAZMGYKMJV

Table 8 Example of encryption using Vigenère cipher

A.5 Beaufort cipher

The Beaufort cipher was created by Sir Francis Beaufort. It is polyalphabetic cipher similar to Vigenère cipher, but uses a slightly modified enciphering mechanism and tableau. Its most famous application was in a rotor-based cipher machine, the Hagelin M-209. Machine Hagalin M-209 was portable mechanical cipher machine used by US Navy during the World War II and later in Korean war.

To encrypt you are using a tabula recta. First find the plaintext character on the top row of the tableau, for example column P. Secondly, travel down through column P to the corresponding letter in the key, for example K. Finally, move directly left on that row to the left edge of the tableau, the encryption of letter P with the key letter K will be there, in this case letter V.

Due to the similarities between this cipher and the Vigenère cipher it is possible to apply transformation on a tabula recta and then encrypt messages as a Vigenère cipher.

Size of the keyspace is same as in Vigenère cipher, therefore it is equal to 26^k, where k is the length of the keyword.

Key:	BEAUFORTCIPHER BEAUFORTCIPH
pt:	polyalphabetic substitution
CT:	MQPWFDCMCHLOWP JKZCMGYZJABU

Table 9 Example of encryption using Beaufort cipher

A.6 Gronsfeld cipher

The cipher has been named after Johann Franz von Gronsfeld (1640-1719). He was the imperial field marshal in the Bavarian national uprising of 1705.

The Gronsfeld cipher is exactly the same as the Vigenère cipher, but the numbers are used instead of letters in the key. There is no other difference. The advantage of this cipher is that the key can be picked from a sequence, e.g. the Fibonacci series or from some pseudo-random sequence. You can also use as a key the decimal expansion of some irrational number, which makes breaking this cipher really difficult.

If we consider only the keys, that are represented as an integer, then the size of the keyspace is 10^k, where k is the length of the keyword.

A.7 Autokey cipher

The autokey cipher works as Vigenère cipher, but instead of repeating the keyword, plaintext is used to encrypt itself. To encrypt one would append the plaintext to keyword, so during the encryption process the first letters of plaintext are encrypted by keyword and then the rest is encrypted by plaintext itself. This makes frequency analysis really difficult.

The size of the keyspace is exactly 26^k.

Keyword:	AUTOKEY
Key	AUTO KE Y thisis asecret
pt:	this is a secret message
CT:	TBBG SW Y LLKJML MWWURKX

Table 10 Example of encryption using autokey cipher

A.8 Playfair cipher

The Playfair cipher was invented in 1854 by Charles Wheatstone, but it is named after Lord Playfair who promoted the use of the cipher. It was used for tactical purposes by British forces in the first World War and for the same purpose by the Australians and Germans during World War II. This was because the Playfair is reasonably fast to use, requires no special equipment and by the time the enemy cryptanalysts could break the message, the information would be useless.

The cipher use shuffled *Polybius square* and 3 simple rules. The Polybius square is 5 by 5 square, which contains all letters of alphabet except of one letter (usually I and J is considered as one letter). To create this square you need to know the keyword. The letters of the keyword are written into the square one letter by one into the cells of the square, skipping letters which are already in the square. After that, the empty cells of the square are filled with the rest of letters from the ordered alphabet.

P	L	A	Y	F
I	R	B	C	D
E	G	H	K	M
N	O	Q	S	T
U	V	W	X	Z

Table 11 Example of Polybius square with keyword PLAYFAIR

During the encryption, one would break the message into groups of 2 letter. If there is any of pairs containing one letter twice, then it is needed to insert any letter between them (usually letter X) to separate these pairs. If needed append one letter at the end to make the text length even. Then encrypt each pair using one of the following rules:

1. If both letters appear on the same row of the table, replace each of them by the letters, which are immediate right to them (wrap around to the left side of the table if necessary).

2. If both letters appear on the same column of the table, replace each of them by letters, which are immediate below them (wrap around to the top side of the table if necessary).

3. If the letters are not on the same row or column, replace the first letter with the letter which is on the same row as first letter and on same column as second letter. And the second letter replace with the letter which is on the same row as second letter and on the same column as the first letter.

The size of the keyspace is equal to 25!. Note that keys are consisted of only 25 letters.

Key:	PLAYFAIR
pt:	sy mX me tr ic ci ph er
CT:	XC KZ EG OD RD DR AE GI

Table 12 Example of encryption using Playfair cipher

A.9 Foursquare cipher

The foursquare cipher was invented by famous French cryptographer Felix Delastelle in 1902 as a variant of Playfair cipher. It uses 4 Polybius square, which are connected to each other and creating one bigger square. The second and the third Polybius square is mixed using 2 keywords as in Playfair cipher.

This cipher removes the flaw of Playfair cipher, where the word ABBA for example would be encoded by XY YX, the characters are reversed in both the plaintext and the ciphertext.

During the encryption process one split the text into pairs of letters. First letter of each plaintext pair is found in the first square, the second letter is found in the fourth square. These two cells are considered opposite corners of a rectangle. Cipher substitutes are found at the other corners of that rectangle, first in the square 2 and the second in the square 4.

The size of the keyspace depends on keylengths of both keywords and is equal to $\frac{25!}{(25-k)!} * \frac{25!}{(25-l)!}$, where *k, l* are lengths of the keywords. Note that keys are consisted of only 25 letters.

1					2				
A	B	C	D	E	E	X	A	M	P
F	G	H	I	K	L	B	C	D	F
L	M	N	O	P	G	H	I	K	N
Q	R	S	T	U	O	Q	R	S	T
V	W	X	Y	Z	U	V	W	Y	Z
K	E	Y	W	O	A	B	C	D	E
R	D	A	B	C	F	G	H	I	K
F	G	H	I	L	L	M	N	O	P
M	N	P	Q	S	Q	R	S	T	U
T	U	V	X	Z	V	W	X	Y	Z
3					4				

Table 13 Four squares created with keywords EXAMPLE and KEYWORD

Key:	EXAMPLE and KEYWORD
pt:	he ll ow or ld
CT:	FY GF HX HQ KK

Table 14 Example of encryption using foursquare cipher

A.10 Pollux cipher

The pollux cipher is based on the Morse alphabet, which itself is composed of dots, dashes and separators. The Pollux cipher encodes each of these three symbols into numbers. Where each symbol can be represented by more then one number, therefore any word in plaintext can be encrypted in many different ways.

The key is represented as a string of 10 symbols (dot, dash, slash). Where the first symbol of key is encoded to the number zero, second symbol into the number one and so on. This makes a list of numbers assigned to dots, dashes or slashes.

During the encryption process one should pick symbols from these lists at random.

The size of the keyspace is $3^{10} - 3 \cdot 2^{10} = 55977$ (the key must contain each symbol at least once). Because of this small size of the keyspace, the easiest method to break this cipher is to use the brute force attack, which takes for this cipher same time as the use of the MCMC method. During the brute force attack one tries every possible key and return the decryption with the best score.

Key:	-././-/..
Representation of dots:	2,4,8,9
Representation of dashes:	0,1,6
Representation of slashes:	3,5,7
pt:	pollux
Morse code:	.--./---/.-../.-../..-/-..-/
CT:	816856015464874689394671946 7

Table 15 Example of encryption using pollux cipher

A.11 Morbit cipher

As a pollux cipher the morbit cipher uses a Morse alphabet. The main difference is that it takes off the plaintext in units of 2 during the encryption and replaced it by the numbers 1-9. The key assigns to each of the 9 pairs of symbols (permutation of dots, dashes or slashes) one number. The key can be represented either as string of numbers or string of letter, where each letter is mapped onto a number based on its order in the alphabet.

The size of the keyspace is exactly 9!.

Key:	TIMETABLE
Num. key:	857391264
Encoding:	...---/// .-/.-/.-/
pt:	morbit
Morse code:	--/---/.-./-.../../-
CT:	9 6 9 2 3 6 8 7 8 6

Table 16 Example of encryption using morbit cipher

A.12 Simple transposition cipher

The transposition cipher is a method of encryption by which the positions held by units of plaintext are moved to the new positions, so that the ciphertext is a permutation of the plaintext.

The simple transposition cipher split the text into blocks with same length as a key word and use the same permutation of letters in each block according to the keyword. The letters in the keyword cannot repeat.

The size of the keyspace is equal to $\frac{26!}{(26-k)!}$, where k is length of the keyword.

Keyword:	DOG
Key:	132 132 132 132 132 132 132
pt:	tra nsp osi tio nci phe r..
CT:	TAR NPS OIS TOI NIC PEH R..

Table 17 Example of encryption using simple transposition cipher

A.13 Columnar transposition cipher

The columnar transposition cipher extends the simple transposition cipher, in the way that it changes positions of letters across the whole plaintext rather than across the block. During the process of encryption, one would write the whole text into the table with width same as the length of the key, usually the key is transform into the numbers and is written above the table. Then to create ciphertext one would write first letters from column under the smallest key letter (or the column marker with key number as

1), the letters from column marked with number 2 and so on.

The size of the keyspace is equal to $\frac{26!}{(26-k)!}$, where k is length of the keyword.

Keyword:	RADIO
pt:	columnar transposition
Key:	51234
	colum nartr anspo sitio
pt:	n....
CT:	OANI. LRST. UTPI. MROO. CNASN

Table 18 Example of encryption using columnar transposition cipher

Appendix B

Tables

Cipher	Unig.	Big.	Trig.	Tetrag.	Pentag.	Linear Inter.
Autokey	71.1	84	94.8	92.7	52.6	98.5
Beaufort	62.5	85.6	93.8	90.8	61.1	97.1
Caesar	100	100	100	100	100	100
Col. transpos.	7.3	59.3	58.4	76	63.7	76.6
Foursquare	10	10.4	8.2	7	6.3	9.1
Gronsfeld	77.8	94.5	96.8	97.7	91.1	98.8
Keyword	23.6	65.8	61.2	52.7	43.4	72.5
Morbit	5.9	48	69.9	83.1	91.5	77
Playfair	7.5	7	6.8	6	5.7	7.2
Pollux	6.4	91.7	99.7	100	100	100
Substitution	23.3	57	60.4	54.8	48.8	74.8
Simple transpos.	10.7	56.5	61.4	81.3	69.2	75.4
Vigenère	59	86.3	92.7	95.7	63.1	98.3

Table 19 Results of the test, which n-grams are best for specific ciphers. Shows how many percent of the text were correctly decrypted

	0.01	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	10
Autokey	22.6	66.4	88.2	93.4	94.8	92.8	90.7	84.7	83.6	79.6	79.4	56.1
Beaufort	24	67.8	88	92.6	93.8	94.4	89.4	88.9	85.5	79.9	78.7	65.4
Col.trans.	25.4	39.3	47.5	60.7	63.9	65.5	64.1	60.6	56.3	52.5	56.6	53
Gronsfeld	31.6	64.5	84.3	90.3	92.3	93.6	94.6	92.8	92	91.5	91	75.9
Keyword	19.4	19.5	39.5	55	54.5	53.2	52.1	46.1	48.3	45.6	41.3	34.7
Morbit	12.5	96.9	97.7	95.2	91.8	86.7	82.6	85.7	79.9	76.4	76	54.9
Substitution	18.3	21	42.2	56.1	61.1	55.9	50.8	51.2	46.7	43.7	43.1	31.8
S. trans.	28.7	38.5	50.2	57.5	67.1	59.3	60.2	59.8	63.5	61	59.9	51.1
Vigenère	26.3	65.3	88.5	94.1	94.5	94	89.6	90.7	85	84.6	80.7	64.1

Table 20 This table shows how many percent of the text can be broken with different scaling parameters

	500	1000	2500	5000	7500	10000	15000	20000
Autokey	12.1	19.8	34.9	49.5	65	70.9	82.1	86.6
Beaufort	13.7	18.7	36.4	53.8	66.7	72.1	79.8	79.6
Col. Trans.	27.3	34	42.5	48.7	50.8	54.5	57.9	54.6
Gronsfeld	29.6	40.9	61.7	79.5	80.9	84.8	88.1	88.6
Keyword	17.5	24.2	33.4	36.2	44.6	49	54.7	53.1
Morbit	65	87.8	93.6	95.5	97.9	96.9	96.5	97.9
Substitution	17.7	21.2	30.1	41	42.5	47	53.2	56.8
S. trans.	26.7	39.9	45.3	47.8	52.2	55.4	50.8	51.9
Vigenère	13.7	18.8	33.7	59.5	66.9	72	83.2	84.3

Table 21 The results of the test showing how many iterations break the most percentage of the ciphertext.

	1x	3x	5x	10x	20x	Extend iterations
Autokey	90.2	93.1	94.5	95.3	95.6	89.6
Beaufort	91.9	94.2	95.6	96.3	96.8	92.4
Columnar transposition	69.9	81.2	81.5	86.3	86.5	74.6
Gronsfeld	95.5	96.6	96.8	96.7	96.4	95.5
Keyword	50.5	57.4	60.8	65.7	67.7	51.5
Morbit	87.2	98	99.6	100	100	86.9
Substitution	49.5	57.2	58.9	65.4	66.9	50.8
Simple transposition	77.9	88.7	89.5	92.1	92.9	78.5
Vigenère	90.6	94.4	94.5	96	97.1	91.1

Table 22 Results of the test of improving the Metropolis algorithm showing how many percent of the ciphertext can be broken.

Bibliography

[1] P. Diaconis. "The Markov Chain Monte Carlo Revolution". In: *Bull. Amer. Math. Soc.* (Nov. 2008).

[2] J. S. Rosenthal J. Chen. "Decrypting Classical Cipher Text Using Markov Chain Monte Carlo". In: *Department of Statistics, University of Toronto* (May 2010).

[3] David Kahn. *The Codebreakers, the Story of Secret Writing*. 1973.

[4] J. R. Norris. *Markov chains*. Ed. by University of Cambridge. 1997.

[5] J. L. Snell C. M. Grinstead. *Introduction to Probability, 2nd edition*. Ed. by Amer. Math. Soc. 2003.

[6] D. J. Spiegelhalter W. R. Gilks S. Richardson. *Markov Chain Monte Carlo in Practice*. Ed. by Chapman & Hall/CRC. 1996.

[7] Michael Collins. *COMS W4705: Natural Language Processing*. Ed. by Columbia University. 2013. URL: `http://www.cs.columbia.edu/~cs4705/`.

[8] J. Goodman S. F. Chen. "An Empirical Study of Smoothing Techniques for Language Modeling". In: *Harvard University Cambridge, Massachusetts* (Aug. 1998).

[9] Michael J. Cowan. "Breaking Short Playfair Ciphers with the Simulated Annealing Algorithm". In: *Cryptologia, 32:1, 71-83* (2008).