An Automatic Cryptanalysis of Playfair Ciphers Using Compression

Noor R. Al-Kazaz[1]
School of Computer Science
Bangor University
Bangor, UK
n.al-kazaz@bangor.ac.uk
noor82.nra@gmail.com

Sean A. Irvine
Real Time Genomics
Hamilton, New Zealand
sairvin@gmail.com

William J. Teahan
School of Computer Science
Bangor University
Bangor, UK
w.j.teahan@bangor.ac.uk

Abstract

This paper introduces a new compression-based approach to the automatic cryptanalysis of Playfair ciphers. More specifically, it shows how the Prediction by Partial Matching ('PPM') data compression model, a method that shows a high level of performance when applied to different natural language processing tasks, can also be used for the automatic decryption of very short Playfair ciphers with no probable word. Our new method is the result of an efficient combination between data compression and simulated annealing. The method has been tried on a variety of cryptograms with different lengths (starting from 60 letters) and a substantial majority of these ciphers are solved rapidly without any errors with 100% of ciphers of length over 120 being solved. In addition, as the spaces are omitted from the ciphertext traditionally, we have also tried a compression-based approach in order to achieve readability by adding spaces automatically to the decrypted texts. The PPM compression model is used again to rank the solutions and almost all the decrypted examples were effectively segmented with a low average number of errors. Furthermore, we have also been able to break a Playfair cipher for a 6×6 grid using our method.

1 Introduction

Compression can be used in several ways to enhance cryptography and cryptanalysis. For example, many cryptosystems can be broken by exploiting statistical regularities or redundancy in the source. Since compression removes redundancy from a source, it is immediately apparent why compression is advocated prior to encryption (Irvine, 1997). However, this paper considers another application of compression to tackle the plaintext identification problem for cryptanalysis. This is an approach that has resulted in relatively few publications compared to the many other methods that have been proposed for breaking ciphers. The purpose of this paper is to explore the use of a compression model for the automatic cryptanalysis of Playfair ciphers.

The primary motivation for data compression has always been making messages smaller so they can be transmitted more quickly or stored in less space. Compression is achieved by removing redundancy from the message, resulting in a more 'random' output. There are two main classes of text compression adaptive techniques: dictionary based and statistical (Bell et al., 1990). Prediction by Partial Matching ('PPM'), first described in 1984 (Cleary and Witten, 1984), is an adaptive statistical coding approach, which dynamically constructs and updates fixed order Markov-based models that help predict the upcoming character relying on the previous symbols or characters being processed. PPM models are one of the best computer models of English and rival the predictive ability of human experts (Teahan and Cleary, 1996).

Our new approach to the automatic cryptanalysis of Playfair ciphers uses PPM compression to tackle the plaintext recognition problem. We rank the quality of the different plaintexts using the size of the compressed output in bits as the metric. We also use another PPM-based algorithm to automatically insert spaces into the decrypted texts in order to achieve readability.

This paper is organised as follows. Section 2 covers the basics of Playfair ciphers and

[1] Computer Science Department, College of Science for Women, Baghdad University, Baghdad, Iraq.

Proceedings of the 1st Conference on Historical Cryptology, pages 115– 124,
Uppsala, Sweden, 18-20 June, 2018

also includes a general overview of previous research on the cryptanalysis of Playfair ciphers as well as a discussion of its weaknesses. Our PPM based method and the simulated annealing search we use are explained in section 3. Section 4 covers the experimentation and results obtained with the conclusions to our findings presented in the final section.

2 Playfair Ciphers

The Playfair cipher is a symmetric encryption method which is based on bigram substitution. It was first invented by Charles Wheatstone in 1854. The cipher was named after Lord Lyon Playfair who published it and strongly promoted its use. It was considered as a significant improvement on existing encryption methods. A key is written into a 5×5 grid and this may involve using a keyword (as in the example below). For English, the 25 letters are arranged into the grid with one letter omitted from the alphabet. Usually, the letter 'I' takes the place of letter 'J' in the text to be encrypted.

To generate the key that is used, spaces in the grid are filled with the letters of the keyword and then the remaining spaces are filled with the rest of the letters from the alphabet in order. The key is usually written into the top rows of the grid, from left to right, although some other patterns can be used instead. For example, if the keyword 'CRYPTOLOGY' is used, the key grid would be as below:

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

To encrypt any plaintext message, all spaces and non-alphabetic characters must be removed from the message at the beginning, then the message is split into groups of two letters (i.e. bigrams). If any bigrams contain repeated letters, an 'X' letter is used to separate between them. (It is inserted between the first pair of repeated letters, and then bigram splitting continues from that point). This process is repeated (as necessary) until no bigrams with repeated letters. If the plaintext has an odd number of letters, an 'X' is inserted at the end so that the last letter is in a bigram (Klima and Sigmon, 2012). For example, the message "*To be or not to be that is the question*" would end up as:
"TO BE OR NO TX TO BE TH AT IS TH EQ UE ST IO NX".

There are three basic encryption rules to be applied (Klima and Sigmon, 2012):

- If both letters of the bigram occupy the same row, replace them with letters to the immediate right respectively, wrapping from the end of the row to the start if the plaintext letter is at the end of the row.

- If both letters occupy the same column, then replace them with the letters immediately below them. So 'IS' enciphers to 'SZ'. Wrapping in this case occurs from the bottom to the top if the plaintext letter is at the bottom of the column.

- If both letters occupy different rows and columns, replace them with the letters at the free end points of the rectangle defined by both letters. Thus 'TO' enciphers to 'CB'. The order is important—the letters must correspond between the encrypted and plaintext pairs (the one on the row of the first letter of the plaintext should be selected first).

Following these rules, the encrypted message would be:

"CB LI LC KG PZ CB LI PI BP SZ PI HM VD ZB DB QW"

The Playfair cipher is one of the most well known multiple letter enciphering systems. However, despite the high efficiency demonstrated by this cipher, it suffers from a number of drawbacks. The existing Playfair method is based on 25 English alphabetic letters with no support for any numeric or special characters. Several algorithms have been proposed aiming to enhance this method (Srivastava and Gupta, 2011; Murali and Senthilkumar, 2009; Hans et al., 2014). One particular extended Playfair cipher method (Ravindra Babu et al., 2011) is based on 36 characters (26 alphabetical letters and 10 numeric characters). Here, a 6×6 key matrix was constructed with no need to replace the letter 'J' with 'I'. By using the same previous keyword 'CRYPTOLOGY', the key matrix in this case would be:

C	R	Y	P	T	O
L	G	A	B	D	E
F	H	I	J	K	M
N	Q	S	U	V	W
X	Z	0	1	2	3
4	5	6	7	8	9

Plaintexts containing any numerical values such as, contact number, house number, date of birth, can be easily enciphered using this extended method (Ravindra Babu et al., 2011).

2.1 Cryptanalysis of Playfair Ciphers

Different cryptanalysis methods have been invented to break Playfair ciphers using computer methods. An evolutionary method for Playfair cipher cryptanalysis was presented by Rhew (2003). The fitness function was based on a simple version of dictionary look-up with the fitness calculated based on the number of words found. However, results obtained from this method were poor with run-time requiring several hours. A genetic algorithm was proposed by Negara (2012) where character unigram and bigram statistics were both used as a basis of calculating the fitness function. The efficiency of the algorithm is affected by different parameters such as the genetic operators, ciphertext length and fitness function. Five initial keys out of twenty were successfully recognized in less than 1000 generations and ten out of twenty were fully recovered in less than 2000 generations. Two ciphertexts were examined in this paper: one with 520 characters and the other with 870 characters. Hammood (2013) presented an automatic attack against the Playfair cipher using a memetic algorithm. The fitness function calculation was based on character bigram, trigram and four-gram statistics. A ciphertext of 1802 letters was examined in this paper and 22 letters out of 25 were successfully recovered using this method.

Simulated annealing was successful at solving lengthy ciphers as reported by Stumpel (2017). However, he found that short Playfair ciphers of 100 letters or so were unable to be solved. Simulated annealing was also used with a tetragraph scoring function for the automatic cryptanalysis of short Playfair ciphers by Cowan (2008). Cowan managed to solve seven short ciphertexts (80-130 letters) that were published by the American Cryptogram Association.

In summary, several different cryptanalysis methods have been proposed aiming to break Playfair ciphers with varying degrees of success. However, most of these methods were focused on long ciphertexts of 500 letters or more, except Cowan's method (2008). A large amount of information that is provided by long ciphertexts makes breaking them easier while short Playfair ciphers are extremely difficult to break without some known words. In our paper, even Playfair ciphertexts as short as 60 letters (without a probable crib) have been successfully decrypted using our new universal compression-based approach. We use simulated annealing in combination with compression for the automatic decryption. Moreover, we have also effectively managed to break extended Playfair ciphers that use a 6×6 key matrix.

2.2 Playfair's Weaknesses

The Playfair cipher suffers from some major weaknesses. An interesting weakness is that repeated bigrams in the plaintext will create repeated bigrams in the ciphertext. Furthermore, a ciphertext bigram and its reverse will decipher to the same pattern in the plaintext. For example, if the ciphertext bigram "CD" deciphers to "IS", then the ciphertext "DC" will decrypt to "SI". This can help in recognising words easily, especially most likely words. Another weakness is that English bigrams that are most frequently occurring can be recognised from bigram frequency counts. This can help again in guessing probable plain words (Smith, 1955; Cowan, 2008).

Breaking short Playfair ciphertexts (less than 100 letters) without good depth of knowledge of previous messages or with no probable words has proven to be a challenge. Past research has often used much longer ciphertexts—for example, Mauborgne (1914) developed his methods by deciphering a Playfair ciphertext of 800 letters. Also, the Playfair messages that were circulating between the Germans and the British during war had enough depth with many probable words to make them easily readable between these two sides, with no predictor of decrypting success for short messages on anonymous topics (Cowan, 2008). However, the two conditions that the message is short with little depth (no probable words) apply to cryptograms published by the American Cryptogram Association.

3 Our Method

This section describes our new method for the automated cryptanalysis of the Playfair cipher. The problem of quickly recognising a valid decrypt in a ciphertext only attack has been acknowledged as a difficult problem (Irvine, 1997). What we require is a com-

puter model that is able to accurately predict natural language so that we can use it as a metric for ranking the quality of each possible permutation (Al-Kazaz et al., 2016). The PPM text compression algorithm provides one possibility since it is known that PPM compression models can predict language about as well as expert human subjects (Teahan and Cleary, 1996).

Hence, the main idea of our approach depends on using the PPM method to compute the compression 'codelength' for each putative decryption of the ciphertext with the given key. The codelength of a permutation for a cryptogram in this case is the length of the compressed cryptogram, in bits, when it has been compressed using the PPM language model. The smaller the codelength, the more closely the cryptogram resembles the model. Experiments have shown that this metric is very effective at finding valid solutions automatically in other types of cryptanalysis (Al-Kazaz et al., 2016). In this paper, we show how to use this approach to quickly and automatically recognise the valid decrypt in a ciphertext only attack specifically against Playfair ciphers.

In the PPM compression algorithm, the probability of the next symbol is conditioned using the 'context' of the previously transmitted symbols. These probabilities are based on simple frequency counts of the symbols that have already been transmitted. The primary decision to be made is the maximum context length to use to make the predictions of the upcoming symbol. The 'order' of the model is the maximum context length used to make the prediction. Many variants of the original Cleary and Witten approach have been devised such as PPMA, PPMB, PPMC and PPMD. These differ mainly by the maximum context length used, and the mechanism used to cope with previously unseen or novel symbols (called the zero frequency problem). When a novel symbol is seen in a particular context, an 'escape' is encoded, which results in the encoder backing off to the next shorter context. Several escapes may be needed before a context is reached which predicts the symbol. It may be necessary to escape down to the order 0 (null) context which predicts each symbol based on the number of times it has occurred previously, or for symbols not previously encountered in the transmission stream, a default model is used where an order '-1' context predicts each symbol with equal probability.

Most experiments show that the PPMD variant developed by Howard (1993) produces the best compression compared to the other variants. The probabilities for a particular context using PPMD are estimated as follows:

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

where $p(s)$ is the probability for symbol s, $c(s)$ is the number of times symbol s followed the context in the past, n is the number of times the context has occurred, t denotes the number of symbol types and e is the probability assigned to perform an escape. 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 $\frac{3}{6}$.

As PPM is normally an adaptive method, at the beginning there is insufficient data to effectively compress the texts which results in the different permutations producing similar codelength values. This can be overcome by priming the models using training texts that are representative of the text being compressed. In our experiments described below, we use nineteen novels and the Brown corpus converted to 25 letter English by case-folding to upper case with I and J coinciding for the 5×5 grid and 36 alphanumeric characters for the 6×6 grid to train our models. Also, unlike standard PPM which uses purely adaptive models, we use static models which are not updated once they have been primed from the training texts.

Our new method is divided into two main phases. The first phase (Phase I) is based on trying to automatically crack a Playfair ciphertext using a combination of two approaches, which is the compression method for the plaintext recognition and simulated annealing for the search. The second phase (Phase II) is based on achieving readability by automatically adding spaces to the decrypted message produced from phase I, as the spaces are omitted from the ciphertext traditionally.

A variation of an order 5 PPMD model without update exclusions has been used in our experiments for both Phase I and Phase II. This variation is where symbol counts are updated for all contexts unlike standard PPM where only the highest order contexts are updated

until the symbol has been seen in the context. In our experiments, this variation has proven to be the most effective method that can be applied to the problem of automatically recognising the valid decryption for Playfair ciphers, but also in other experiments with transposition ciphers (Al-Kazaz et al., 2016).

Simulated annealing is a probabilistic method for approximating the global optimisation of a given function in a large search space. It is a descendant of the hill-climbing technique. This latter technique is based on starting with a random key, followed by a random change over this key such as swapping two letters, to generate a new key. If this key produces a better solution than the current key, it replaces the current one. Different n-graph statistics were used as the scoring function to judge the quality of solutions. After millions of distinct random changes, this technique attempts to discover the correct key.

The weakness of this approach lies in the possibility of being stuck in local optima, where the search has to be abandoned and it is necessary to restart all over again. Simulated annealing (inspired by a process similar to metal annealing) is similar to hill-climbing with a small modification that often leads to an improvement in performance. In addition to accepting better solutions, simulated annealing also accepts worse solutions in order to avoid the local optima. This approach permits it to jump from local optima to different locations in order to find new optima. The probability of the acceptance of the specific solution is dependent on how much the score value is worse. The formula for calculating the acceptance probability is $P_A = \frac{1}{e^{(d/T)}}$ where e is the exponential constant 2.718, d denotes the difference between the score of the new solution and the score of the current solution, and T is a value called temperature (further details concerning this parameter are described below). Whenever the difference is small, the probability of accepting the new solution is high, while if this solution is much worse than the current one (the difference is large in magnitude), the probability becomes small. The probability value is also influenced by the temperature T. Initially, the algorithm starts with a high temperature value, then it is reduced ('cooled') at each step according to some annealing schedule, until it reaches zero or some low limit. As the temperature drops, the probability of acceptance also decreases and when T is set to zero, the simulated annealing becomes identical to the hill climbing technique.

The main idea of using simulated annealing for the breaking of Playfair ciphers is to modify the current key in the hope of producing a better key. This is based on an approach proposed by Cowan (2008). This can be done by randomly swapping two characters. However, this random change is not enough to effectively break the Playfair cipher by itself. It will usually result in a long search process that often gets stuck within reach of the final solution. So other modifications are needed such as randomly swapping two rows, swapping two columns, reversing the key, and reflecting the key vertically and horizontally (flipping the key top to bottom and left to right). Using a mix of these modifications can lead to the valid solution. For example, swapping two rows will help rearrange rows if they are out of order, as it is very important that rows be in the correct order according to the encipherment rules (Lyons, 2012).

During the whole search process, the hope is that the best plaintext solution that appears is also the correct plaintext. Alternatively, the whole process must be restarted all over again and the value of the temperature should be reset to its original high value (Cowan, 2008). An important aspect of this whole process is the metric that is used to rank the different plaintexts (such as our PPM method). A good metric needs to be able to distinguish effectively between good and poor plaintexts.

Algorithms 1 and 2 present the pseudo code for the first phase of our method. In a preprocessing step prior to the applications of these algorithms, all non-letters including spaces, numbers and punctuation were removed from the ciphertext if a grid of 5×5 is chosen. If a 6×6 grid-width is selected, all non alphabetic letters and numbers were removed from the ciphertext instead. According to selected grid-width, a random key is generated (line 1) and the deciphering operation is initiated using this key. In order to rank the quality of the solutions, the PPM compression method is used by calculating the codelength value for each possible solution (lines 3 and 4). For each iteration, a sequence of changes is performed over the generated key in order to find a solution with a smaller codelength value which represents the valid decryption (lines 5 to 33). The greater the number of iterations, the more likely a solution will be found, but longer ex-

ecution time will be needed. It is important to note here that we have used negative scores based on the PPM codelengths values in order to maximize rather than minimize scores for the simulated annealing process as per the standard approach adopted in various solutions (Cowan, 2008; Lyons, 2012).

The temperature for the simulated annealing based algorithm is initially set to 20 and reduced by 0.2 in subsequent iterations. (The smaller this amount is, the more likely a solution will be found but this will also result in longer execution time). The initial temperature value is essentially dependent on the cryptogram's length. The shorter the ciphertext, the lower the temperature will be needed and vice versa. We have found in experiments with different length ciperhetexts that for cryptograms of a length of around 70, an initial temperature will need to start at around 10, but for the cryptogram of 700 characters, a temperature at 20 or so is effective.

For each temperature, 10,000 keys are tested then a reduction in the temperature is performed (see lines 9 to 32 in the algorithm). A loop is executed 10,000 times (lines 10 to 31) that modifies the key in the hope of finding a better key with a smaller codelength value. A sequence of different modifications over the key is performed in lines 11 to 17. The encrypted text is then deciphered using the modified key and the codelength value is calculated using the PPM compression method (lines 19 and 20). Then, the difference is calculated between the new codelength value and the previous one. If the new value (line 21) is better (that is, the codelength value is smaller), then the maximum score is set to the new score (line 22), otherwise a probability of acceptance is calculated (line 24) if the temperature is greater than 0 (line 23). In this case, a random number between 0 and 1 is generated, and if the calculated probability is greater than this number, the modified key is accepted (see lines 26 to 27). If we have a new best score, then the old one is replaced (line 29) and systematic rearrangements are performed by calling Algorithm 2. These include mutations (lines 4 to 10 in the new algorithm), row swapping and column swapping (lines 11 to 17) and an exhaustive search over all 4! possible permutations of each group of four symbols (lines 18 to 24). Swapping single pairs of letters results in the search getting stuck in local maxima too often, so we added the swapping of all possible combinations of 4 symbols to try to avoid that. Trying 3, 5, or even more combinations of symbols is possible, but of course the higher the number, the search starts getting very expensive, so 4 provides a reasonable compromise. Finally, the deciphered text is returned with the smaller codelength value which represents the best solution found (line 34). This has proved adequate for the solution of most ciphers, but if necessary, it is still possible to iterate the attack several more times.

Algorithm 1: Pseudo code of the main decryption phase 'Phase I'.

```
   Input  : ciphertext, Playfair grid-width to be either 5×5
            or 6×6
   Output: deciphered-text
 1 generate a random key according the Playfair grid-width
   selected
 2 currentBestKey ← randomKey
 3 decipher the ciphertext using the currentBestKey and
   calculate the codelength value using the PPM
   compression method
 4 currentBestScore ← − PPM-codelength score (decipher-text)
 5 for Iteration ← 0 to 99 by 1 do
 6     maxKey ← currentBestKey
 7     decipher and calculate the codelength value using
       the PPM compression method
 8     maxScore ← − PPM-codelength score (decipher-text)
 9     for Temp ← 20 downto 0 by 0.2 do
10         for Count ← 0 to 9999 by 1 do
11             modify maxKey by choose a random
               number between (1,50):
12                 if the number is 0 then swap two
                   rows, chosen at random
13                 if the number is 1 then swap two
                   columns, chosen at random
14                 if the number is 2 then reverse the
                   key
15                 if the number is 3 then reflect the
                   key vertically, flip top to bottom
16                 if the number is 4 then reflect the
                   key horizontally, flip left to right
17                 if any other number then swap two
                   characters at random
18             newKey ← modified-maxKey
19             decipher and calculate the codelength
               value using the PPM compression method
20             newScore ←
               − PPM-codelength score (decipher-text)
21             calculate diff ← newScore − maxScore
22             if diff >= 0 then {maxScore ← newScore;
               maxKey ← newKey}
23             else if Temp > 0 then
24                 calculate probability ← exp(diff/Temp)
25                 generate a random number between
                   (0,1)
26                 if probability > randomNumber then
27                     {maxScore ← newScore;
                       maxKey ← newKey}
28             if maxScore > currentBestScore then
29                 currentBestScore ← maxScore;
                   currentBestKey ← maxKey
30                 Make systematic
                   rearrangements(ciphertext,
                   currentBestKey, currentBestScore)
31         end
32     end
33 end
34 return the deciphered text with the best key
```

Concerning the second phase of our approach, Algorithm 3 illustrates the pseudo code for this phase. The main idea of this phase, as stated before, is to try to insert spaces into the deciphered text outputted from

Algorithm 2: Make systematic rearrangements

```
   Input  : ciphertext, currentBestKey, currentBestScore
   Output: currentBestKey, decipher-text
1  flag ← true
2  while flag do
3      flag ← false
4      perform systematic mutations over the
       currentBestKey:
5          decipher and calculate the codelength value
           using the PPM compression method
6          newscore ←
           – PPM-codelength score (decipher-text)
7          if newscore > currentBestScore then
8              flag ← true
9              currentBestScore ← newScore;
               currentBestKey ← newKey
10             continue outer While loop
11     perform systematic row-swaps and column-swaps
       over the currentBestKey:
12         decipher and calculate the codelength value
           using the PPM compression method
13         newscore ←
           – PPM-codelength score (decipher-text)
14         if newscore > currentBestScore then
15             flag ← true
16             currentBestScore ← newScore;
               currentBestKey ← newKey
17             continue outer While loop
18     perform swapping of four characters:
19         decipher and calculate the codelength value
           using the PPM compression method
20         newscore ←
           – PPM-codelength score (decipher-text)
21         if newScore > currentBestScore then
22             flag ← true
23             currentBestScore ← newScore;
               currentBestKey ← newKey
24             continue outer While loop
25 end
26 return currentBestKey, decipher-text
```

Phase I in order to achieve readability. PPM is again applied to rank the solutions. The Viterbi algorithm is used in this phase to find the best possible segmentation. In this algorithm, looping over the deciphered text (that was produced as output from Algorithm 1) is performed in line 2. A word segmentation algorithm based on the Viterbi algorithm (Teahan, 1998) is then used to search for the best performing segmentations to keep in a priority queue, and those which showed poor codelength values are pruned (see lines 3 to 5). The best segmented deciphered text is returned in the last line (line 6).

Algorithm 3: Pseudo code for Phase II

```
   Input  : the deciphered text from Phase I
   Output: segmented deciphered text
1  maximum size of Q1 (priority queue) ← 1;
2  do
3      use the Viterbi algorithm to search for the best
       segmentation sequences;
4      store the text that have the best segmentation
       which present in Q1;
5  while the end of the deciphered text;
6  return the best segmented deciphered text from Q1;
```

4 Experimental Results

In this section, we discuss the experimental results of our approach. As stated, in our method the order-5 PPMD model has been trained on a corpus of nineteen novels and the Brown corpus using 25 English letters (when a 5×5 grid is used) and 36 alphanumeric characters (when a 6×6 grid is used). After this training operation and during cryptanalysis, these models remain static. Regarding the cryptograms test corpus, 70 different cryptograms were chosen at random from different resources including cryptograms published by the American Cryptogram Association, cryptograms published by geocache enthusiasts, and two cryptograms that were also experimented with by Negara (2012). Cryptogram lengths ranged from 60 to 750 letters.

A sample trace of a decryption is shown in Figure 1 for the cryptogram: '`dohrxnwpscqusfrwchrnpctsehagvpstsfaprdtuipwolacgqupfwptslaqsizbedxqusfwscosfraevstngqu`'. This shows the best score as it changes during the execution of Algorithm 2 for the main decryption phase. The scores are increasing (i.e the codelengths are decreasing). The solution of this ciphertext is a proverbial wisdom that has been attributed to Damon Runyon: "*It may be that the race is not always to the swift nor the battle to the strong but that is the way to bet*". This ciphertext is one of the short cryptograms (82 character long) that have been published by the American Cryptogram Association, which usually publishes 100 ciphertexts every two months including one or more Playfair ciphers, as a challenge to its members (Cowan, 2008). Cowan has stated that it is extremely difficult to break short messages of 100 letters or so, especially when there are no suspected probable words or cribs and very little depth of knowledge of previous messages. However, our method is able to solve the following examples in addition to the other cryptograms that were listed by Cowan as well as even shorter ciphertexts of 60 letters or so.

A second example in Figure 2 illustrates the robustness of our compression approach by showing how it is able to solve a very short cryptogram. The ciphertext is a 60 letter sentence (a quote by Garrison Keillor): *Cats are intended to teach us that not everything in nature has a purpose.* The best solution for this example is '`catsareintendedtoteachusthatnotexerythinginxnaturehaoapurposew`' with the best codelength value `-137.68` resulting in only two errors: `x`→`v` in '`exerything`' and `o`→`d` in '`hao`'.

```
Iteration:35
Mutation  -221.66 ridaybetoktxtherkdeconotalwaystothescru-
gain:     fyorthebrtxtletothestucngmitxthatoithewaytobetx
          Key:zkbncwagerfhmlduvxyitsqpo
Mutation  -220.15 ridaybetvstxthersaeksnotalwaystotheskru-
gain:     fyorthebatxtletothestukngmitxthatksthewaytobetx
          Key:  zcbnkwagerfhmlduvxyitsqpo
Mutation  -215.91 ridaybetvstxthersaemsnotalwaystothesmru-
gain:     fyorthebatxtletothestumngkitxthatmsthewaytobetx
          Key:  zcbnmwagerfhkldu vxyitsqpo
Mutation  -207.60 rmdaybetvstxthersaeisnotalwaystothesiru-
gain:     fnorthebatxtletothestningkmtxthatisthewaytobetx
          Key:  zcbniwagerfhkldu vxymtsqpo
Mutation  -204.66 itzaybetvstxthersaeisnotalwaystotheswiu-
gain:     fnorthebatxtletothestorngbutxthatisthewaytobetx
          Key:  dcbniwagerfhklzuvxymtsqpo
Mutation  -195.04 itmaybetvstxthersaeisnotalwaystotheswiu-
gain:     fnorthebatxtletothestomngbutxthatisthewaytobetx
          Key:  dcbniwagerfhklmuvxyztsqpo
Row-swap  -184.98 itmaybethvtxthervaeisnotalwaystotheswif-
gain:     tnorthebatxtletothestzongbutxthatisthewaytobetx
          Key:  dcbniwagerfhklmtsqpouvxyz
Row-swap  -162.46 itmaybethatxtheraceisnotalwaystotheswif-
gain:     tnorthebatxtletothestrongbutxthatisthewaytobetx
          key:  dcbnifhklmtsqpouvxyzwager
```

Figure 1: Example cryptogram of 82 letters from the American Cryptogram Association.

```
Iteration:89
          -164.32 catsareintencectoteakiusthhonotexerythi-
          nginxnaturehatapurposid
Mutation  -161.21 catsareintencectoteakiusthaonotexerythi-
gain:     nginxnaturehatapurposid
Mutation  -160.36 pltsapeintencectoteadbusthahnotexerythi-
gain:     nginxnatureoatapurposid
Mutation  -159.28 pltsapeintendedtoteacbusthahnotexerbthi-
gain:     nginxnatureoatapurposic
Mutation  -156.08 datsareintendedtoteakiusthaonotexerfthi-
gain:     nginxnaturehatapurposic
Mutation  -155.29 katsareintendedtoteakiusthatnotexerythi-
gain:     nginxnaturehaoapurposic
Mutation  -154.12 ratsakeintendedtoteakiusthatnotexeocthi-
gain:     nginxnaturehasapukposic
Mutation  -150.31 ratsileintendedtotealausthatnotexeokthi-
gain:     nginxnaturehasapudiospc
2-Mutation-149.97 ratsileintendedtoealhusthatnotexevcthi-
gain:     nginxnaturehasapudiosev
Mutation  -143.16 ratsaceintendedtoteachusthatnotexelvthi-
gain:     nginxnaturehasapucposev
Mutation  -138.52 catsareintendedtoteachusthatnotexerythi-
gain:     nginxnaturehaoapurposev
Mutation  -137.68 catsareintendedtoteachusthatnotexerythi-
gain:     nginxnaturehaoapurposew
```

Figure 2: Example short 60 letter cryptogram.

A third example is a puzzle cryptogram of 96 letters from the geocache world (`https://bcaching.wordpress.com/2008/08/08/puzzles-part-3/`): '`sa cb av hm ka do st th ps mn qs fr hm sx bt su tw tg wg mh mc ok sd oz ts fy tw ts vc ec gs gt wl dl sr oz tb tl ps tg ex cm co dl kh wl wg mh ex av`'. Figure 3 presents the intermediate results and the final solutions produced by each iteration for this cryptogram. According to this example, iteration 89 produced the best solution with the best score with a compression codelength value of 215.81 and is the valid decrypt.

Our method was also able to solve a 6×6 Playfair cipher with a few minor errors. The next sample is a cryptogram that was posted on a puzzles forum originating from geocache enthusiasts (`http://members2.boardhost.com/barryispuzzled/msg/1500564217.html`):

```
Iteration:0
Mutation  -311.85 tuemuirecolstaurytrtforxreafmstoopanile-
gain:     rcekqbsulatydopatiekedanalondtfulsmonytancheck-
          andeqloilerchui
...
Iteration:2
          -311.01 adpcdhowwhsvalarcaucedofowleyldmailiumw-
          oseheatrelarbailaonmemlilgatstorelylscalikeese-
          ctswpgaumwokedh
...
Iteration:24
          -309.15 hkxepmmaskbitesratokhdtvmabasedaeferela-
          mlamntbinetnrefetlpgwhereeceithinesevaterbcalx-
          leismecelambcpm
...
Iteration:30
Row       -304.26 amsegplaysonasarealmcarblaterstcrustita-
swap      lsniymeinsahirusaezkstatsorodtbinsrmreastpensn-
gain:     kodyboritalpegp
...
Iteration:89
Row       -215.81 thecoxordinatesarenorthfortydegrexeszer-
swap      opointfivethrexetwowestseventyfivedegreestwopoi-
gain:     ntfivezerotwox
...
Iteration:100
```

Figure 3: Solutions produced during selected iterations for a puzzle cryptogram of 96 letters.

```
lpqtj zfpvf ndsvb joamd j4mva nrfeu nbhis nhcru
chhfs otb1e kbueg qejtv kgscn kq3ez kgwix eavej
nstda usbfj cvkgs cbtqz 5nmqa nc0jc xeiue nhtnb
rbwhg krabz 1j0mn dierf rabfq vjvkf dnrbk nkd0u
cbwhn dsdlv vpvha gvucb bnyjc vtzpf brrab gtmqa
j4fmt ryjbu vldtq fsnts awfhl pvthc 0hpvv obvmd
jvra7 zhfew irgh3 8qpck r5z7r gaw1k biyjg h3w8w
qfau3 v7dra bfnbe jgkke kvhfc 0dsjy raghx bjbqm
bhhgc hnwyj bxgkk ekvhf dcvjo uebxr rsch1 jwmvu
bxlbi nmraw ckbnh g1jrn dtchl vfpur raihj 4fmoq
jbrbj zfmtr yjbur acjck vughh tchjv rauxc hrzkc
hchff elnob mvvjh cgerf rgawu xchrz uxvfb fcqbt
mdfjh chgrf sujep qbrej yjrgy jchmv eseue ckgau
ejrbr uvlej mq1jv mh0mv txhz
```

Part of the execution trace is shown in Fig. 4.

```
Iteration: 1
-1604.59 jolxlygoodandwelldoneyouhavecrackedanextendedpl-
ayfaircipherusingasixbysixgridtheadvantageofusingall30sl-
phanumericxcharactersisthatyoucangivethecoxordinatesasnu-
mbersandnotwordsbutbecarefultogetthefullkeycorxrecttoavo-
idawastedjourneynowofftonorthztdegrexesx3poinwytzwest3de-
greess5x9xoint1x2athecacheishidxdenunderthestilepleaseens-
ureitishidxdenfromviewxwhenyouputitbackjustincaseyouhave-
notgotxthekeycompletelycorxrecttheminutesarenorthtwentyn-
inedecimalfouronefivewesttwentytwodecimaloneonesevenweho-
petherewereenoughcribsinthetexttohelpyouonyourway
-1594.57 jolxlygoodandwelldoneyouhavecrackedanextendedpl-
ayfaircipherusingasp5bysixgridtheadvantageofusingall50kl-
phanumericxcharactersisthatyoucangivethecoxordinatesasnu-
mbersandnotwordsbutbecarefultogetthefullkeycorxrecttoavo-
idawastedjourneynowofftonorthztdegrexesw3poinxytzwest5de-
greeswp3point1w2athecacheishidxdenunderthestilepleaseens-
ureitishidxdenfromviewxwhenyouputitbackjustincaseyouhave-
notgotxthekeycompletelycorxrecttheminutesarenorthtwentyn-
inedecimalfouronefivewesttwentytwodecimaloneonesevenweho-
petherewereenoughcribsinthetexttohelpyouonyourway
```

Figure 4: Example solutions produced for a 6×6 Playfair cryptogram.

The experimental results of Phase I, when the order 5 PPM method without update exclusions is used, showed that most of the cryptograms are successfully decrypted with no er-

rors. Table 1 presents the results from testing ciphertexts for various lengths. The results overall showed that we are able to attain very high success rates and 60 ciphertexts out of 70 were efficiently solved. Also, 100% of ciphers of length greater than 120 were decrypted.

Cipher Length	**60-79**	**80-99**	**100-119**	**120-149**	**150-199**	**200-750**
No. of Ciphers	9	21	15	11	8	6
Success Rate (%)	67	81	80	100	100	100

Table 1: Results when testing ciphertexts with different lengths.

Referring to the second phase of our method, as the spaces are omitted from the ciphertext traditionally, this phase focuses on segmenting the decrypted messages that are outputted from the first phase. The edit distance (or Levenshtein distance) metric is used to qualify how the decrypted message is differentiated from the original message by counting the minimum number of the removal, insertion, or substitution operations required to transform one message into the other (Levenshtein, 1966). In almost all cases, the correct readable decryptions were efficiently found as the illustrated in Figure 5.

Ciphertext	byntlbneonnuimmzqnhpbkxnqmfqoqnmugclqmeuersuqpnzigqbqyipilqtku
Decrypted text	cats are intended to teach us that not exerything in nature ha o a purpose
Ciphertext	kuinbrnuikcnqmhuvgtnnmybkgbromruknqmnknqdmpvgniignkoneumokgpgxytqsu
Decrypted text	experience is the worst teacher it gives the test before presenting the lesson
Ciphertext	pqghqncnndqyhfqugqqmeusxqmfqdpkgqbitqdkunurqioinnlpgvqvpbmlvhuqoimigbzka
Decrypted text	a n egotist is a man who thinks that if he hadnt been born people would have wondered why
Ciphertext	qmghblxytkyfihogkunugiqoqmgnqmgincimtlqmmpnuikiwszqmgiknliqrbhafgtigtldnnqgtxz
Decrypted text	the grass may be greener on the other side of the fence but there s probably more of it to mow
Ciphertext	hmfnuwfntufdbgushmtuqmckqnutfpmatuzfmbfntylxqpthrkucnrkrmcqdamibarurntumucoffdummbrnki
Decrypted text	the likeliness of a thing happening is inversely proportional to its desirability fin agles first law
Ciphertext	dohrxnwpscqusfrwchrnpctsehagvpstsfaprdtuipwolacgqupfwptslaqsizbedxqusfwscosfraevstngqu
Decrypted text	it may be that the race is not always to the swift nor the battle to the strong but that is the way to bet

Figure 5: Example of solved ciphertexts with spaces inserted after Phase II.

The number of space insertion errors for each testing cryptogram is plotted in Figure 6. We can see that the number of errors for most cryptograms are very low and the correct segmentations are obtained in most cases. The average space insertion errors for the ciphertexts that were experimented with in Phase II is less than one error.

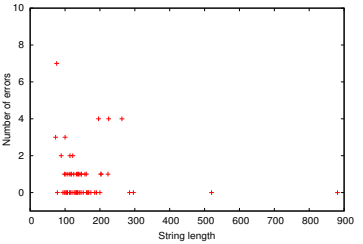

Figure 6: Segmenting errors produced as a result of the Phase II algorithm.

Table 2 lists the high recall and precision rates and the low error rate produced by our segmentation algorithm. The recall rate is calculated by dividing the number of successfully segmented words over the number of words in the original testing texts, the precision rate by dividing the number of successfully segmented words by the number of words which are correctly and incorrectly segmented and the error rate by dividing the number of unsuccessfully segmented words by the number of words in the original testing texts (Al-Kazaz et al., 2016).

Recall (%)	**Precision (%)**	**Errors (%)**
96.72	96.12	3.28

Table 2: Recall, precision and errors rates for our method for word segmenting the decrypted output produced from Phase I.

The execution times required to decrypt a number of Playfair ciphertexts by our method are presented in Table 3. This table shows the decryption time in seconds for Phase I of our method. The results indicate that our method produces reasonable decryption times, and in most cases the successful decrypts of longer ciphertexts were obtained after only one or two iterations.

Ciphertext Length *(Letter)*	**60**	**71**	**86**	**100**	**124**	**185**	**235**	**526**	**730**
Time *(Sec)*	457	539	507	93	36	17	135	107	101

Table 3: Decryption times for Phase I for different ciphertexts.

5 Conclusion

An automatic cryptanalysis of Playfair ciphers using compression has been introduced in this paper. In particular, a combination of simulated annealing and PPM compression was used in the automatic decryption method. The compression scheme was found to be an effective method for ranking the quality of each possible permutation as the search was performed. In 60 of the 70 ciphertexts that were experimented with (without using a probable word) for different lengths (from as short as 60 letters up to 750), almost all the correct solutions were found. The exception was just two very short ciphers which resulted in two minor errors in the decrypted output. Moreover, we have also managed to decrypt an extended Playfair cipher for a 6×6 key matrix.

In addition, a compression-based method was used to segment the decrypted output by insertion of spaces in order to improve readability. Experimental results show that the segmentation method was very effective producing on average less than one space insertion error with a recall and precision of over 96% for the ciphertexts that were tested.

As PPM provides a different type of scoring function compared to the standard n-gram analysis (such as update exclusions, the escaping back-off mechanism for smoothing the models), it is not clear whether using longer context for n-grams might lead to better results. It is also not clear how PPM compares to the standard n-grams approach and further experimentation (for example with hexagrams) needs to be done.

References

Noor R Al-Kazaz, Sean A Irvine, and William J Teahan. 2016. An automatic cryptanalysis of transposition ciphers using compression. In *Int. Conference on Cryptology and Network Security*, pages 36–52. Springer, Springer Int. Publishing.

Timothy C Bell, John G Cleary, and Ian H Witten. 1990. *Text compression*. Prentice-Hall, Inc.

John Cleary and Ian Witten. 1984. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402.

Michael J Cowan. 2008. Breaking short playfair ciphers with the simulated annealing algorithm. *Cryptologia*, 32(1):71–83.

Dalal Abdulmohsin Hammood. 2013. Breaking a playfair cipher using memetic algorithm. *Journal of Engineering and Development*, 17(5).

Swati Hans, Rahul Johari, and Vishakha Gautam. 2014. An extended playfair cipher using rotation and random swap patterns. In *Computer and Communication Technology (ICCCT), 2014 International Conference on*, pages 157–160. IEEE.

Paul Glor Howard. 1993. The design and analysis of efficient lossless data compression systems. Ph.D. thesis, Brown University, Providence, Rhode Island.

Sean A Irvine. 1997. Compression and cryptology. Ph.D. thesis, University of Waikato, New Zealand.

Richard E Klima and Neil P Sigmon. 2012. *Cryptology: classical and modern with maplets*. CRC Press.

Vladimir I Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710.

James Lyons. 2012. Cryptanalysis of the playfair cipher. `http://practicalcryptography.com/cryptanalysis/stochastic-searching/cryptanalysis-playfair/`.

Joseph Oswald Mauborgne. 1914. *An advanced problem in cryptography and its solution*. Fort Leavenworth, Kansas: Leavenworth Press.

Packirisamy Murali and Gandhidoss Senthilkumar. 2009. Modified version of playfair cipher using linear feedback shift register. In *Information Management and Engineering, 2009. ICIME'09. International Conference on*, pages 488–490. IEEE.

G Negara. 2012. An evolutionary approach for the playfair cipher cryptanalysis. In *Proc. of the Int. Conference on Security and Management (SAM)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

K Ravindra Babu, S Uday Kumar, A Vinay Babu, IVN S Aditya, and P Komuraiah. 2011. An extension to traditional playfair cryptographic method. *International Journal of Computer Applications*, 17(5):34–36.

Benjamin Rhew. 2003. Cryptanalyzing the playfair cipher using evolutionary algorithms. Available: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.129.4325&rep=rep1&type=pdf`.

Laurence Dwight Smith. 1955. *Cryptography: The science of secret writing*. Courier Corporation.

Shiv Shakti Srivastava and Nitin Gupta. 2011. Security aspects of the extended playfair cipher. In *Communication Systems and Network Technologies (CSNT), 2011 International Conference on*, pages 144–147. IEEE.

Jan Stumpel. 2017. Fast playfair programs. `www.jw-stumpel.nl/playfair.html`. last accessed December 13, 2017.

William J Teahan and John G Cleary. 1996. The entropy of English using PPM-based models. In *Data Compression Conference, 1996. DCC'96. Proceedings*, pages 53–62. IEEE.

William J Teahan. 1998. Modelling English text. Ph.D. thesis, University of Waikato, New Zealand.