An Automatic Cryptanalysis of Transposition Ciphers Using Compression

Noor R. Al-Kazaz(✉), Sean A. Irvine, and William J. Teahan

School of Computer Science, Bangor University,
Dean Street, Bangor, Gwynedd LL57 1UT, UK
{elp486,w.j.teahan}@bangor.ac.uk, sairvin@gmail.com
http://www.bangor.ac.uk/cs

Abstract. Automatically recognising valid decryptions as a result of ciphertext only cryptanalysis of simple ciphers is not an easy issue and still considered as a taxing problem. In this paper, we present a new universal compression-based approach to the automatic cryptanalysis of transposition ciphers. In particular, we show how a Prediction by Partial Matching (PPM) compression model, a scheme that performs well at many language modelling tasks, can be used to automatically recognise the valid decrypt with a 100 % success rate. We also show how it significantly outperforms another compression scheme, Gzip. In this paper, we propose a full mechanism for the automatic cryptanalysis of transposition ciphers which also automatically adds spaces to decrypted texts, again using a compression-based approach, in order to achieve readability.

Keywords: Cryptanalysis · Transposition ciphers · Plaintext recognition · Compression · PPM · Word segmentation

1 Introduction

Text compression is the method of deleting redundant information in some text in order to reduce space that is needed to store it, thereby minimising the time which is also needed to transmit this information without losing any information from the original text. Practically, there are two major ways of constructing text compression models: dictionary and statistical approaches [1]. Prediction by Partial Matching (or PPM) is a finite-context statistical based approach while Gzip is an example of a dictionary based approach (which uses the Lempel-Ziv algorithm [9]). PPM models perform well on English text compared to other models and they emulate human predictive ability [25].

There are variety of approaches and algorithms used for cryptanalysis. Using compression schemes as one way to tackle the plaintext recognition problem is still a relatively new approach with few publications. In essence, we investigate how to devise better solutions to the plaintext recognition problem by using transposition ciphers as a test bed. In this paper, we propose a novel compression-based approach for the automatic cryptanalysis of transposition ciphers with

S. Foresti and G. Persiano (Eds.): CANS 2016, LNCS 10052, pp. 36–52, 2016.
DOI: 10.1007/978-3-319-48965-0_3

no need for any human intervention. Furthermore, we propose further methods also based on using compression to automatically insert spaces back into the decrypted texts in order to achieve readability (as we perform our experiments on English alphabetic characters).

The paper is organised as follows. The next section gives a brief description of most of the previous research into the cryptanalysis of transposition ciphers. Section 3 motivates the use of our compression-based approach as a method of tackling the plaintext recognition problem and reviews the calculations of the codelength metric used by our method which is based on the PPM and Gzip compression schemes. Transposition ciphers are described in Sect. 4. The full description of our method and the pseudo-code is illustrated in Sect. 5. Our experimental results are discussed in Sect. 6.

2 Previous Work

Various cryptanalysis methods have been used to break transposition ciphers, starting with traditional attacks such as exhaustive search and anagramming, and then leading to genetic algorithm based methods. Anagramming is a well-known traditional cryptanalysis method. It is the method of repositioning disarranged letters into their correct and original positions [10,20,22]. Although, the traditional attacks are more successful and easy to implement, but automating these types of attack is not an easy issue. It requires an experienced and trained cryptanalyst. Mathematical techniques have been used in these attacks but the main role tends to be on the human expert. The final decision is made by the human cryptanalyst with regards to which algorithm is used in attack.

Many researchers have been interested in developing and automating cryptanalysis against transposition ciphers. One of the earliest papers was published by Matthews in 1993 [17]. He presented an attack on transposition ciphers using a genetic algorithm known as GENALYST. The fitness function was based on the frequency of the common English digrams and trigrams that appear in the deciphered text. This attack was only successful at key size of 7, with no successes at key length of 9 and 11. It achieved average success rates of 2–4 %.

Clark [5] published three algorithms that used simulated annealing, genetic algorithm and tabu search in the cryptanalysis of transposition ciphers. The fitness function used also depended on the frequencies of digrams and trigrams. By using a genetic algorithm, the success rates of block sizes of 4 and 6 ranged from 5 to 91 %. Tabu search was faster than the other algorithms while simulated annealing was the slowest but with a high performance of solving ciphertexts especially with large periods. It was able to correctly recover 26 of the key elements, for a transposition cryptosystem of period 30 [4]. Dimovsk and Gligoroski [8] came to similar conclusions presented in Clark's publication. The fitness function that was used in their paper was based on bigrams statistics due to the expensive task of calculating trigrams statistics.

Toemeh and Arumugam [26] used a genetic algorithm to break transposition ciphers. They used a slightly modified list of the most common bigram and

trigrams than were used in Clark paper. Three additional trigrams EEE, AND and ING were included with the Clark table. They concluded that when the ciphertext size is larger, the number of breakable keys increases.

In 2003, a genetic algorithm was presented by Grundlingh and Van Vuuren [12], which resulted in a 6–7 % success rate. The fitness function they used in their research was based on the discrepancies between the expected number of occurrences of a digram in a natural language text (per N characters), and the observed count of this digram in a ciphertext of length N. For this attack, no convergence of fitness function values had been found. They concluded that genetic algorithmic attacks were not effective against columnar transposition ciphers since this cipher is more robust than substitution ciphers. This attack was only successful at a key size of 7.

A permutation-based genetic algorithm was used by Bergmann, Scheidler and Jacob [2]. Two fitness functions were used in this research. The first function was based on calculating the redundancy of the decrypted text, then comparing it to the expected redundancy of the same sized English text and the same sized random text. The second function was based on the use of known text appearing in a message. Hamming Distance was used to check the presence of this supplied piece of known text in the decrypted text at a predetermined position. A transposition cipher with a key size of up to 12 and 500 characters in length was able to be deciphered correctly using this algorithm.

Giddy and Safavi-Naini [11] used a simulated annealing approach. The algorithm was not able to decrypt the cipher correctly, if the length of this ciphertext was short (100 characters or less). They noted that this is the supposed behaviour of all cryptanalysis schemes. Ciphertexts that have dummy characters added to them were decrypted poorly as well. The cost function here was based in terms of bigram frequencies.

As mentioned above, many of the research methods applied genetic algorithms to the automatic cryptanalysis of transposition ciphers. They used different key lengths ranging from 2 to 30 with different ciphertext lengths. None of these algorithms were able to correctly recover all the plaintext and achieve full success. In fact, Delman [7] concluded that the genetic algorithm-based approach did not deserve further effort. He stated that further investigation in traditional cryptanalysis techniques was warranted rather than for genetic algorithms.

Two heuristics were adopted by Russell, Clark and Stepeny [19]. They used the Ant Colony System algorithm which used a dictionary to recognise the plaintext, and bigrams to indicate adjacent columns. This attack was able to decipher shorter cryptograms (300 characters) by a factor of about a half compared to other previous meta-heuristic methods. Chen and Rosenthal [3] used a Markov chain Monte Carlo method to break transposition ciphers. Bigram statistics was used as a base to calculate the score function. This method presented a good performance with key length 20 and 2000 characters ciphertext. Wulandari, Rismawan and Saadah [27] presented another attack against transposition ciphers by using a differential evolution algorithm. The fitness function was based on bigram and trigram statistics. This algorithm was able to decrypt the ciphertext

correctly, with key lengths up to 9; with key length equal to 10, it was able to find half of the correct answers.

Irvine [15] has been the only researcher to date to have used a compression algorithm to break a cipher system—in this case, substitution ciphers. He used a compression algorithm (PPM model) combined with simulated annealing approach to perform an automated cryptanalysis of substitution ciphers. He was able to achieve a success rate of 83 %. However, a similar approach has yet to have been applied to other ciphers systems, including transposition ciphers.

In this paper, we will propose a novel compression-based approach for the automatic recognition of the plaintext of transposition ciphers with a 100 % success rate. We will use different key lengths (ranging from 2 to 12) and different ciphertext lengths, even very short messages with only 12 characters while the shortest messages used in the previous research was not less than 100 characters. In our paper, we will present both a method for automatically decrypting transposition ciphertexts and then automatically achieving readability subsequently. This will automatically insert spaces into the decrypted text, while most of the previous works did not address or refer to this fundamental aspect of the cryptanalysis.

3 Compression as a Cryptanalysis Method

The ciphertext only attack (cryptanalysis) of simple ciphers is not a trivial issue as evidenced by the wide range of literature in the previous section. It heavily depends on the source language and its statistical features. Particularly in ciphertext only attack, it is difficult to recognise the right decrypt quickly. Many of the published cryptanalysis techniques can not run without human intervention, or they assume that at least the plaintext is known, in order to be able to detect the proper decryption quickly.

Having a computer model that is able to predict and model natural language as well as a human is critical for cryptology [24]. Teahan showed that PPM compression models had the ability to predict text with performance levels close to expert humans [25]. The basic idea of our approach depends on using the PPM model as a method for computing the compression 'codelength' of each possible permutation which is a measure of the information [15] contained in each. Permutations with shorter codelengths help to reveal better decrypts. We show how to use this to easily and automatically recognise the true decrypt in a ciphertext only attack against transposition ciphers. In this paper, we also try another compression method, Gzip, as a basis for calculating the codelength metric in order to determine which is the most effective compression method to use to recognise the valid decryptions, but with significantly less success compared to the PPM-based method.

3.1 PPM Compression Code Length Metric

Prediction by Partial Matching, or PPM, is a finite-context statistical based approach. This technique was first described in 1984 by Cleary and Witten [6]. The major concept of this modelling is dependent on using previous symbols

(called the context of order k, where k represents the number of prior symbols) to predict the next or upcoming symbol. The number of symbols used in the context defines the maximum order of the context model.

For each context model, prediction probabilities can be calculated from the observation of all symbols or characters that have succeeded every length-k subsequence, as well as from the frequencies of occurrence for each character. A different predicted probability distribution is gained from each model; as a result, the probabilities of each character or symbol that has followed the previous k characters (the previous time) are used to estimate the next character. PPM models have been shown to be very effective at compressing English and have comparable performance to human experts' predictive ability [25].

Several PPM variations have been devised depending on the methods that have been suggested for computing the probabilities of each symbol. PPMD is one of these variants first developed by Howard [13]. Experiments show that in most cases, PPMD performs better than the other variants PPMA, PPMB and PPMC. It is similar to PPMC with the exception that the probability estimation of a new symbol or character is different. The treatment of the new symbols becomes more consistent [14] through adding $\frac{1}{2}$ to the count of the new symbol and to the escape count as well:

$$e = \frac{t}{2n} \quad \text{and} \quad p(s) = \frac{2c(s) - 1}{2n}$$

where e denotes the escape probability, $p(s)$ is the probability of symbol s, $c(s)$ is the number of times that the symbol s followed the context, n denotes the number of tokens that have followed and t refers to the number of types. For example, if a specific context has occurred three times previously, with three symbols a, b and c following it one time, then, the probability of each one of them is equal to $\frac{1}{6}$ and escape symbol probability is $\frac{3}{6}$.

The essential idea of our approach depends on using a PPMD compression model to compute codelength values of each possible permutation. From a compression point of view, the 'codelength' of a permutation for a cryptogram is simply the absolute ratio of the cryptogram length (in bits) when compressed to the cryptogram length in characters. In particular, the smaller the codelength, the more likely the cryptogram is close to the language source. By using this metric for assessing the quality of the solution, it can be used for finding valid decryptions automatically.

One of the most important steps in our implementation is the step of priming the compression models using a large set of English training data. This step allow us to overcome the problem of using an uninitialised compression scheme, where at the beginning of a message there is not enough and sufficient data to effectively compress the texts.

3.2 Calculating Codelengths Using the Gzip Compression Method

Gzip or GNU zip is one of the most important compression utilities. It is one of the most common lossless compression method on the Unix operating system

and on the Internet. It was written by Jean- Loup Gailly and Mark Adler. The Gzip compression scheme is a dictionary based approach (using Lempel-Ziv algorithm [9]) whereas PPM is a statistical context based approach.

The fundamental reason of experimenting with another compression method (Gzip) in our approach is to determine which is the most effective method, when applying to the problem of plaintext recognition using a compression-based approach.

In our paper, we will calculate the codelength values for the Gzip compression scheme by using a relative entropy technique. As a result, we do not need to re-implement the Gzip compression scheme itself, we can simply use "off-the-shelf" software. We calculate the codelength value using the relative entropy method using the equation $h_t = h_{T+t} - h_T$ where T is some large training data, h represents the file size after it has been compressed, and t represents the testing text. Simply, the basic idea of this method is to calculate the difference (in size) between the compressed training text with the testing text added to it compared to the size of the compressed training text by itself.

We also tested our relative entropy method using another well-known compression scheme, Bzip2. This is another lossless compression scheme that uses a block sorting algorithm. However, due to the algorithm's nature, some of the relative entropy calculations ended up being negative, so these could not be used.

4 Transposition Ciphers

In cryptography, a transposition cipher is a method of encryption by which the content of a message is concealed by rearranging groups of letters, therefore resulting in a permutation. The concept of transposition is an essential one and has been used in the design of modern ciphersystems [23]. Originally, the message was written out into a matrix in row-order and then read out by column-order [15]. The technique can be expanded to d dimensions, by dividing a message into blocks or groups of fixed size d (called the period) and perform a permutation over these blocks. This permutation represents the key. The size of the key is the same as the length of the block. Generally, if $f : Z_d \rightarrow Z_d$ is a permutation over Z_d, $Z_d = \{1, ..., d\}$, then according to f, blocks of fixed length (d characters) are encrypted by applying a permutation to the characters [18,21]. For example, if $d = 4$ and the plaintext $x = 1234$ then the encrypted message (ciphertext) f might have the permutation: $f(x)$: 4213. Here, the first character in the original message is moved to the third position, the third character in the block to the fourth position, and the fourth character to the first position. Thus the original message cryptographydemo is encrypted as:

Position: 1234 1234 1234 1234
Plaintext: cryp togr aphy demo
Ciphertext: prcy rotg ypah oedm

This ciphertext is divided into blocks of four letters and in order to hide the key size (period), a stream of characters is transmitted continuously. In the case

of a short block at the end, it would be encrypted by moving the letters to their relative permutation positions with dummy letters added or just left blank.

In general, transposition ciphers are considered much harder to crack than other basic cryptosystems such as simple substitution ciphers. Many statistical tools have been developed aiding automated cryptanalysis of substitution ciphers while the automatic cryptanalysis of transposition ciphers has proven more difficult. Generally, cryptanalysis of transpositions is quite interventionist in that it requires some knowledge of the probable contents of the encrypted text to give an idea into the rearrangement order that has been used [17].

5 Our Method

In this section, we will give a full description of the new approach for the automatic cryptanalysis of transposition ciphertext. The basic idea of our approach depends on using a compression model as a method of computing the 'codelength' of each possible permutation. This compression model and the codelenghth metric represent the assessment function that can be relied on it to automatically rank alternative permutations and recover correct messages. In our method, the PPMD and Gzip compression methods are used in the experiment.

Our new approach consists of two essential phases. The main idea of the first phase (Phase I) depends on trying to break a ciphertext automatically using a transposition of specified size by exhaustively computing all possible transpositions. The second phase (Phase II) focuses on inserting spaces automatically (segmenting words) into the decrypted message which is outputted from the first phase in order to achieve readability (since we remove spaces from the ciphertext at the beginning of Phase I, as is traditional).

The pseudo-code for our method is presented in Figs. 1, 2 and 3. At the beginning of the first phase (Phase I), we remove all the spaces from the encrypted message as presented in line 3 in Fig. 1. The approach at this stage uses text comprising just 26 alphabetic English characters). The text is divided into blocks with a specified size according to the period (key size) of transposition (see line 6 in Fig. 1). Then, all possible transpositions are generated, and the compression codelength recomputed at each stage (see line 7 to 10 in the next figure). PPM and Gzip are used as the means for computing the codelength. As we start to get permutations with smaller codelength values, this means we are closer to finding the correct message. In other words, the hope is that the cryptogram that has the smallest codelength value will represent the valid decrypted message.

As the output of the previous phase are texts without any spaces, the second phase focuses on inserting spaces into the input text automatically 'segmenting words', in order to achieve readability. We have investigated two alternative ways of doing this. In the first method, Phase II-A, according to the decrypted message permutation which is outputted from the previous phase, all further possibilities are explored where a space is inserted after each character. Underperforming possibilities which have bad colelength values are pruned from the priority queue and only the best performing possibilities are returned at the end (see Fig. 2).

```
1- READ ciphertext
2- SET maximum-period of transposition to a specified size
3- REMOVE spaces from the ciphertext
4- FOR each period
5-    IF ciphertext length divided by period equal to 0 THEN
6-        DIVIDE the ciphertext into blocks according to the period
7-        FOR each possible permutation
8-            CALCULATE codelength value using PPM compression
              model or Gzip
9-            STORE a permutation that have smaller codelength
              value in priority queue
10-       ENDFOR
11-   ENDIF
12- ENDFOR
13- RETURN priority queue
```

Fig. 1. Pseudo-code for main decryption phase, Phase I

In the second method, Phase II-B, the Viterbi algorithm is used to search for the best probable segmentation sequences (see Fig. 3). Compression schemes are used as a base of measuring codelengths and automatically detecting correct solutions in both these two alternatives (Phase II-A and Phase II-B). The pseudo code of the two approaches are shown in Figs. 2 and 3.

In our method, we have used two variants of the PPMD model, one without update exclusions [24] and the standard PPMD. This was done to investigate

```
1- SET priority queue Q1 to have the decrypted message from the priority
   queue from Phase I
2- REPEAT
3-     SET Q2 to empty
4-     FOR each message in the priority queue Q1
5-         CREATE new message with a single space addad
6-         INSERT modified message into Q2
7-         CALCULATE codelength value for the new message using PPM
           compression model or Gzip method
8-         STORE new message that have a smaller codelength value
           than those in the priority queue Q2
9-     ENDFOR
10-    IF there is any improvement in the codelength value THEN
11-        REPLACE Q1 with Q2
12- UNTIL there is no improvement in the value of the codelength
13- RETURN front of priority queue Q1
```

Fig. 2. Pseudo-code for Phase II-A

```
1- READ message from the priority queue from Phase I
2- USE Viterbi algorithm to search for the best probable segmentation
   sequences
3- RETURN Decrypted message which have best segmentations that present
   the best encoding sequence
```

Fig. 3. Pseudo-code for Phase II-B

which is the most effective model when applied to the problem of the automatic cryptanalysis of transposition ciphers.

In order to clarify and organize our experiments and results, we divide our different experiments into different variants with a specified label as shown in the Table 1.

Table 1. Variants used in our experiments

Variants	Phase I	Phase II-A	Phase II-B
Variant A	PPMD with no update exclusions	PPMD with no update exclusions	
Variant B	PPMD with no update exclusions		PPMD with no update exclusions
Variant C	PPMD	PPMD	
Variant D	PPMD		PPMD
Variant G	Gzip	Gzip	

According to Table 1, the first variant is called Variant A. In this variant, PPMD without update exclusions is used to calculate the compression codelength values. This is used for the main deception phase—Phase I and for Phase II-A as well. All cryptograms can be solved using an order-4 model. In the second variant Variant B, PPMD4 without update exclusions is used in both phases Phase I and in Phase II-B. The Viterbi algorithm is used in the second phase.

A different version of the PPMD compression model is used in the third variant, which is named "Variant C". The standard PPMD4 (with update exclusion) is used as the method for calculating the codelength values for both phases (Phase I and Phase II-A). Variant D uses the standard order-4 PPMD, as well in the calculation of the codelength values. For the second phase, Phase II-B, this compression model is also used as a basis for segmenting the words.

For variant G, we examine the effectiveness of another type of compression method which is the Gzip compression system. The Gzip algorithm is used in the main decryption phase and for the second phase "Phase II-A", as the basis for computing the codelength metric.

6 Experimental Results

In our method, the order-4 PPMD models were trained on nineteen novels and the Brown corpus using 26 and 27 character (including space) English text. After this training operation, these models remain static during cryptanalysis. In our experiments, we use a corpus of 90 cryptograms with different lengths form different resources as testing texts. The lengths of the ciphertexts that have been examined in our experiments are ranging from 12 letters to over 600 letters. Table 2 presents a sample of decryption.

Table 2. Output sample for the different phases for the ciphertext 'prcy rotg ypah oedm'. (Compression codelengths are listed in bits with the lowest 5 results presented for Phase-II-A.)

Phase I	Phase II-A	Phase II-B
53.73 cryptographydemo	42.85 cryptography demo	Cryptography demo
	50.94 cryptographyde mo	
	59.41 cryptographyd emo	
	59.68 cryptograph ydemo	
	67.64 c ryptographydemo	

A random key is generated to encipher the original text (plaintext) for each run. After that, the attack is performed on the ciphertext. Different key sizes (period or permutation size) and different ciphertexts with different lengths have been experimented in our method. The results of the first phase Phase I by using the PPM method showed that all the valid decryptions were recognised and all the ciphertexts were able to be decrypted successfully with no errors. In our method, and with different variants, except Variant G, we achieve a success rate of 100 %. We have used different key size (block sizes) from two to twelve. We experimented with 90 different standard ciphertexts with different lengths (including very short) with different key sizes and all can be solved correctly. In contrast, by using the Gzip algorithm in the last variant (G), we achieved a success rate of 94 % as result of Phase I.

For each variant, we have performed two types of experiments (except for Variant G). Since in Phase I we deal with texts without any spaces included, our first experiment is done by using PPMD models after being trained on 26 English characters (instead of 27) as the basis of calculating codelengths. In the other experiment, PPMD compression models trained on 27 character English texts were used. The output result from these two experiments is the same achieving a 100 % success rate.

The second phase focused on inserting spaces automatically into the decrypted message that is outputted from the first phase. We used the Levenshtein distance as a metric of measuring differences between the plaintext and the decrypted text with spaces. Levenshtein distance metric is a commonly used string metric for

counting the differences between two strings (such as insertions, deletions or substitution) [16]. In our approach, in almost all cases the correct (readable) solution was found. The next table (Table 3) provides example output (with spaces) produced by the different variants.

Table 3. Example of solved cryptograms with spaces by different variants.

Variants	Number of errors	Decrypted message (with spaces)
Variant A	2	an excuse is worse and more terrible than
		a li**ef o**r an excuse is a lie guarded
Variant B	0	an excuse is worse and more terrible than
		a lie for an excuse is a lie guarded
Variant C	3	a**ne**xcuse is worse and more terrible than
		a li**ef o**r an excuse is a lie guarded
Variant D	1	a**ne**xcuse is worse and more terrible than
		a lie for an excuse is a lie guarded
Variant G	14	a**ne**xcus**eisw**orse an**dm**or**et**errible
		tha**nalief**o**ran**excus**eisalieg**uarded

For variant A, Fig. 4 shows the number of errors for each testing text as a result of the second phase. Clearly, we can see that most of the space insertion errors are less than two. The results show that 50 % of texts have correctly inserted spaces with no errors, and more than 45 % of the cryptograms are solved with three errors or less. The errors that occurred in some of the solved cryptograms were minor ones, all involving spaces only. There are just three examples that showed either 6 or 7 errors, the main reason being that each of these examples had unusual words on particular topics not occurring in the training data.

Variant B produces less errors than other variants. The results show that 59 % of the decrypted texts have correctly added spaces with no errors. Furthermore, over 36 % of the examples are spaced with just two or one errors, and about 4 % with three errors. Just two examples had six errors and all of these are shown in Fig. 5.

Variant C produces slightly worse results, with just 46 % of examples having successfully inserted spaces without any errors with about 45 % are spaced with three errors or less. In addition, nine of the solved cryptograms have four errors or more. Figure 6 shows the results of variant C. On the other hand, variant D presents very good results, it produces similar results to variant B but with a few minor differences.

Figure 7 presents the number of errors for variant G as a result of phase two. Clearly the number of errors for each solved cryptogram is much higher, in this case with most of the space insertion errors being greater than 15. Moreover,

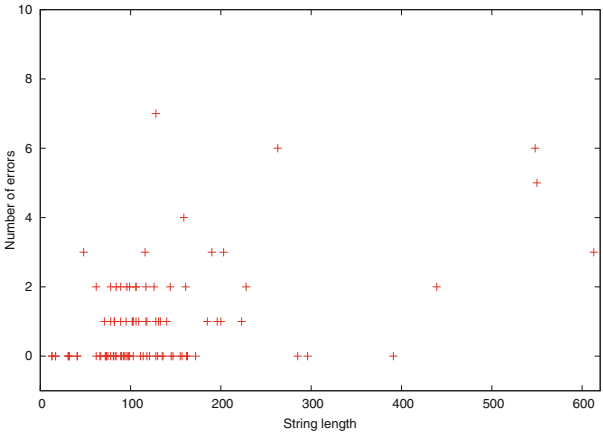

Fig. 4. Errors produced from variant A

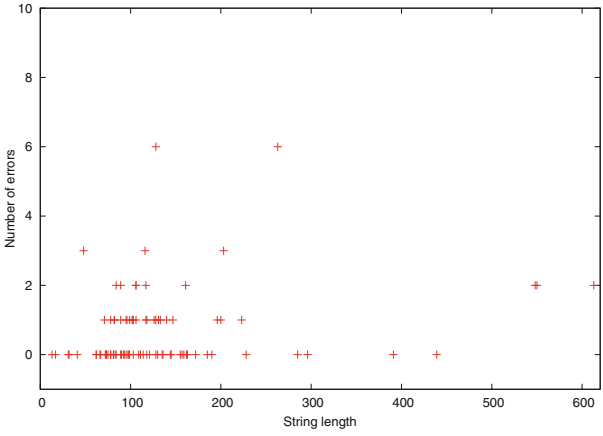

Fig. 5. Errors produced from variant B

none of the examples produced no errors and there is just five decrypted texts that were spaced with less than 10 errors.

Table 4 presents results concerning the average number of space insertion errors for the 90 texts we experimented with for each of the variants. It is clear that variant B produces the best results although other variants produce good results as well. What is interesting is that the PPM method without update exclusions which usually does slightly worse at the task of compression, does better here at decryption.

The last column in the table presents the number of average errors for variant G. The results show that the Gzip algorithm is not a good way for finding the right solutions with a high average number of errors.

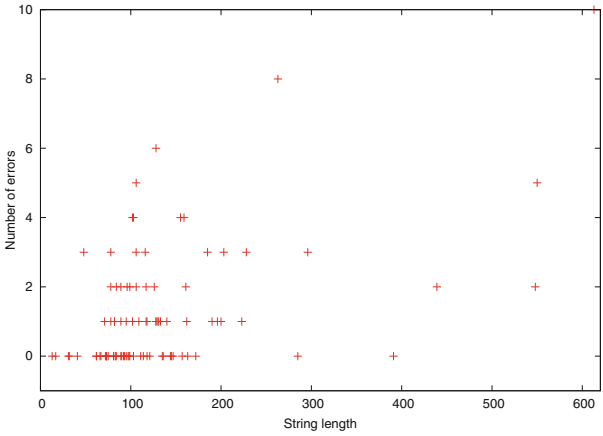

Fig. 6. Errors produced from variant C

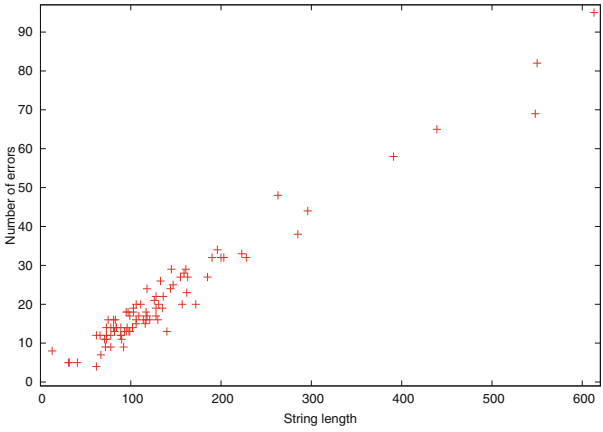

Fig. 7. Errors produced from variant G

In order to investigate further the accuracy of our spaces insertion algorithms in segmenting the 90 decrypted texts, we used three further metrics: recall rate, precision rate and error rate. Recall is calculated by dividing the number of correctly segmented words (using a compression model) by the total number of words in our original 90 testing texts. The error rate metric is calculated by dividing the number of incorrectly segmented words by the total number of words in the testing texts. Precision is calculated by dividing the number of correctly segmented words by the total number of words which are correctly and incorrectly segmented.

According to Table 5, it is clear that the first four variants, which are based on PPMD compression models, achieve quit high recall rates, which indicates

Table 4. Average number of errors for the phase two variants.

Variants	A	B	C	D	G
Average errors	1.02	0.69	1.30	0.81	21.62

Table 5. Recall, precision and error rates for the different variants on segmenting words.

Variants	Recall rate %	Precision rate %	Error rate %
Variant A	95.08	95.08	4.92
Variant B	96.30	96.38	3.70
Variant C	93.91	94.34	6.09
Variant D	95.71	95.91	4.29
Variant G	3.96	16.11	96.04

high accuracy of segmenting the 90 decrypted texts. All the errors generated, which are quite low, are those on unusual words not found in the training texts.

The average elapsed time that is required to find the valid decryptions of the transposition ciphertexts with different lengths for different key size is presented in Table 6. This table shows the average execution time for decrypting three ciphertexts of different lengths, for both the main decryption and spaces insertion phases combined (labelled as 'Both' in the table) and just for the main decryption phase (Phase I). The time which is needed to insert spaces automatically into the decrypted text (the second phase) is based on the execution of Phase II-A (slightly additional time is needed when using Phase II-B). The average execution time in seconds for each ciphertext is calculated by running the program ten times and then calculating the average.

In summary, the results showed that we are able to achieve 100 % success rate as a result of the first phase (Phase I) either by using standard PPMD or PPMD with-no update exclusions models. We manage to recognise all the plaintexts and solve all the cryptograms in this phase without any errors. We have used different block sizes (periods) ranging from two to twelve.

Table 6. Average required time to automatically cryptanalysis ciphertexts with different lengths for different keys size.

Ciphertext length (letter)	Key size											
	Time (in seconds)											
	5		6		7		8		9		10	
	Both	Phase I	Both	Phase I	Both	Phase I	Both	Phase I	Both	Phase I	Both	Phase I
40	0.72	0.68	0.73	0.69	0.77	0.75	0.97	0.93	2.40	2.38	12.07	12.06
150	1.12	0.7	1.14	0.73	1.20	0.80	1.77	1.35	13.75	10.06	48.07	47.07
300	3.39	0.71	3.62	0.75	3.71	0.87	4.86	1.97	23.01	20.41	95.32	92.41

In the second phase, we add spaces to these texts to improve readability by using two methods. The first method is a slightly faster new method based on a priority queue while the second method uses Viterbi algorithm to segment words. Our results show that almost all decrypted texts are segmented correctly. The maximum average number of errors (due to space insertions in incorrect places), when using PPMD compression models, is just slightly over one (for Variant C). This variant depends on the standard PPMD compression model as a basis for calculating the codelength values in Phase II-A. The results showed that by using the Viterbi algorithm (Phase II-B), we can gain slightly better results than the other method (Phase II-A), but it needs slightly more execution time. Variant B showed the best results. This variant depends on using a PPMD without update exclusions model using the Viterbi method as the basis for segmenting words.

7 Conclusions

We have introduced another use of the compression-based approach for cryptanalysis. A novel universal automatic cryptanalysis method for transposition ciphers and plaintext recognition method have been described in this paper. Experimental results have shown a 100 % success rate at automatically recognising the true decryptions for a range of different length ciphertexts and using different key sizes. This effective algorithm completely eliminates any need for human intervention. The basic idea of our approach depends on using a compression scheme as a base of calculating the 'codelength' metric, which is an accurate way of measuring information in the text. In this paper, we provided pseudo-code for two main phases: automatically decrypting ciphertexts and then automatically achieving readability using compression models to automatically insert spaces into the decrypted texts, as we performed our experiments on ciphertext in alphabetic English characters, while most previous works did not address this essential problem.

Two compression schemes have been investigated in our paper which are Predication by Partial Matching (PPM) and Gzip. The experimental results showed that PPM notably outperforms the other compression scheme Gzip. We also found that both PPMD variants were able to recognise all the valid decryptions. Concerning automatically adding spaces (word segmentation) afterwards, PPMD without update exclusions performs slightly better than the standard PPMD method (with update exclusions). The algorithm was able to achieve 100 % success rate using the PPM compression model on different amounts of ciphertext ranging from 12 to 625 characters, and with different key lengths ranging from 2 to 12. Larger key sizes and longer ciphertext length can be used, but of course it will require longer execution times to perform the decryptions.

Acknowledgments. The authors would like to thank the Iraqi Ministry of Higher Education and Scientific Research (MOHESR)-Baghdad University-College of science for women for supporting (sponsoring) this work.

References

1. Bell, T.C., Cleary, J.G., Witten, I.H.: Text Compression. Prentice-Hall, Upper Saddle River (1990)
2. Bergmann, K.P., Scheidler, R., Jacob, C.: Cryptanalysis using genetic algorithms. In: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, pp. 1099–1100. ACM, July 2008
3. Chen, J., Rosenthal, J.S.: Decrypting classical cipher text using Markov chain Monte Carlo. Stat. Comput. **22**(2), 397–413 (2012)
4. Clark, A.J.: Optimisation heuristics for cryptology. Ph.D. thesis, Queensland University of Technology (1998)
5. Clark, A.J.: Modern optimisation algorithms for cryptanalysis. In: Proceedings of the 1994 Second Australian and New Zealand Conference on Intelligent Information Systems, pp. 258–262. IEEE, December 1994
6. Cleary, J., Witten, I.: Data compression using adaptive coding and partial string matching. IEEE Trans. Commun. **32**(4), 396–402 (1984)
7. Delman, B.: Genetic algorithms in cryptography (2004)
8. Dimovski, A., Gligoroski, D.: Attacks on the transposition ciphers using optimization heuristics. In: International Scientific Conference on Information, Communication and Energy Systems and Technologies, ICEST, October 2003
9. Gailly, J.L.: GNU gzip (2010)
10. Gaines, H.F.: Cryptanalysis: Study a of Ciphers and Their Solutions. Dover, New York (1956)
11. Giddy, J.P., Safavi-Naini, R.: Automated cryptanalysis of transposition ciphers. Comput. J. **37**(5), 429–436 (1994)
12. Grundlingh, W., Van Vuuren, J.H.: Using genetic algorithms to break a simple cryptographic cipher. Retrieved 31 March 2003
13. Howard, P.G.: The design and analysis of efficient lossless data compression systems (1993)
14. Howard, P.G., Vitter, J.S.: Practical implementations of arithmetic coding. Department of computer science. Brown university. Providence, Rhode Island 02912, CS-91-45, July 1991
15. Irvine, S.A.: Compression and cryptology. Ph.D. thesis, University of Waikato, NZ (1997)
16. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Sov. Phys. Dokl. **10**(8), 707–710 (1966)
17. Matthews, R.A.: The use of genetic algorithms in cryptanalysis. Cryptologia **17**(2), 187–201 (1993)
18. Robling, D., Dorothy, E.: Cryptography and data security (1982)
19. Russell, M.D., Clark, J.A., Stepney, S.: Making the most of two heuristics: breaking transposition ciphers with ants. In: The 2003 Congress on Evolutionary Computation, CEC 2003, vol. 4, pp. 2653–2658. IEEE, December 2003
20. Seberry, J., Pieprzyk, J.: Cryptography: An Introduction to Computer Security. Prentice Hall, Sydney (1989)
21. Shannon, C.E.: Communication theory of secrecy systems. Bell Syst. Tech. J. **28**(4), 656–715 (1949)
22. Sinkov, A.: Elementary Cryptanalysis. Mathematics Association of America, Random House (1966)
23. Stamp, M., Low, R.M.: Applied Cryptanalysis: Breaking Ciphers in the Real World. John Wiley and Sons, Hoboken (2007)

24. Teahan, W.J.: Modelling English text. Ph.D. thesis, University of Waikato, New Zealand (1998)
25. Teahan, W.J., Cleary, J.G.: The entropy of English using PPM-based models. In: Proceedings Data Compression Conference, Snowbird, Utah, pp. 53–62 (1996)
26. Toemeh, R., Arumugam, S.: Breaking Transposition Cipher with Genetic Algorithm. Electron. Electr. Eng. **7**(79) (2007)
27. Wulandari, G.S., Rismawan, W., Saadah, S.: Differential evolution for the cryptanalysis of transposition cipher. In: 2015 3rd International Conference on Information and Communication Technology (ICoICT), pp. 45–48. IEEE, May 2015