An automatic cryptanalysis of simple substitution ciphers using compression

Alkazaz, Noor R.; Irvine, Sean A.; Teahan, William J.

Information Security Journal: A Global Perspective

DOI:
10.1080/19393555.2018.1426799

Published: 01/01/2018

Peer reviewed version

Information Security Journal: A Global Perspective manuscript No.
(will be inserted by the editor)

An Automatic Cryptanalysis of Simple Substitution Ciphers Using Compression

Noor R. Al-Kazaz · Sean A. Irvine · William J. Teahan

Received: date / Accepted: date

Abstract Automatic recognition of correct solutions as a result of a ciphertext only attack of simple ciphers is not a trivial issue and still remains a taxing problem. A new compression based method for the automatic cryptanalysis of simple substitution ciphers is introduced in this paper. In particular, this paper presents how a Prediction by Partial Matching ('PPM') text compression scheme, 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 simple substitution ciphers. Experimental results showed that approximately 92% of the cryptograms were decrypted correctly without any errors and 100% with just three errors or less. Extensive investigations are described in this paper, in order to determine which is the most appropriate type of PPM scheme that can be applied to the problem of automatically breaking substitution ciphers. This paper shows how a new character-based PPM variant significantly outperforms other schemes including the standard Gzip and Bzip2 compression schemes. We also apply a word-based variant which when combined with the character-based method leads to further improved results.

Keywords Cryptanalysis · substitution ciphers · plaintext recognition · compression · PPM.

1 Introduction

Compression can be used in several ways to enhance cryptography and cryptanalysis. For example, many cryptosystems can be broken by exploiting statis-

School of Computer Science, Bangor University.
College of Science for Women, Baghdad University.
{n.al-kazaz@bangor.ac.uk,w.j.teahan}@bangor.ac.uk
sairvin@gmail.com
noor82.nra@gmail.com
http://www.bangor.ac.uk/cs

tical properties or redundancy in the ciphertext. Clearly for this reason, compression is highly recommended before the encryption process, as it removes redundancy from the source (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.

This paper proposes a new compression based approach applied to the problem of automatically decrypting simple substitution ciphers with no need for any human intervention. In spite of the fact that simple substitution ciphers are relatively insecure compared to modern ciphering techniques, it still remains a classical problem that has defied many reliable automated cryptanalysis methods (Lucks, 1990). Our automatic cryptanalysis method uses a new variation of the Prediction by Partial Matching ('PPM') text compression scheme. This paper investigates different variants of PPM to ascertain the most efficient type when applied to the problem of decrypting simple substitution ciphers automatically using compression.

Text compression is about removing redundancy from a text source by reducing the space required to store the text and the time required to transfer this text as well without losing any information from the original source. In practical terms, two main classes of adaptive techniques are commonly used: dictionary and statistical approaches (Bell, Cleary, & Witten, 1990). The dictionary approach is usually found to be faster than the statistical approach. In contrast, statistical based approaches are usually better than dictionary approaches in terms of compression rate. More recently, a third class based on block-sorting using the Burrows-Wheeler algorithm (Burrows & Wheeler, 1994) has appeared which approaches the compression rates of statistical algorithms but at much faster speeds, although not as fast as dictionary-based approaches.

PPM 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. This class of text compression models performs well on English and it rivals the predictive ability of humans compared to other computer models (Teahan & Cleary, 1996). The Gzip compression program is an example of a dictionary-based method, which is based on the Lempel-Ziv algorithm (LZ77) (Gzip, 2012). The Bzip2 compression program (Bzip2, 2016) implements the Burrows-Wheeler algorithm.

The rest of this paper is organised as follows. Simple substitution ciphers are described in the next section. Section 3 provides a summary of the previous research used in the solution of simple substitution ciphers. Section 4 motivates the use of compression as an automatic cryptanalysis method and reviews the codelength metric calculations used in our approach which is based on the PPM, Gzip and Bzip2 methods. The pseudo-code and the full description of our method are presented in section 5. Experimental results are discussed in section 6. The final section provides the conclusion.

2 Simple Substitution Ciphers

A simple substitution cipher (also called a monoalphabetic cipher) replaces each character in the plaintext with another predetermined character to form the ciphertext (Robling Denning, 1982). Formally, let A be a plaintext alphabetic character of size n, where $A \in \{a_0, a_1, ..., a_{n-1}\}$ and C is a ciphertext alphabetic character of size n, $C \in \{f(a_0), f(a_1), ..., f(a_{n-1})\}$. Each symbol of A has a one-to-one mapping to the corresponding symbol of C, $f : A \rightarrow C$. Generally, C is a simple rearrangement of the lexical order of the symbols in A, for example:

A : A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
C : I R U S N V D W O X A P G T J Y B K E L M F C Z Q H
Then the message 'CRYPTOGRAPHY DEMO' is encoded as:

Plaintext	CRYPTOGRAPHY DEMO
Ciphertext	UKQYLJDKIYWQ SNGJ

The permutation selected represents the key.

For an alphabet with n characters, there are $n!$ possible permutations; for example, for the 26 letter English alphabet, there are 26! possible permutations or 4.03e+26. So with this large number of possibilities, finding the correct permutation through an exhaustive search is considered to be infeasible.

This type of cryptosystem is easy to implement and to use. However, it is not difficult to crack, as it does nothing to conceal the statistical properties of the language. Hence, it does not provide much security and can be easily broken by frequency distribution analysis.

By using frequency analysis of individual letters, the cryptanalyst can readily decrypt a ciphertext manually that uses this cryptosystem. This will still happen even if the character frequencies of the ciphertext is different from those of normal English text. With a few attempts and after trying some of the possibilities, the cryptanalyst will be able to find the correct substitution (Irvine, 1997; Eskicioglu & Litwin, 2001). Digram and trigram distributions provide more useful information that can also be accessed by the cryptanalyst. Many digrams could occur more frequently than some single letters while other digrams such as 'qj' rarely occur in English. Typically, different languages have different letter frequencies. So, it is possible to determine the plaintext language before starting to decrypt the ciphertext if a long enough ciphertext is given.

Despite the fact that simple substitution ciphers are not typically used in real-world encoding systems nowadays, many effective and secure modern ciphers use substitution ciphers in combination with other ciphers, for example, transposition ciphers, modular arithmetic, Boolean algebra and so on. This powerful combination is an important innovation as it results in a method that is stronger than its original components (Eskicioglu & Litwin, 2001). Although monoalphabetic ciphers are not considered secure, they are frequently used as building blocks in larger state of the art cryptographic systems. Con-

sequently, it is so important to understand the vulnerability of these simple ciphering systems, to help with building more complex ciphers (Grundlingh & Van Vuuren, 2003).

Classical ciphers generally fall into two main categories: substitution ciphers and transposition ciphers. Modern encryption systems have now superseded the classical systems; however, the most popular cryptological application and implementation for metaheuristic search research is the cryptanalysis of classical ciphers. The essential concepts of substitution ciphers and transposition ciphers are still widely used nowadays in the Advanced Encryption Standard (AES) and the International Data Encryption Algorithm (IDEA). As long as the operations and the concepts of the classical cipher systems are the basic building blocks of more secure modern ciphers, then the classical ciphers are typically the first ciphers considered in the case of investigating and examining new attacks (Garg & Sherry, 2005).

3 Previous work

Several cryptanalysis techniques have been devised for the solution of simple substitution ciphers, starting with a number of strategies for hand analysis, leading to automated cryptanalysis methods. In this section, we will concentrate on previous approaches for automated cryptanalysis.

Typically, human experts who have experience in cryptanalysis can solve a sentence-long ciphertext in a few minutes. Many hand analysis strategies have been described (Ball, 1960; Friedman, 1976; Gaines, 1956). These strategies are generally a combination of three main classes: zero order frequency analysis, a pattern matching approach and word recognition. However, none of these strategies are explicit enough to be called an algorithm. Many different solutions have been devised over the last few decades with varying degrees of success. An attempt to automate cryptanalysis by developing an expert system that generally tries capturing the knowledge of the experienced cryptanalyst was presented in (Carroll & Martin, 1986). Similarly, other systems (Schatz, 1977; Anderson, 1989a, 1989b) have firstly attempted to predict the vowels in the cryptograms.

The following summarises some of the more interesting or important automatic cryptanalysis methods and their drawbacks. The purpose is to highlight the breadth of research that has previously been applied to the problem. In particular, most of the previous attempts for automated cryptanalysis are based on two main approaches: a probabilistic approach (Peleg & Rosenfeld, 1979; Hunter & McKenzie, 1983; King & Bahler, 1992; King, 1994) and a pattern matching approach (Lucks, 1990; Hart, 1994). Automatic methods for decrypting substitution ciphers using a 'relaxation' algorithm were presented in (Peleg & Rosenfeld, 1979; Hunter & McKenzie, 1983; King & Bahler, 1992). With a 400 character long ciphertext, algorithm in (King & Bahler, 1992) was able to correctly recover 93% of the ciphertext symbols within an average execution time of 13 minutes. In the relaxation algorithm, breaking the cipher

is considered as a probabilistic problem. Each letter in the cipher is assigned probabilities representing each letter in the plaintext. By using joint letter probabilities, all ciphertext letter probabilities are generally updated in parallel. After a number of iterations, hopefully there is an improvement in the estimations that lastly lead to solve the ciphertext. In (Peleg & Rosenfeld, 1979), the joint letter probabilities which are used to update the symbol probabilities are generally based on the trigram frequencies. Just two examples were examined in this paper: a 996 character long ciphertext using a paragraph from a technical report and a 1149 character long ciphertext using Lincoln's Gettysburg Address.

On the other hand, pattern matching algorithms (Lucks, 1990; Hart, 1994) work better on short ciphertexts, but can not solve ciphertexts for which there are no words in the dictionary being used by the algorithm. They are not able to handle trivial variations, like examples with removing spaces. According to the pattern matching approach, each word in the ciphertext is structurally compared with words a dictionary. The accuracy of this approach and the time required to break the ciphertext depends on the size of the dictionary. A dictionary size of over 19,000 entries was used in (Lucks, 1990). A success rate of 60% was achieved; however, about 30% of the trials required further human intervention. Algorithm in (Hart, 1994) could provide decrypted messages for short cryptograms in a matter of seconds. Both methods in (Lucks, 1990; Hart, 1994) did not produce complete or unique solutions. That is because either there were some words that did not appear in the dictionary or multiple possibilities were deciphered. All these previous approaches (Peleg & Rosenfeld, 1979; Hart, 1994; Lucks, 1990) deal with just twenty six alphabet English letters and consider that the spaces between words are already identified or not ciphered. In contrast, our method deals with twenty seven English characters (twenty six alphabetic letters and space).

An enhanced English frequency analysis technique was introduced in (K. Lee, Teh, & Tan, 2006). It uses a combination of unigram frequencies, keyword rulings and dictionary checking. Two ciphertexts were examined, one with 9006 letters and the other with 2802 letters, and the method was able to achieve good decryption results. Dictionary-based automatic attack was demonstrated in (Olson, 2007). Twenty one cryptograms were examined and all of them were successfully solved. However, the algorithm also struggled on short cryptograms. A genetic algorithm for the cryptanalysis of simple substitution ciphers was published in (Spillman, Janssen, Nelson, & Kepner, 1993). To evaluate the quality of a key, a fitness function was used based on character unigram and digram English frequencies. There was no specific description about the test set characteristics that they used in this paper, and the exact key was not always found. A simulated annealing approach was used in (Forsyth & Safavi-Naini, 1993). The evaluation was based on using bigram statistics. With a very long ciphertext (5000 characters), the algorithm performed quite well, but it was less efficient with shorter ciphers. The use of genetic algorithms, simulated annealing and tabu search for the cryptanalysis was presented in (Clark, 1994; Garg & Sherry, 2005). Character unigram and digram frequencies were

adopted as the basis of the fitness function in (Clark, 1994) and trigram as well in (Garg & Sherry, 2005). For a cryptogram of 800 characters, 25 out of 27 key elements were recovered, and for a cryptogram of a length of 500, it was able to recover 23 keys (Clark, 1998). For a ciphertext of 200 characters, the amount of recovered keys was about 12 whereas with a 1000 character long ciphertext, about 24 keys was successfully recovered out of 27 (Garg & Sherry, 2005). Another genetic algorithm based solution (Grundlingh & Van Vuuren, 2003) was successfully implemented. Just one long ciphertext of 2519 characters was examined. The fitness function was based on character unigram and bigram analysis. Many previous works in this area were summarized in (Delman, 2004). The use of genetic algorithms was specifically explored. Attempts to extend these works were unsuccessful. This resulted in the conclusion that the genetic algorithms approach did not merit further effort, since although the traditional cryptanalysis methods require more execution time, they were easier to implement and much more successful

A fast automated attack using hill climbing was presented (Jakobsen, 1995). With a ciphertext of 100 characters, the algorithm achieved a success rate of 50%, and with a ciphertext of 400 characters in length, a success rate of 98% was reached. The time needed to cryptanalyze a cipher ranged from half a second to two seconds. HMMs (Hidden Markov Models) was used in (D.-S. Lee, 2002). It showed that the method systematically outperformed the relaxation algorithm using character bigram models. It achieved a 95% decoding rate for cryptograms of 500 characters, whereas just 80% was achieved by the relaxation approach. An attack using HMMs with multiple random restarts and hill climbing was showed in (Vobbilisetty, Di Troia, Low, Visaggio, & Stamp, 2017). A 70% accuracy rate was achieved as a result of solving a ciphertext of 200 letters in length. With ciphertexts of 300 and 400 letters, a 95% accuracy rate was achieved. A modified Markov Chain Monte Carlo (MCMC) algorithm was introduced in (Chen & Rosenthal, 2012) where ciphertexts of 1000 and 2000 characters were used. Accuracy rate of 93% against a 2000 character long ciphertext was achieved.

An attack based on using the PSO (Particle Swarm Optimization) algorithm was proposed in (Uddin & Youssef, 2006). Character unigram and bigram statistics were both used for the evaluation function. Using bigram statistics resulted in a 45% success rate for a ciphertext of 100 characters, and 95% for a cipher of 400 characters. Another automatic attack based on stochastic optimization algorithms was presented in (Hilton, 2012). Character unigram, bigram and trigram statistics were investigated. About six correct keys were recovered out of 26 for a 100 character long ciphertext, and 20 correct keys for a cryptogram with a length of 500. An exact method for solving letter-substitution ciphers using a generalized Viterbi algorithm was described (Corlett & Penn, 2010). A character-level trigram model was used to rank solutions. Texts with different sizes (1000, 3500,... 13500) were tested. With a 3500 character long ciphertext, the accuracy was 96.30% with an execution time of 38 minutes. The integration of a genetic algorithm with the metaheuristic algorithm was examined in (Luthra & Pal, 2011). For ciphertexts of

500 characters, they were able to recover 21 correct keys out of 27, and 22 correct key from ciphertexts of 1000 characters in length.

A cryptanalysis method using low-order n-gram models (unigram, bigram and trigram models for English) was presented in (Ravi & Knight, 2009). Fifty ciphers of different lengths up to 265 were examined. With ciphertexts of 64 characters, this method produced good results with 10% average errors and average execution time of approximately 76 minutes, with a 5% error rate for ciphertexts of 128 characters using the trigram model. This method produced better results than Knight et al.'s method (Knight, Nair, Rathod, & Yamada, 2006), which used the expectation-maximization algorithm (EM). A simple beam search using high order n-gram models (3-grams, 4-grams, 5-grams and 6-grams) to the problem of solving substitution ciphers was introduced in (Nuhn, Schamper, & Ney, 2013). Short cryptograms (up to 256 characters) were tested. By using a 5-gram model, ciphertexts of a length 16 were decrypted with 45% average error rate, and 60% using 6-gram models. Cryptograms of length 64 were decrypted successfully with less than 5% deciphering errors with a reported decryption time of two and a half minutes using 6-gram models.

Irvine (Irvine, 1997) has been the only researcher to have previously used a text compression method to decrypt a cipher system (in this case, simple substitution ciphers) and (Al-Kazaz, Irvine, & Teahan, 2016) used a compression method to break transposition ciphers. Irvine used a variation of PPM modelling system (combined with simulated annealing) for the automatic solution of simple substitution ciphers, with good results achieved compared to other methods with 60% of ciphertexts solved without any errors, and 83% with less than four errors.

In this paper, we investigate more deeply the use of PPM compression method by proposing a new variation for tackling the problem of the automatic decryption of simple substitution ciphers. However, our approach uses a different search algorithm and a new modified version of PPM, which achieves a high success rate achieved close to 100%. The use of other compression schemes (Gzip and Bzip2) are also examined in this paper. Our evaluation also analyses the decryption of different ciphertext lengths, including very short cryptograms with just 20 characters, whereas the cryptograms used in most of the previous research were not less than 100 characters long or usually required quite a long time to execute with a relatively high error rate. In our paper, we will present different PPM compression variants and investigate which variant is the most effective when applied to the problem of automatically decrypting simple substitution ciphertexts.

4 Automated Cryptanalysis Using Compression

The ciphertext only cryptanalysis of simple ciphersystems heavily depends on the statistical features of the source language, and it is not a trivial issue to get computers performing this analysis. Several previously published crypt-

analysis methods can not run without human intervention or they assume at least known plaintext because of the difficulty of quickly recognizing a correct decryption in a ciphertext only cryptanalysis (Irvine, 1997; Schneier, 1993; Wiener, 1993).

Having a computer model that is able to predict and model natural language as well as a human is critical for cryptology (Teahan, 1998). Teahan demonstrated that the PPM modeling system has the ability to predict text in a way that is close to achieving human performance level (Teahan & Cleary, 1996). The essential idea of our technique uses PPM for calculating the compression 'codelength' value of each possible permutation (which is used to measure the amount of information in each (Irvine, 1997)). Permutations with smaller codelength values help to determine better decryptions (Al-Kazaz et al., 2016). We present how to use this to automatically and easily recognise the valid solution in a ciphertext only cryptanalysis against simple substitution systems. Also, further investigations on different variants of PPM compression method are performed in this paper. Other compression methods (Gzip and Bzip2) as a basis method for calculating the codelength metric are also tried. This is to ascertain the most effective method to use to automatically break simple substitution cryptosystems.

4.1 PPM Compression Codelength Metric

Prediction by Partial Matching (PPM) has set the performance standard in lossless compression of text throughout the past three decades (Teahan, 1998). The initial method was first published by Cleary & Witten (Cleary & Witten, 1984). It is an adaptive statistical coding approach whose main idea is based on using the last few characters in the input to predict the upcoming one using a Markov-based method. Models that set their predictions on a few prior symbols are termed finite-context models of order k, where k denotes the number of previous symbols used. The order of the model represents the maximum context length used to predict the next symbol.

Prediction probabilities for each model are calculated from all characters or symbols that have followed every subsequence observed from 1 to length k, and from the number of times that each character has occurred. From each model, the probabilities associated with each symbol or character that has followed the preceding k characters (in the past) are estimated to predict the upcoming character. Prediction by Partial Matching modelling systems have been found to be very efficient at compressing English text (Teahan & Cleary, 1996).

Usually, each character or symbol in the model will be encoded by using arithmetic coding depending on their associated probabilities (Witten, Neal, & Cleary, 1987). According to the PPM scheme, it begins from the highest context order k. Where this context predicts the upcoming symbol, then each symbol will be encoded according to the probability distribution associated with it. In the case that a previously unseen character or symbol is observed in

this context, the context model will not be able to encode this character and an alternative solution must be adopted. An "escape" symbol whose probability is predicted by the PPM compression method, will be transmitted to signal the encoder to switch to the next context model of order $k-1$. The operation will continue until it reaches the order in the compression model in which the upcoming character is not novel. If needed, when a completely novel symbol is encountered, the method will escape down to the $k = -1$ (order -1) default model, which is the lowest order context in the model where all symbols will be encoded with equal probability of $\frac{1}{|A|}$, where A denotes the size of the alphabet.

By using this escape mechanism, different order models are effectively blended in order to 'smooth' the probability estimates. Many previous results have shown that typically no further improvement can be achieved in the compression results by using increasing context lengths greater than five for English texts (Cleary & Witten, 1984; Cleary, Teahan, & Witten, 1995; Teahan, 1998). Further improvements can be achieved when escaping has occurred by excluding all symbols already predicted by higher order contexts since these symbols would have already been encoded using a higher order context if they had occurred. This mechanism is called "full exclusion".

Moffat (Moffat, 1990) devised another simple mechanism that improve results further which is called "update exclusion". This mechanism is based on how the symbol counts for each context model are updated. When encoding with update exclusions, the predicted symbol count is incremented only if it is not already predicted by any higher order context. This means that the counts are updated only for the higher order contexts that are actually used to predict it. Thus, the counts reflect better which symbols are likely to be excluded by the higher-order contexts. This mechanism typically improves the compression rate by 2% as Bell, Cleary and Witten stated (Bell et al., 1990). On the other hand, when encoding without update exclusions, all the counts for all orders of the model are updated. The counts are incremented even if they are already predicted by a higher order context. The use of both of these mechanisms are investigated in our paper: PPM without update exclusions; and PPM with update exclusions (standard PPM).

Various variations of the PPM compression scheme have been invented, depending on the methods that have been proposed for calculating symbol probabilities. They differ by the escape method used for each. For example, PPMC uses escape method C, and PPMD uses escape method D. Also, the maximum order of the context models may be included when the variant is described in the literature; for example, PPMD4 refers to a fixed order 4 PPM model using escape method D.

The PPMC variant was developed by Moffat (Moffat, 1990) and has become the benchmark version. The probability of this method (method C) is based on using the number of symbols that have occurred before, called the number of types:

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

where e represents the probability of the escape symbol, $p(s)$ denotes the probability for symbol s, $c(s)$ is the number of times the context was followed by the symbol s , n is the total number of times that the current context has occurred and t denotes the total number of types.

PPMD is another improved variant. It was first developed by Howard (Howard, 1993). PPMD usually shows better performance than the other PPM compression variants: PPMA, PPMB and PPMC. This variant is similar to PPMC except that the probability of the new symbol is estimated differently. The new symbol's treatment becomes more consistent (Howard & Vitter, 1992) by adding $\frac{1}{2}$ to both the symbol and escape counts:

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

To illustrate the process of the PPM method, Table 1 presents the state of the PPMC and PPMD models where $k = 2, 1, 0$ and -1 after the input string 'stressless' has been processed. For illustration purposes for this example, the highest context order is for $k = 2$. If the next symbol or character is estimated successfully by the modeling context, the probability p will be used to encode it, while c denotes the occurrence counts. Referring to the example, if the input string 'stressless' is followed by the character 'l', the probability of the prediction 'ss'→'l' in order 2 (which is $\frac{1}{2}$) would be used to encode it requiring only one bit as a result ($-\log_2 \frac{1}{2} = 1$).

Assume instead that the character 't' follows the string 'stressless'. As the order 2 model does not predict this character, the escape probability of $\frac{1}{2}$ will be encoded for this order, and the encoder will move from the order two model ($k = 2$ in the first column) down to the order one model ($k = 1$ in the second column). In this context, 's'→'t' does predict the character 't', with probability $\frac{1}{7}$. So, the total probability needed to encode the 't' character is $\frac{1}{2} \times \frac{1}{7}$, or 3.8 bits. Actually in this context, a more accurate probability estimation is gained by noticing that the character 't' cannot be encoded using this context, as if it did it would have been already encoded by the order two context. Therefore, we can exclude the already predicted symbols and this is what is termed the full exclusion mechanism, which corrects the probability for his context to be $\frac{1}{6}$. Finally, the total probability will be $\frac{1}{2} \times \frac{1}{6}$ with 3.6 bits required for the compression codelength.

If the next character, however, has never been seen before, such as 'm', the escape will be repeated down through the models to the default order -1 context ($k = -1$), where all symbols or characters have equal probabilities with $\frac{1}{|A|}$ where A refers to the size of the alphabet. Supposing that the alphabet size is 256 for the English language encoded using 8-bit ASCII. Consequently, the total probability for encoding the 'm' character will be $\frac{1}{2} \times \frac{3}{7} \times \frac{5}{15} \times \frac{1}{256}$, or 11.8 bits when encoded using arithmetic coding. The full exclusion mechanism can be used to get a more accurate probability estimation, which will exclude characters already appearing in higher orders. When this is applied, the new probability for the 'm' character will be equal to $\frac{1}{2} \times \frac{3}{6} \times \frac{5}{11} \times \frac{1}{251}$, with the total codelength of 11.1 bits.

Table 1: PPMC and PPMD models after processing the string "stressless" with maximum order of 2.

Order k=2					Order k=1					Order k=0				Order k=-1			
Prediction			c	p	Prediction			c	p	Prediction		c	p	Prediction		c	p
PPMC																	
st	→	r	1	$\frac{1}{2}$	s	→	t	1	$\frac{1}{7}$	→	s	5	$\frac{5}{15}$	→	A	1	$\frac{1}{\lvert A\rvert}$
	→	Esc	1	$\frac{1}{2}$		→	s	2	$\frac{2}{7}$	→	t	1	$\frac{1}{15}$				
						→	l	1	$\frac{1}{7}$	→	r	1	$\frac{1}{15}$				
tr	→	e	1	$\frac{1}{2}$		→	Esc	3	$\frac{3}{7}$	→	e	2	$\frac{2}{15}$				
	→	Esc	1	$\frac{1}{2}$						→	l	1	$\frac{1}{15}$				
					t	→	r	1	$\frac{1}{2}$	→	Esc	5	$\frac{5}{15}$				
re	→	s	1	$\frac{1}{2}$		→	Esc	1	$\frac{1}{2}$								
	→	Esc	1	$\frac{1}{2}$													
					r	→	e	1	$\frac{1}{2}$								
es	→	s	2	$\frac{2}{3}$		→	Esc	1	$\frac{1}{2}$								
	→	Esc	1	$\frac{1}{3}$													
					e	→	s	2	$\frac{2}{3}$								
ss	→	l	1	$\frac{1}{2}$		→	Esc	1	$\frac{1}{3}$								
	→	Esc	1	$\frac{1}{2}$													
					l	→	e	1	$\frac{1}{2}$								
sl	→	e	1	$\frac{1}{2}$		→	Esc	1	$\frac{1}{2}$								
	→	Esc	1	$\frac{1}{2}$													
le	→	s	1	$\frac{1}{2}$													
	→	Esc	1	$\frac{1}{2}$													
PPMD																	
st	→	r	1	$\frac{1}{2}$	s	→	t	1	$\frac{1}{8}$	→	s	5	$\frac{9}{20}$	→	A	1	$\frac{1}{\lvert A\rvert}$
	→	Esc	1	$\frac{1}{2}$		→	s	2	$\frac{3}{8}$	→	t	1	$\frac{1}{20}$				
						→	l	1	$\frac{1}{8}$	→	r	1	$\frac{1}{20}$				
tr	→	e	1	$\frac{1}{2}$		→	Esc	3	$\frac{3}{8}$	→	e	2	$\frac{3}{20}$				
	→	Esc	1	$\frac{1}{2}$						→	l	1	$\frac{1}{20}$				
					t	→	r	1	$\frac{1}{2}$	→	Esc	5	$\frac{5}{20}$				
re	→	s	1	$\frac{1}{2}$		→	Esc	1	$\frac{1}{2}$								
	→	Esc	1	$\frac{1}{2}$													
					r	→	e	1	$\frac{1}{2}$								
es	→	s	2	$\frac{3}{4}$		→	Esc	1	$\frac{1}{2}$								
	→	Esc	1	$\frac{1}{4}$													
					e	→	s	2	$\frac{3}{4}$								
ss	→	l	1	$\frac{1}{2}$		→	Esc	1	$\frac{1}{4}$								
	→	Esc	1	$\frac{1}{2}$													
					l	→	e	1	$\frac{1}{2}$								
sl	→	e	1	$\frac{1}{2}$		→	Esc	1	$\frac{1}{2}$								
	→	Esc	1	$\frac{1}{2}$													
le	→	s	1	$\frac{1}{2}$													
	→	Esc	1	$\frac{1}{2}$													

The same procedure applies to the second part of the table (table 1) when using the PPMD compression variant but using the modified symbol and escape counts as discussed above. For example, the probability of the prediction 'es'→'s' in order 2 for PPMD how now changed to $\frac{3}{4}$ from $\frac{2}{3}$ and the context's escape probability has changed to $\frac{1}{4}$ from $\frac{1}{3}$ because of the different way the probabilities are now being calculated.

The PPM method can also be applied to streams of word-based symbols as opposed to character streams. Several word-based systems have been proposed (Horspool & Cormack, 1992; Bentley, Sleator, Tarjan, & Wei, 1986; Moffat, 1989; Teahan, 1998). The word-based approach typically provides faster compression compared to the character-based models as fewer symbols are being encoded. Similar to the previously described character-based models, word-based models use the preceding words to predict the next word using a similar PPM encoding mechanism with escapes to lower-order models. A number of methods for estimating the escape probability for the word-based models have been explained in (Witten & Bell, 1991) and (Teahan, 1998). An escape to an order -1 word context signifies that the word needs to be encoded character by character. In this case, each symbol or character in the word (even the space character that marks the end of each word) is separately encoded. Once a word has been encoded once, a word symbol associated with that word that identifies it uniquely can now be encoded instead. Essentially, the word symbols are added to an expanding alphabet of word symbols as the new words are encountered.

Previous experiments have shown that the performance of the word-based scheme degrades with higher orders. The performance of the order 2 word bigram models are slightly worse than the order 1 word unigram models. Order 3 trigram word models and higher follow the same trend. Experimental results show that for the English language, the performance of the word-based schemes slightly outperform the character-based ones. However, the character-based models are more economical in terms of memory space and are more easily applied to various applications in natural language processing which require the correction of character sequences such as OCR, spelling correction and cryptology (Teahan, 1998).

Table 2: Models for predicting character and word streams (Teahan, 1998).

C|C^5 Model	**W|W Model**
$p(c_i\|c_{i-1}c_{i-2}c_{i-3}c_{i-4}c_{i-5})$	$p(w_i\|w_{i-1})$
$\hookrightarrow p(c_i\|c_{i-1}c_{i-2}c_{i-3}c_{i-4})$	$\hookrightarrow p(w_i\|)$
$\hookrightarrow p(c_i\|c_{i-1}c_{i-2}c_{i-3})$	$\hookrightarrow$ Character model
$\hookrightarrow p(c_i\|c_{i-1}c_{i-2})$	
$\hookrightarrow p(c_i\|c_{i-1})$	
$\hookrightarrow p(c_i\|)$	
$\hookrightarrow p_{eq}(c_i\|)$	

In our work, we make use of two PPM based models, one character-based and the other word-based, which provide effective results in terms of compression rate and lead to significant improvements both in terms of compression rate as published in previous experiments (Teahan, Wu, & Liu, 2014) and in terms of the reduction in the number of decryption errors as per the experimental results discussed below. The first model, which is labeled $C|C^5$ in Table 2, represents an order 5 PPM character model (order 5 and order 4 models are used in our experiments) where the predictions are based on the stream of character symbols. So, the probability of S (where S is a sequence of length m characters) is given by:

$$p(S) = \prod_{i=1}^{m} p'(c_i|c_{i-1}c_{i-2}c_{i-3}c_{i-4}c_{i-4}c_{i-5})$$

where p' is the probabilities estimated by the PPM model. (Note: the symbol $\hookrightarrow$ in the table represents an escape).

The second model, which is labeled $W|W$, represents an order 1 PPM word bigram model. The predictions of this model are based on the stream of word symbols as shown in the next formula:

$$p(S) = \prod_{i=1}^{m} p'(w_i|w_{i-1})$$

where p is the probability of S, S is a sequence of m words and p' is the probabilities estimated by the word model (Teahan, 1998).

The fundamental concept of our cryptanalysis method is based on using a PPM compression model to calculate codelength values of each possible permutation. The 'codelength' of a permutation for a cryptogram is the length of the compressed cryptogram, in bits, when it has been compressed using the PPM language model. The smaller the codelength value, the more closely the ciphertext resembles the text used to train the language model. This metric has been found to be effective for measuring the quality of the different decryptions, and therefore can be used for automatically recognising correct solutions (Al-Kazaz et al., 2016).

One of the problems of using an adaptive compression method such as PPM is that at the beginning the models are empty and there is not sufficient data to effectively compress the texts resulting in the different permutations producing similar codelength values. To overcome this problem, a simple expedient is to prime the models using representative training texts. In our experiments described below, we use nineteen novels and the Brown corpus converted to 27 character English to train our models (by case-folding to lower case and collapsing non-alphabetic sequences to a single space). We also use static models, unlike standard PPM—that is, once the models have been primed using the training texts, they are not further updated when processing the ciphertexts.

4.2 Calculating Compression Codelengths Using Gzip and Bzip2

We have also investigated alternative compression methods for performing the codelength calculations—Gzip and Bzip2. The essential reason for experimenting with other compression schemes in our paper is to find out which is the most efficient scheme that can be used in the automatic solution of simple substitution cryptosystems using a compression based technique.

Gzip is one of the most important compression methods that was written by Jean-Loup Gailly & Mark Adler, and created for the GNU project. Gzip is now a common lossless compression scheme on the Internet and the Unix operating system. It uses a dictionary based approach using Lempel-Ziv coding (Gzip, 2012) while PPM is a finite-context statistical approach. Bzip2 is another well-known compression scheme that was written by Julian Seward (Bzip2, 2016). It is a lossless compression method which uses a block sorting approach (the Burrows-Wheeler block sorting compression algorithm). The compression performance of this method (Bzip2) is usually better than the Gzip; however, its speed is slower. It approaches the performance of the best compression techniques such as those produced by PPM (Bzip2, 2016).

In this paper, the calculation of the codelength metric for these two compression methods (Gzip and Bzip2) will be based on using a relative entropy calculation which allows us to use 'off-the-shelf' software without the need to re-implement the methods themselves. The codelength metric can be calculated using the relative entropy technique by the following formula (Al-Kazaz et al., 2016):

$$h_t = h_{T+t} - h_T$$

where T denotes the training text (which will usually be large in size), t denotes the testing text, and h_T refers to the size of the compressed file T. Essentially, the codelength for a particular compression scheme is calculated as being the difference between the compressed size of some large training text with testing text concatenated after it compared to the compressed size of just the training text by itself.

5 Our Method

A full description of our new method for the automated solution of simple substitution ciphers will be presented in this section. As stated, the fundamental idea of our method is based on using a compression scheme to calculate the codelength value to use as a metric for ranking the quality of each possible permutation (Al-Kazaz et al., 2016). PPMD, PPMC, Gzip and Bzip2 are the compression schemes used in our experiments for the codelength calculations. The main idea of our approach is based on trying to break a cryptogram by essentially substituting one letter at a time throughout the text, starting with the most frequent. Then one of the compression methods is used to compute the codelength value used for automatically scoring the possibilities.

The pseudo-code for our approach is presented in Algorithm 1. At the start (see line 1 in the pseudo code), we remove all non-alphabetic characters from the ciphertext and keep only letters and spaces (i.e. our approach processes only 27 characters). However, the methods presented here can be adapted to arbitrary alphabets (whether or not spaces are included). After that, all the remaining characters in the ciphertext are examined in order to determine frequencies, and arranged from the most frequent character to the least frequent (see line 2). The search is initialised by setting each character in the ciphertext to a special symbol (a full-stop) that is not an English alphabetic character or space (see line 3). Then by replacing one ciphertext character *cc* at a time (see line 6), it simply tries each unused character in turn as a candidate for *cc* (see lines 9 to 11). The compression codelength is calculated for each possibility using PPM, Gzip or Bzip2 (see line 12). The ciphertexts are then ranked using a priority queue according to the codelength values (see lines 13 to 17). As we find permutations with smaller compression codelength values, we are closer to finding the valid decrypt. We keep at most only the 500 best results at each stage in the priority queue. The maximum size of the priority queue provides a means for trading off between greater speed (when the size of the queue is reduced) and less decryption errors (when a greater size is used). Experimental results show (see below) that a size of 500 provides a good compromise.

Our method builds up the solution incrementally, replacing one crypto character, *cc*, at a time, dealing with the most frequent *cc* first. So starting with a new cryptogram, it picks the most frequent symbol (say x) in the cryptogram (most likely this corresponds to space or perhaps the letter 'e'). It tries substituting x with one of the English alphabetic characters 'a' to 'z' or space. (Note: At this stage, all of these will be tried since the size of the alphabet, 27, is less than the maximum size of the priority queue i.e. 500). Then it picks the next most common crypto symbol to substitute, say y, for each of the previous 27 possibilities, substituting y with one of the English alphabetic characters or space (but excluding any already substituted characters). This gives 702 ($= 27 \times 26$) possibilities, so at this stage solutions start getting discarded if only a maximum of 500 possibilities are kept in the priority queue. This is repeated for each remaining character from the most frequent characters down to the least frequent character.

In order to get further improvements in our results, a word-based PPM compression system is applied to the output produced from Algorithm 1. The pseudo-code for this is provided in Algorithm 2. The codelength values are re-calculated using the word-based model, then these are stored in the new queue as they provide a potentially more accurate estimate of their quality.

Two variants of the PPM modelling system, PPMD and PPMC models have been used in our method. Also two forms of these schemes are examined, one with update exclusions (i.e the standard PPMD or PPMC) (Teahan, 1998) and one without update exclusions. Both of these variants are further investigated with a new variation of the PPM algorithm where a specific high codelength value is assigned to all contexts for which an escape down to an order -1 context has occurred when the symbol being predicted has not al-

Algorithm 1: Pseudo code of our automatic cryptanalysis method.

```
   Input  : ciphertext
   Output: decrypted text
 1 REMOVE all non-alphabet characters -except space- from the ciphertext;
 2 EXAMINE the ciphertext to create a sorted list of the zeroth order frequencies for
   the alphabet;
 3 REPLACE the characters in the ciphertext with the special symbol '.';
 4 INITIALISE Q1 (priority queue) with modified ciphertext;
 5 maximum-size of Q2 (priority queue) ← 500;
 6 for each crypto character 'cc' in the zeroth order frequent characters (starting from
   the most to the least frequent characters) do
 7 |   Q2 ← empty;
 8 |   for each ciphertext in Q1 do
 9 |   |   for each alphabetic and space characters 'ac' do
10 |   |   |   if 'ac' is not used before as a replacement of the previous crypto
   |   |   |   characters then
11 |   |   |   |   REPLACE each crypto character 'cc' in the ciphertext with the
   |   |   |   |   unused character 'ac' as a candidate for 'cc';
12 |   |   |   |   CALCULATE codelength value of the ciphertext using the PPM,
   |   |   |   |   Gzip or Bzip2 compression model;
13 |   |   |   |   if the size of the priority queue Q2 < 500 then
14 |   |   |   |   |   ADD the ciphertext to Q2;
15 |   |   |   |   else if the codelength value of the last element in Q2 > codelength
   |   |   |   |   value of the current ciphertext then
16 |   |   |   |   |   REMOVE the last element in Q2;
17 |   |   |   |   |   ADD the ciphertext to Q2;
18 |   |   |   end
19 |   |   end
20 |   end
21 |   REPLACE Q1 with Q2;
22 end
23 return front of priority Q1 (the 'decrypted text');
24 return first ten results in Q1 (the ten best results)
```

Algorithm 2: Pseudo code of word-based ranking algorithm.

```
   Input  : the priority queue Q1 output from Algorithm 1
   Output: decrypted text
 1 maximum-size of Q3 (priority queue) ← 500;
 2 for each text in Q1 do
 3 |   CALCULATE compression codelength value of the text using the PPM
   |   word-based compression method;
 4 |   STORE the text in the priority queue Q3;
 5 end
 6 return front of priority Q3 (the 'decrypted text');
 7 return first ten results in Q3 (the ten best results)
```

ready occurred in any higher order context. The idea behind assigning a high codelength value for these order -1 contexts is to penalise these cases during the search as they provide strong evidence of being of lower predictive quality. During the execution of Algorithm 1, these contexts occur frequently at the start since all the characters in the ciphertexts are initialised to the special

symbol (full-stop) which is a symbol not found in the 27 character alphabet that is used for the training text. In normal PPM coding when an order -1 context occurs, the probability can be estimated as $\frac{1}{N}$ where N is the size of the text already processed. We are also using static PPM models which means that the probability of previously unseen characters such as the special full-stop character does not subsequently change as the ciphertext is being processed. Therefore, we can simply used a fixed codelength value for all the order -1 contexts which is equal to $-\log_2 \frac{1}{N}$. (The size of the training data we use in our experiments is $N = 21824832$ so the specific codelength value we assign for order -1 contexts is 24.38.)

To organize and clarify our results, our experiments are divided into different variants as presented in Table 3. For the PPM-based variants, both order 4 and order 5 models are used in our experiments as discussed below. Experiments with a full range of variations have been conducted (PPMC, or PPMD, with and without update exclusions, with and without explicit order -1 codelengths; Gzip; and Bzip2). However, for the purposes of this paper, only the results for the seven variations in the table are shown in order to illustrate either the best performing schemes or to illustrate interesting results for comparison.

According to the first experimental variant in Table 3, Variant A, PPMD5 without update exclusions is applied to compute the codelength values. In Variant B, a new variation of the PPM method is used which is PPMD without update exclusions with specific order -1 codelength values. Both order 4 and order 5 PPMD models are examined. The standard version of the PPMD5 compression method with update exclusions is used for Variant C. The fourth variant, Variant D, is the standard order 5 PPMD model but with specific order -1 codelength values being used instead of the standard PPM order -1 encoding method. Another new version of PPM is used for variant E, PPMC without update exclusion but using the order -1 codelength method. Both order 4 and order 5 PPMC models are examined. For the last two variants, variants F and G, we examine the effectiveness of using the other compression methods Gzip and Bzip2 for computing the codelength metric using the relative entropy calculation as discussed above.

Table 3: Compression method variants.

Name	**Compression method**
Variant A	PPMD without update exclusions
Variant B	PPMD without update exclusions with the same specific codelength value assigned to all order -1 context predictions
Variant C	Standard PPMD (i.e with update exclusions)
Variant D	Standard PPMD with the same specific codelength value assigned to all order -1 context predictions
Variant E	PPMC without update exclusions with the same specific codelength value assigned to all order -1 context predictions
Variant F	Bzip2
Variant G	Gzip

6 Experimental Results

In this section, we discuss experimental results for Variants A to G as described in Table 3. As stated, nineteen novels and the Brown corpus were used to train these models using 27 character English text. After this training step and during cryptanalysis, the PPMC and PPMD models remain static. A corpus of 110 different ciphertexts chosen at random form many different resources (newspapers and magazines) as testing texts are used. The lengths of these ciphers range from 20 characters to almost 300 characters. Table 4a and table 4b present samples of decryption.

Table 4a shows the output sample showing the execution of the algorithm for the ciphertext '`zjyvgelyzjgqwyzjykoaakbyjgaejvb`' including intermediate results. Compression codelengths with the lowest five results are listed in bits. Table 4b presents the ten best solutions as a result of our method of the automatic ciphertext only attack of the simple substitution cipher when using the new version of the PPMD modelling system (labelled as Variant B). In this case, text with the shortest codelength value (the best solution), represents the decrypted text. According to the example (in Table 4b), the best solution was '`so much sound so little outcome`', which has the shortest compression codelength value 63.884 and is the valid decrypt.

To encrypt the plaintext (original text), a random key is generated for each run. Afterwards, the attack is performed on the cryptogram. Various ciphertexts with different lengths (even very short) have been examined in our experiments. We experimented with 110 different ciphertexts. In order to measure the success and the accuracy of our automatic cryptanalysis algorithms, alphabetic substitution errors are counted. The results of our experiments showed that only when using the new PPM variants (Variants B and E), as a method of calculating the codelength values, were almost all the ciphertexts decrypted successfully. In contrast, the other PPM variants produced a significantly greater number of errors. The same was repeated when using the Gzip and Bzip2 algorithms in the last two variants (G and F). Example output produced by the different variants is shown in Table 5.

For variant A, Figure 1a presents the number of errors for each testing cryptogram as a result of our automatic cryptanalysis method using PPMD without update exclusions. Clearly, we can see that the number of errors for most tested cryptograms are high. In this case, just one cryptogram is solved with no errors, and only four examples are found to have ten errors or less. The results show that over 75% of the decrypted cryptograms have more than ten errors and 20% have greater than twenty errors.

For variant B, both order 4 and order 5 PPMD models are examined. For the order 4 model, the results show that 81% of the ciphertexts are correctly solved with no errors (that is, the best solution with minimum codelength value is the correct solution). About 19% of the examples are decrypted with just three errors or less. For the order 5 model, the results show better performance with over 87% of the cryptograms correctly solved without any errors, also 12% of the ciphers are decrypted with just one or two errors, and only one example

Table 4a: Example output.

```
Ciphertext: zjyvgelyzjgqwyzjykoaakbyjgaejvb
Processing the 1st most frequent character 'j';
Buffer length is: 27.
Codelength = 636.168: . ....... ..... ........ ... ..
Codelength = 640.479: .e.......e.....e........e...e..
Codelength = 642.719: .t.......t.....t........t...t..
Codelength = 643.671: .a.......a.....a........a...a..
Codelength = 643.956: .o.......o.....o........o...o..
Processing the 2nd most frequent character 'y';
Buffer length is: 500.
Codelength = 532.282: . t....t. ...t. t......t ... ..
Codelength = 532.948: .e .... .e... .e ...... e...e..
Codelength = 532.951: .t .... .t... .t ...... t...t..
Codelength = 534.603: .s .... .s... .s ...... s...s..
Codelength = 535.227: . s....s. ...s. s......s ... ..
Processing the 3rd most frequent character 'g';
Buffer length is: 500.
Codelength = 467.465: .t .h.. .th.. .t ...... th..t..
Codelength = 470.531: . t.a..t. a..t. t......t a.. ..
Codelength = 470.643: .t .o.. .to.. .t ...... to..t..
Codelength = 471.204: .e .n.. .en.. .e ...... en..e..
Codelength = 471.567: . t.i..t. i..t. t......t i.. ..
Processing the 4th most frequent character 'a';
Buffer length is: 500.
Codelength = 404.402: .t .h.. .th.. .t ..ee.. the.t..
Codelength = 408.380: .ti.h..i.th..i.ti..  ..ith .t..
Codelength = 408.697: .ht.e..t.he..t.ht..  ..the .h..
Codelength = 408.945: .er.d..r.ed..r.er..  ..red .e..
Codelength = 409.339: .t .h.. .th.. .t ..oo.. tho.t..
...
Processing the 8th most frequent character 'b';
Buffer length is: 500.
Codelength = 216.382: hetor..ther..theti.  inter .eon
Codelength = 217.041: e sai..se i..se so.ttons it. an
Codelength = 217.069: so mu.. sou.. so l.ttle out.ome
Codelength = 217.460: t sai..st i..st se.nners in. ar
Codelength = 217.894: to un.. ton.. to d.eeds one.ous
Processing the 9th most frequent character 'e';
Buffer length is: 500.
Codelength = 167.839: so muc. sou.. so l.ttle outcome
Codelength = 172.087: t shad.st a..st si.nnies and he
Codelength = 173.728: t spad.st a..st se.nners and pr
Codelength = 174.211: he san. hea.. he o.rrot earnest
Codelength = 176.220: he rat. hea.. he i.ssin eastern
Processing the 10th most frequent character 'q';
Buffer length is: 500.
Codelength = 145.242: so muc. soun. so l.ttle outcome
Codelength = 147.082: so muc. sour. so l.ttle outcome
Codelength = 147.961: so muc. soug. so l.ttle outcome
Codelength = 150.597: so muc. soup. so l.ttle outcome
Codelength = 151.714: t shad.st al.st si.nnies and he
...
Processing the 13th most frequent character 'w';
Buffer length is: 500.
Codelength = 63.884: so much sound so little outcome
Codelength = 72.105: so much sourd so little outcome
Codelength = 73.876: so much soung so little outcome
Codelength = 75.474: so muck sound so little outcome
Codelength = 75.519: so much sound so lyttle outcome
```

Table 4b: Example output (ten best solutions).

Ten best solutions	
63.884	so much sound so little outcome
72.105	so much sourd so little outcome
73.876	so much soung so little outcome
75.474	so muck sound so little outcome
75.519	so much sound so lyttle outcome
76.426	so much sound so lattle outcome
76.677	so mucy sound so little outcome
79.165	so muck sough so little outcome
79.350	so much sourg so little outcome
79.896	so much souvy so little outcome

Table 5: Sample of solved cryptograms by different variants.

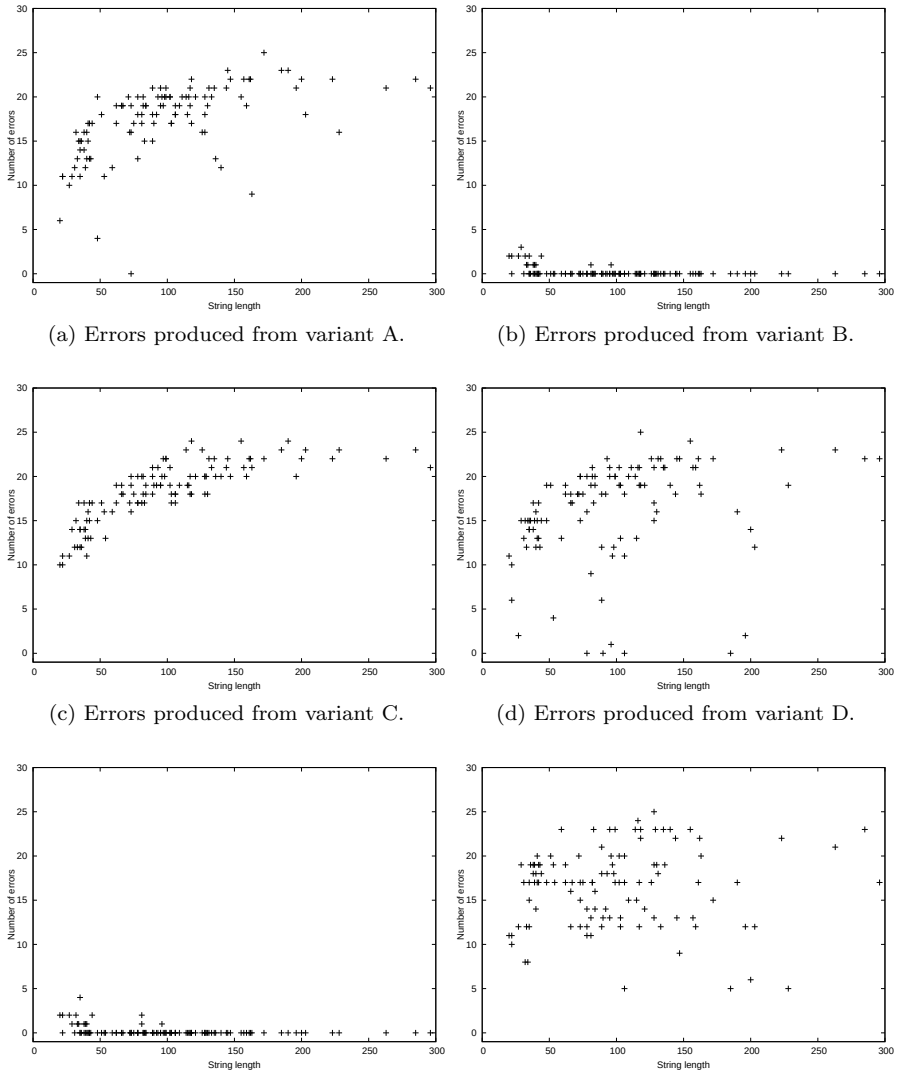
Variants	**Number of errors**	**Decrypted message**
Variant A	17	ladylamandems dejaegonzalpsteyemainerosecine scheynerosle domickepoleontre oetonw
Variant B	0	retirement must be wonderful i mean you can suck in your stomach for only so long
Variant C	20	nod nineosiukosqnsprean tuysdsincesbrushceskuhlsdesbru skorichmstr sreybskrsyrex
Variant D	19	spaxsprpia roma hp cviupsloe x rpyi tvo nyi monk xi tvos mavrynw lvs viet mv evid
Variant E	2	retirement must be wonderful i mean you can such in your stomack for only so long
Variant F	21	myr myaywruatiruzyugvwhymotbu uaydwuxvtuedwuitenu wuxvtmuirvadekuovmuvwbxuivubvwc
Variant G	11	detadement mrst he pongedbrl a mein for cin srck an ford stomicy bod onlf so lonj

had three errors as shown in Figure 1b. Moreover, in almost all these examples, the correct solution can also be found within the ten best solutions. It is clear that the number of errors for this variant is much lower than other variants. All the cryptograms of length longer than 100 are successfully solved without any errors and in almost all cases it found a correct solution.

Figure 1c illustrates the number of errors for each cryptogram for variant C. We can see that almost all the decryption errors are more than ten, with just two examples being solved with ten errors. Over 71% of the cryptograms are decrypted with greater than ten errors, and over 26% of the examples have more than twenty errors. None of the examples produced no errors.

In variant D, the results show that most of the errors are greater than 10 with just four of the ciphertexts solved without errors. Over 58% of the decrypted cryptograms have ten errors or more, and approximately 32% have twenty errors or more with just 6% having less than ten errors.

Variant E produces slightly worse results than variant B, with 80% of examples having been successfully solved without any errors and 20% decrypted



(a) Errors produced from variant A.

(b) Errors produced from variant B.

(c) Errors produced from variant C.

(d) Errors produced from variant D.

(e) Errors produced from variant E.

(f) Errors produced from variant G.

Fig. 1: Errors produced from different variants

with four errors or less when using the order 4 model. The order 5 PPMC model produces slightly better results. When we examine only the best solution, 86% of examples are successfully decrypted without any errors, and 13% solved with one or two errors, and one of the decrypted cryptogram having four errors. When we examine the ten best solutions, almost all the examples have no errors. Figure 1e presents the results of variant E using the order 5 model.

Table 6a: Average errors for each different variant when examining the best solution.

Variants	**A**	**B**	**C**	**D**	**E**	**F**	**G**
Average errors	17.37	0.20	18.29	16.21	0.22	–	16.59

Table 6b: Average errors for each different variant when examining the ten best solutions.

Variants	**A**	**B**	**C**	**D**	**E**	**F**	**G**
Average errors	17.45	1.38	18.31	16.28	1.4	–	16.83

The number of errors produced form variant G is shown in Figure 1f. It is clear that the number of errors for each decrypted ciphertext is much higher, with most of the errors being greater than 15. Also none of the cryptograms offered no errors and just seven cryptograms were decrypted with the number of errors being less than 10.

Results regarding the average number of automatic cryptanalysis errors for the 110 cryptograms we tested with each of the variants are presented in Tables 6a and 6b. Table 6a presents the average number of errors when just examining the best solution, and Table 6b shows the average errors for the ten best solutions. Clearly, the best performing model overall is PPMD5 without update exclusions using the order -1 correction model (Variant B). However, Variant E, which used PPMC5 without update exclusions along with order -1 correction, presents excellent results as well. On the other hand, the other Variants A, C and D produce poor results. Interestingly, the PPM without update exclusions method, which typically shows slightly worse performance at the compression task, shows better performance at decryption here.

We also experimented with using our relative entropy calculation method using Bzip2 for Variant F. But due to the block-sorting nature of the Burrows-Wheeler algorithm, the calculation of some of the relative entropy codelengths ended up being negative. Thus these results could not provide us with a complete picture with regards to the average number of errors. However, none of the positive results did show any success, with a quite high number of errors. The average number of errors produced for Variant G (using Gzip2) is presented at the last column in the table. The results show that the Gzip compression scheme is not an effective way of recognising the valid decryptions as it also results in a high number of errors. In addition, the time that is needed to break the ciphers (by using Gzip and Bzip2) using the relative entropy calculation is considerably longer (as it involves repeatedly compressing the training text), and thus also makes the use of these methods completely impractical.

Like other cryptanalysis approaches, very short cryptograms can often not always be solved correctly, even with a better model of English as Irvine claimed in his thesis (Irvine, 1997). This is because these cryptograms are

inherently ambiguous as a simple substitution unicity distance is about 26, according to this equation:

$$U = \frac{H(k)}{R} = \frac{\log 27!}{\log 27 - 1.2} = 26.2$$

where U represents the unicity distance, $H(k)$ denotes the entropy of the key space and R is the plaintext redundancy in bits per character (Irvine, 1997). For an order 4 model, the unicity distance is equal to 33.2 and for an order 5 model is equal to 31.8. So we can not expect to correctly decrypt cryptograms shorter than this number of symbols. However, in our approach (Variant B), many different ciphertexts with short lengths (ranging from 20 to 40) have been tried, and in almost all cases the right solution (without any errors) was found either with the best solution or within the ten best solutions. Table 7 lists some examples of different cryptograms with short lengths and the successfully decrypted text.

Table 7: Examples of decryption of short cryptograms (length from 20 to 40 characters; the compression codelength is shown in the first column).

	Ciphertext	**Successfully decrypted text**
30.77	fvwdwradbsdfvwdobija	the end of the world
37.76	yniseiyre sgosynisbrvy	the return of the suit
64.71	fbapipuymswykpdbpubjypvumttyt	how i learned to love glasses
68.50	cgjgulg flrmuglfv clomxli clwhyf	never eat more than you can lift
74.84	vlmvhhvwwnlemtnwqljqmrlmekl mjrlasrw	an appalling silence on gun control
77.12	larrpmvcrnjil jm txm mc sspmtaw mpa r	merry christmas and a happy new year
72.72	igecq crgirwqcrkbcnfrnbfrwhr wq xcbinfgwq	silence is one great art of conversation

The average execution time that is needed to determine the correct solutions of the different ciphertexts that were experimented with is presented in Table 8. The time which is required to automatically break each ciphertext is based on the execution of the PPMD model without update executions and with specific order -1 codelength value (Variant B). The average elapsed time in seconds for each cryptogram is computed by running the program ten times on a Dell Inc.-Inspiron 5537 laptop computer (Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz) and then calculating the average. The results show that our method only requires a few seconds on average to decrypt the ciphertexts, and usually the solution is found in less than six seconds of CPU time.

Table 8: Average time needed to automatically cryptanalyse different ciphertexts.

Cipher Length	**20**	**50**	**150**	**300**
Time *(Sec)*	2.22	2.61	3.26	5.57

6.1 Experiments with different buffer sizes

Our search method requires maintaining a current list of the best solutions in a buffer of fixed size. In order to determine which is the best or the most appropriate buffer size in order to obtain the best results, we performed four further experiments using different buffer sizes: 100, 200, 500 and 1000. Table 9a and 9b present results regarding the average number of errors for the 110 ciphertexts when using the different buffer sizes.

Table 9a: Average number of errors when using different buffer sizes when examining just the best solution.

Array size	**100**	**200**	**500**	**1000**
Average errors	1.35	0.70	0.20	0.20

According to these tables, it is clear that buffer sizes of 500 and 1000 produced the smallest average number of errors. In contrast, using a buffer size of 100 resulted in a greater number of errors (1.35 compared to 0.20). However, a trade-off in favour of a smaller buffer size is that it uses up less memory and execution speed is faster. The program has not been optimised for memory usage and execution speed; however, we have noticed that the execution time doubled with the size of the buffer.

6.2 Improving results using a word-based PPM compression method

This section discusses the experimental results obtained when using a further word-based model for the automatic cryptanalysis. As word-based schemes ($W|W$ as described in Table 2) for English text outperform character-based ones in terms of compression rate, the order 1 word-based model is used here to examine the effect of applying this model in a secondary post-processing stage (Algorithm 2 as above) on the output from Algorithm 1 to see if this results in better cryptanalysis. This model re-orders an entry in the priority queue of solutions produced from Algorithm 1 if the word-based codelength is less than the character-based codelength.

Some sample output is shown in Table 10 for the ciphertext 'cgjgulg flr-muglfv clomxli clwhyf' which compares the ten best character-based solutions found by Algorithm 1 with the ten best word-based solutions found by Algorithm 2. The sample shows that the correct solution was found by the word-

Table 9b: Average number of errors when using different buffer sizes when examining the ten best solutions.

Array size	**100**	**200**	**500**	**1000**
Average errors	2.33	1.70	1.38	1.38

Table 10: The ten best character-based solutions compared to the ten best word solutions for the ciphertext 'cgjgulg flrmuglfv clomxli clwhyf'.

Ten best character solutions:		**Ten best word solutions:**	
65.994	never eat more than you can dist	64.041	never eat more than you can lift
66.439	never eat more than you can spit	68.016	never eat more than god pan just
68.030	never eat more than you can sixt	68.219	never eat more than you can list
68.388	never eat more than you can lift	69.054	never eat more than you can gift
68.453	never eat more than you can gift	69.441	never eat more than god can just
68.528	never eat bore than you can dist	69.804	never eat more than you can fist
68.745	never eat more than you can list	70.263	never eat more than you can spit
68.974	never eat bore than you can spit	70.456	never eat more than you can wilt
70.333	never eat wore than you can dist	70.678	never eat more than you can jist
70.565	never eat bore than you can sixt	70.804	never eat more than you can gilt

based method (with the semantically correct last word 'lift'), but in comparison this was ranked in fourth place using the character-based method. Interestingly, both methods have found similar solutions except for the third and last words which in most cases are correctly spelt although semantically incorrect.

This technique was tried only for variants found to be the best performing, Variants B and E. For variant B, five examples out of fourteen (which had been found by the character-based method to have three errors or less), were successfully solved with no errors when this secondary word-based method was applied. For variant E, six examples out of fifteen (which had been found to have three errors or less), are also solved with no errors using the same method.

After applying this word-based method, 92% of cryptograms were now solved without any errors, with an improvement of 5% and 6% for both variants B and E over when just using the order 5 character-based model. Table 11 shows how the word-based approach improves the average number of errors for the best solutions.

Table 11: Average number of errors when examining the best solution.

Variants	**Order 4 character model**	**Order 5 character model**	**Word-based** $(W\|W)$ **model**
Variant B	0.29	0.20	0.13
Variant E	0.31	0.22	0.13

Table 12 presents the summary of results when using our new method Variant B (the best performing method) on the ciphertexts that we used in our experiments. Overall, the results show that we are able to attain a very high success rate, with about 92% of cryptograms being correctly solved with no errors and 100% being decrypted with just three errors or less.

Table 12: Summary of results for Variant B.

Errors	Order 4 character model		Order 5 character model		Order 4 or 5 character model & Word-based model	
	No. of ciphertexts	Cumulative percentage	No. of ciphertexts	Cumulative percentage	No. of ciphertexts	Cumulative percentage
0	89	80.91%	96	87.27%	101	91.82%
≤ 1	101	91.82%	105	95.45%	106	96.36%
≤ 2	108	98.18%	109	99.09%	109	99.09%
≤ 3	110	100%	110	100%	110	100%

7 Conclusions

In this paper, a new approach to cryptanalysis has been introduced. A new method for the plaintext recognition and automated cryptanalysis of substitution ciphers in a ciphertext only attack has been described. An efficient use of the compression-based approach for cryptanalysis has been demonstrated. The fundamental idea behind our approach relies on using a compression method as an accurate way of measuring information in the text. Results on 110 cryptograms ranging from 20 to 300 characters have shown a high success rate at automatically recognising valid solutions for a range of distinct ciphertexts with different lengths with approximately 92% of the cryptograms correctly decrypted without any errors and 100% with just three errors or less. This efficient method works well on even very short ciphertexts and eliminates any need for human intervention.

For our paper, three main compression methods have been investigated: Prediction by Partial Matching (or PPM), Gzip and Bzip2. Various character-based PPM variants were investigated as well, in order to ascertain the most efficient scheme when applied to the problem of automatically breaking simple substitution ciphers. The following variants of PPM were used: PPMD and PPMC, with further variations such as the use or not of update exclusions, a technique found to improve compression but which we have found leads to better decryption if it is removed. Both of these variants were further investigated with a new variation of the PPM algorithm where a specific codelength value is assigned when encoding all order -1 contexts. The experimental results showed that this new combination, PPM without update exclusions using specific order -1 codelength values, noticeably outperforms other compression schemes including Gzip and Bzip2. We have also applied a word-based PPM model as a post-processing stage which led to further improved results.

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

Al-Kazaz, N. R., Irvine, S. A., & Teahan, W. J. (2016). An automatic cryptanalysis of transposition ciphers using compression. In *International Conference on Cryptology and Network Security* (pp. 36–52). Springer International Publishing.

Anderson, R. (1989a). Finding vowels in simple substitutions ciphers by computer. In *Cryptology: Machines, History & Methods.* Artech House.

Anderson, R. (1989b). Improving the machine recognition of vowels in simple substitution ciphers. In *Cryptology: Machines, History & Methods.* Artech House.

Ball, W. W. R. (1960). *Mathematical recreations and essays.* Macmillan, NY.

Bell, T. C., Cleary, J. G., & Witten, I. H. (1990). *Text compression.* Prentice-Hall, Inc.

Bentley, J. L., Sleator, D. D., Tarjan, R. E., & Wei, V. K. (1986). A locally adaptive data compression scheme. *Communications of the ACM*, *29*(4), 320–330.

Burrows, M., & Wheeler, D. (1994). A block-sorting lossless data compression algorithm. *Technical report, Digital Equipment Corporation, Palo Alto, California*.

Bzip2. (2016). *The bzip2 home page.* Retrieved from `http://www.bzip.org`

Carroll, J. M., & Martin, S. (1986). The automated cryptanalysis of substitution ciphers. *Cryptologia*, *10*(4), 193–209.

Chen, J., & Rosenthal, J. S. (2012). Decrypting classical cipher text using Markov Chain Monte Carlo. *Statistics and Computing*, *22*(2), 397–413.

Clark, A. (1994). Modern optimisation algorithms for cryptanalysis. In *Intelligent Information Systems. Proceedings of the 1994 second Australian and New Zealand Conference on* (pp. 258–262). IEEE.

Clark, A. (1998). *Optimisation heuristics for cryptology.* Ph.D. thesis, Queensland University of Technology.

Cleary, J., Teahan, W., & Witten, I. (1995). Unbounded length contexts for PPM. In *Data Compression Conference. DCC'95. Proceedings* (pp. 52–61). IEEE Computer Society Press.

Cleary, J., & Witten, I. (1984). Data compression using adaptive coding and partial string matching. *IEEE transactions on Communications*, *32*(4), 396–402.

Corlett, E., & Penn, G. (2010). An exact A* method for deciphering letter-substitution ciphers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics* (pp. 1040–1047).

Delman, B. (2004). *Genetic algorithms in cryptography.* Master's thesis, Department of Computer Engineering, Rochester Institute of Technology.

Eskicioglu, A., & Litwin, L. (2001). Cryptography. *IEEE Potentials*, *20*(1), 36–38.

Forsyth, W. S., & Safavi-Naini, R. (1993). Automated cryptanalysis of substitution ciphers. *Cryptologia*, *17*(4), 407–418.

Friedman, W. F. (1976). *Elements of cryptanalysis* (Vol. 3). Aegean Park Press, Laguna Hills, CA.

Gaines, H. F. (1956). *Cryptanalysis: A Study of ciphers and their solution.* Dover Publications.

Garg, P., & Sherry, A. (2005). Genetic algorithm & Tabu search attack on the mono-aiphanetic substitution cipher. *Paradigm*, *9*(1), 106–109.

Grundlingh, W., & Van Vuuren, J. H. (2003). Using genetic algorithms to break a simple cryptographic cipher. *Retrieved March*, *31*.

Gzip. (2012). *The gzip home page.* Retrieved from `http://www.gzip.org`

Hart, G. W. (1994). To decode short cryptograms. *Communications of the ACM*, *37*(9), 102–108.

Hilton, R. (2012). *Automated cryptanalysis of monoalphabetic substitution ciphers using stochastic optimization algorithms.* Ph.D. thesis, Department of Computer Science and Engineering, University of Colorado, Denver.

Horspool, R. N., & Cormack, G. V. (1992). Constructing Word-Based text compression algorithms. In *Data Compression Conference* (pp. 62–71). Snowbird, Utah: IEEE Computer Society Press.

Howard, P. G. (1993). *The design and analysis of efficient lossless data compression systems.* Ph.D. thesis, Brown University, Providence, Rhode Island.

Howard, P. G., & Vitter, J. S. (1992). Practical implementations of arithmetic coding. *Image and text compression*, 85–112.

Hunter, D., & McKenzie, A. (1983). Experiments with relaxation algorithms for breaking simple substitution ciphers. *The Computer Journal*, *26*(1), 68–71.

Irvine, S. A. (1997). *Compression and cryptology.* Ph.D. thesis, University of Waikato, New Zealand.

Jakobsen, T. (1995). A fast method for cryptanalysis of substitution ciphers. *Cryptologia*, *19*(3), 265–274.

King, J. C. (1994). An algorithm for the complete automated cryptanalysis of periodic polyalphabetic substitution ciphers. *Cryptologia*, *18*(4), 332–355.

King, J. C., & Bahler, D. R. (1992). An implementation of probabilistic relaxation in the cryptanalysis of simple substitution ciphers. *Cryptologia*, *16*(3), 215–225.

Knight, K., Nair, A., Rathod, N., & Yamada, K. (2006). Unsupervised analysis for decipherment problems. In *Proceedings of the COLING/ACL on Main conference poster sessions* (pp. 499–506). Association for Computational Linguistics.

Lee, D.-S. (2002). Substitution deciphering based on HMMs with applications to compressed document processing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*(12), 1661–1666.

Lee, K., Teh, C., & Tan, Y. (2006). Decrypting english text using enhanced frequency analysis. In *National Seminar on Science, Technology and Social Sciences* (pp. 1–7).

Lucks, M. (1990). A constraint satisfaction algorithm for the automated decryption of simple substitution ciphers. In *Advances in Cryptology-CRYPTO'88* (pp. 132–144). Springer.

Luthra, J., & Pal, S. K. (2011). A hybrid firefly algorithm using genetic operators for the cryptanalysis of a monoalphabetic substitution cipher. In *Information and Communication Technologies (WICT), 2011 World Congress on* (pp. 202–206). IEEE.

Moffat, A. (1989). Word-based text compression. *Software: Practice and Experience*, *19*(2), 185–198.

Moffat, A. (1990). Implementing the PPM data compression scheme. *IEEE Transactions on communications*, *38*(11), 1917–1921.

Nuhn, M., Schamper, J., & Ney, H. (2013). Beam search for solving substitution ciphers. In *Acl (1)* (pp. 1568–1576).

Olson, E. (2007). Robust dictionary attack of short simple substitution ciphers. *Cryptologia*, *31*(4), 332–342.

Peleg, S., & Rosenfeld, A. (1979). Breaking substitution ciphers using a relaxation algorithm. *Communications of the ACM*, *22*(11), 598–605.

Ravi, S., & Knight, K. (2009). Attacking letter substitution ciphers with integer programming. *Cryptologia*, *33*(4), 321–334.

Robling Denning, D. E. (1982). *Cryptography and data security.* Addison-Wesley Longman Publishing Co., Inc.

Schatz, B. R. (1977). Automated analysis of cryptograms. *Cryptologia*, *1*(2), 116–142.

Schneier, B. (1993). Applied cryptography. *John Wiley & Sons, New York*.

Spillman, R., Janssen, M., Nelson, B., & Kepner, M. (1993). Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers. *Cryptologia*, *17*(1), 31–44.

Teahan, W. J. (1998). *Modelling English text.* Ph.D. thesis, University of Waikato, New Zealand.

Teahan, W. J., & Cleary, J. G. (1996). The entropy of english using PPM-based models. In *Data Compression Conference, 1996. DCC'96. Proceedings* (pp. 53–62).

Teahan, W. J., Wu, P., & Liu, W. (2014). Adaptive compression-based models of Chinese text. In *Audio, Language and Image Processing (ICALIP), 2014 International Conference on* (pp. 874–881).

Uddin, M. F., & Youssef, A. M. (2006). Cryptanalysis of simple substitution ciphers using particle swarm optimization. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on* (pp. 677–680). IEEE.

Vobbilisetty, R., Di Troia, F., Low, R. M., Visaggio, C. A., & Stamp, M. (2017). Classic cryptanalysis using hidden Markov models. *Cryptologia*, *41*(1), 1–28.

Wiener, M. J. (1993). Efficient DES key search. In Advances in Cryptology: CRYPTO'93, volume 773 of Lecture Notes in Computer Science, Berlin. Springer-Verlag.

Witten, I. H., & Bell, T. C. (1991). The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE*

transactions on information theory, *37*(4), 1085–1094.
Witten, I. H., Neal, R. M., & Cleary, J. G. (1987). Arithmetic coding for data compression. *Communications of the ACM*, *30*(6), 520–540.