Word-Based Grammars for PPM

Nojood O. Aljehane
College of Computers & Information Technology
University of Tabuk,
Tabuk, Saudi Arabia

William J. Teahan
Department of Computer Science
Bangor University
Bangor, United Kingdom

***Abstract*—The Prediction by Partial Matching (PPM) compression algorithm is considered one of the most efficient methods for compressing natural language text. Despite the advances of the PPM method for the English language to predict upcoming symbols or words, more research is required to devise better compression methods for other languages, such as Arabic due, for example, to the rich morphological nature of the Arabic text, where a word can take many different forms. In this paper, we propose a new method that achieves the best compression rates not only for Arabic text but also for other languages that use Arabic script in their writing system such as Persian. Our word-based method constructs a context-free grammar (CFG) for the text and this grammar is then encoded using PPM to achieve excellent compression rates.**

Keywords—Component; context-free grammar (CFG); grammar-base; word-based; Preprocessing; Prediction by Partial Matching (PPM); encoding

I. Introduction

The Prediction by Partial Matching (PPM) compression algorithm is one of the most effective kinds of statistical compression. First described by Cleary and Witten in 1984 [1], there are many variants of the basic algorithm, such as PPMA and PPMB [1], PPMC [2], PPMD [3], PPM [4], PPMZ [5] and PPMii [6]. Prediction in PPM depends on a bounded number of previous characters or symbols, effectively using a Markov-based approach. Despite the cost in terms of memory and the speed of execution, PPM usually attains better compression rates compared with other well-known compression methods.

In PPM, to predict the next character or symbol, different orders of models are used, starting from the highest order down to the lowest orders. An escape probability estimates if a new symbol appears in the context [1], [2] and if an escape is encoded, the algorithm will back-off to a lower order model. The ‘full exclusions’ mechanism [1] is used to significantly improve compression by excluding the prediction of higher order symbols when an escape has occurred since these characters were not encoded [17]. Experimental results show that not using full exclusions speeds up the execution time of programs but compression is reduced.

However, when a PPM approach is applied to words rather than characters, it is not clear what the most effective method for encoding the text is. This is because there are issues of how to encode the spaces and punctuation along with the text, how to deal with capitalized words, whether to treat digit sequences differently, how to deal with the much larger alphabet when using full exclusions, and so on. This is compounded further when considering certain languages, such as Arabic, which has a rich morphological structure which potentially presents further types of difficulties for word-based compression compared to languages, such as English since the same word can take many different forms.

As an illustration, the lists below in Table 1 show the most common words in each of the examined texts. They are based on an analysis of the Brown Corpus for American English [9], the LOB Corpus for British English [10], the BACC [11] and CCA [12] Corpora for Arabic text, the Hamshahri corpus for Persian text [13] and the CEG corpus for Welsh text [16].

Substitution of these words using our context-free grammar scheme and standard PPM can significantly improve overall compression as shown below. For example, natural languages contain common sequences of words that often repeat in the same order, such as in English “the” , “of” and “and”, and for the Arabic language in the BACC corpus , such as “من”, “ في” and so on. From Table 1, the most common word “the” for both the American and British English is found to be “the”. However, for these corpora if one treats capitalized words as being distinct (that is, “the” is treated as distinct from “The”), we find that the word “The” also appears in the top 20 ranked words, but at different ranks (12 for the Brown Corpus versus 16 for the LOB Corpus). In contrast, the word “had” appears with the same rank for both corpora. Certain words, such as “from” and “at” appear in the list for one corpus but not for the other.

TABLE I. The Top Common 20 Words for the Brown, LOB, BACC, CCA and Hamshahri Text Corpora

Rank	Brown Corpus	LOB Corpus	BACC Corpus	CCA Corpus	Hamshahri
1	the	the	من	في	برورش
2	of	of	بن	من	دستور
3	and	and	قال	على	سمتی
4	to	a	الله	ان	این
5	a	in	ما	إلى	حمید
6	in	that	بالله	التي	در
7	that	is	في	عن	اعلام
8	is	was	أبو	ما	بنیادی
9	was	for	محمد	لا	اظهارات
10	for	it	عليه	هذا	صف
11	with	to	على	هذه	كند
12	The	be	إلى	الذي	افزود
13	as	his	أن	أو	برای
14	he	as	علي	و	بر
15	it	on	عبد	كان	آول
16	his	The	عنه	مع	كردند
17	on	his	له	لم	كجا
18	be	at	ثم	كل	كه
19	from	as	لك	ذلك	كذشته
20	had	had	الذي	بين	اداره

For Arabic text, the most common word for both the BACC and ACC Corpora is found to be "في" (in). Nevertheless, we find that the word "ان" (that) also appears in the top 20 ranked words, but at different ranks (4 for the ACC Corpus versus 6 for the BACC Corpus). In contrast, the word "من" (from) appears with the same rank for both corpora. Certain words such as " التي" (which) and "له" (for him) appear in the list for one corpus but not for the other. For Persian text in the Hamshahri Corpus, even if it uses Arabic script, the top 20 ranked words are noticeably different due to the difference between these two languages.

From these lists, it is clear even just from examining the top 20 ranking words that there are important differences, and therefore word-based compression schemes have to adapt directly to the text being compressed in an online manner (as PPM does) rather than use dictionaries created from general sources. Another factor is that since the most frequent words represent a significant proportion of the text, adaptive word-based schemes can often lead to improved compression for many languages. An added advantage of such schemes is that much less symbols need to be encoded (for example, for English, there is on average approximately five times less word symbols than there are character symbols). However, finding the most effective word-based compression is still an open problem with word-based schemes under-researched compared to character-based schemes. The comparison between the effectiveness of word-based schemes with character-based and parts-of-speech (tags) based ones also provides an interesting tool for performing further linguistic analysis [8]. The main contribution of the work described in this paper is the improved word-based compression method for PPM. This is due to parsing of the text to construct a word-based context free grammar (CFG) which is then compressed using PPM.

The rest of the paper is organized as follows. Previous work is discussed first. Then our new approach is discussed in the next section. We discuss experimental results for various natural language texts in order to evaluate how well the new scheme performs compared to other well-known methods. The summary and conclusions are presented in the final section.

II. Previous Work

As stated, standard PPM word-based models predicts the forthcoming symbol, starting from the highest order context; but when the upcoming symbol has not appeared in this context then a lower context is used and an escape symbol is encoded. There have been a number of methods that have been used to estimate the probability for these escape symbols [7], [8].

Experiments indicate that the X1 method is the best performing for English text in the most cases [8]. This method is given by the formula:

$$e = \frac{t_1+1}{T_d+t_1+1} \qquad (1)$$

Here, t_1 denotes the number of symbols seen previously only once in the context and T_d is the frequency with which the symbol occurs in the context. Therefore, this method estimates the escape symbol probability proportionate to the number of words that have appeared only once in the text.

TABLE II. Some models for predicting characters and words (Teahan, 1998)

C|C^5 Model	W|W Model
$p(c_i \mid c_{i-1}\, c_{i-2}\, c_{i-3}\, c_{i-4}\, c_{i-5})$	$p(w_i \mid w_{i-1})$
$\rightarrow p(c_i \mid c_{i-1}\, c_{i-2}\, c_{i-3}\, c_{i-4})$	$\rightarrow p(w_i)$
$\rightarrow p(c_i \mid c_{i-1}\, c_{i-2}\, c_{i-3})$	$\rightarrow$ *Character model*
$\rightarrow p(c_i \mid c_{i-1}\, c_{i-2})$	
$\rightarrow p(c_i \mid c_{i-1})$	
$\rightarrow p(c_i)$	
$\rightarrow p_e\, q(c_i)$	

Experiments for the English language show that word based models in Table 2 presents the best performance among other models [8].

Model C|C^5 is a PPM character model of order five that predicts the probability of character symbols and used as a compression baseline. In this model, the formula for the probability of text string *S* of *m* characters is given by:

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

Where, the preceding five characters in the text is used to estimate the probability of the forthcoming symbol.

This estimate of the probability for the previous formula depends on the escape method (in Table 2, the symbol $\rightarrow$ denotes an escape). In character based models, if the highest order fails to predict forthcoming symbol, the probability of escape is encoded using the next highest order.

The second model W|W, is a PPM order one word-based model that predicts the probability of word symbols. In this model, the estimation of the probability for the forthcoming word depends on the previous word in the text as represented by the following formula for the probability of text string *S* of *n* words:

$$p(S) = \prod_{i=1}^{n} p(w_i \mid w_{i-1}) \qquad (3)$$

Where, *p* denotes the probability of the symbols in the sequence of the text *S* based on words. If the word is not predicted by this model, then an escape is encoded down to the order 0 model. If the word still has not been seen in this context, then a further escape is encoded followed by each character in the word being encoded separately using the standard PPM character-based model.

III. Word-Based Grammars for PPM (GRW-PPM)

A new approach based on word-based context free grammars (CFGs) for compressing text files is presented here. This algorithm, which we call GRW-PPM (which is short for grammar word-based pre-processing for PPM) uses both CFGs and PPM as the basis of a universal general-purpose adaptive compression method for text files.

In our approach, we essentially parse words, digits, spaces and punctuation in the source file to first generate a grammar with rules and terminal and non-terminal symbols representing each of these text elements. We then substitute every time when one of these text elements occurs in the source text with the single unique non-terminal symbol as specified by its rule in the grammar. This is done during the pre-processing phase prior to the PPM compression phase which is applied to the sequences of non-terminal symbols for words, digits and spaces and punctuation separately.

Our method replaces sequences of words (n-grams) in the text as they are processed from beginning to end in a single pre-processing pass. The PPM algorithm is used as the encoder once the sequences have been replaced. Unlike PPM, our method is off-line during the phase which generates the grammar.

Our approach adapts the W|W word-based method and the character n-graph replacement pre-processing approach of Teahan [8] by using an off-line technique to generate the list of word n-grams first from the source file being compressed. However, our approach is considered within a grammar-based context instead. The main difference with the prior word-based schemes (such as W|W) is the use of PPM to encode the sequence of word symbols directly without the need to escape to a separate character-level encoding and also treatment of digits as word symbols (see below).

The grammar in GRW-PPM shares the same characteristic as Sequitur by Neville-Manning and Witten [14] and GR-PPM [15] which is that no pair of symbols appears in the grammar more than once. This property ensures that every n-gram in the grammar is unique, a property called non-terminal uniqueness using the same terminology proposed by Neville-Manning and Witten. To make sure that each rule in the grammar is useful, the second property, referred to as rule utility, is that every rule in the grammar is used more than once in the corrected text sequence.

Fig. 1 shows the whole process of GRW-PPM. First, the original text will be parsed and word, digit and space/punctuation tokens will be extracted then the CFG will be generated by replacing them in the text wherever they occur with the non-terminal symbols as defined by their rules in the grammar. After the rules have been produced, the grammar is encoded by using PPMD, and the resulting compressed text is then sent to the receiver. The receiver then decodes the grammar by using PPMD to decompress the compressed file that was sent. The reverse mapping is then facilitated by using the decoded grammar to regenerate the original source text.

Table 3 illustrates the process of GRW-PPM using a sentence referring to the song by Manfred Mann: "*The song 'Do Wah Diddy Diddy Dum Diddy Do' was recorded on 11 June 1964 and released on 10 July*". First, the original text will be parsed from left to right and new non-terminal word and digit symbols (S_1 S_2 S_3 S_4 S_5 S_5 S_6 S_5 … S_{12} S_9 D_3 S_{13}) will be substituted for each unique n-gram (defined as being separated by the intervening space and punctuation symbols). For this example (and for the experiments described below), we use single words (unigrams), although the method works in a similar way for word bigrams and trigrams. Referring to Table 3, we replace the unigram "The" with non-terminal symbol S_1, unigram "song" with non-terminal symbol S_2, unigram "Do" with non-terminal symbol S_3 and so on. We use bullet points for spaces to make them visible. Spaces (white-space) and punctuation define the word boundaries (i.e. each word is made up of sequences of anything that is not white-space or punctuation).

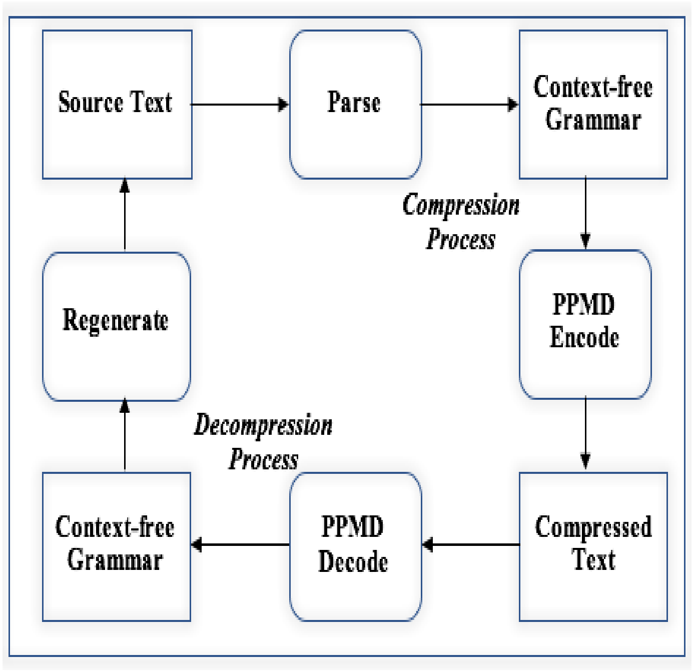
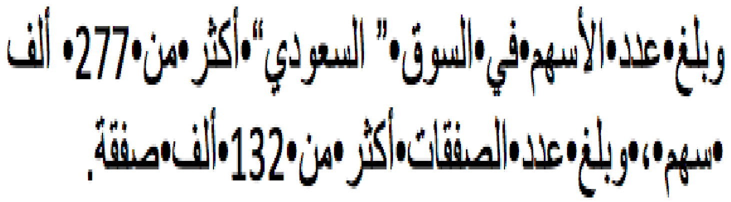


Fig. 1. The complete compression and decompression process of GRW-PPM.



وبلغ•عدد•الأسهم•في•السوق•" السعودي "•أكثر•من•277• ألف
سهم•،•وبلغ•عدد•الصفقات•أكثر•من•132•ألف•صفقة.

Fig. 2. Example of Arabic text.

TABLE III. AN EXAMPLE OF HOW GRW-PPM WORKS FOR A SAMPLE ENGLISH TEXT

Sequence:	
The•song•"Do•Wah•Diddy•Diddy•Dum•Diddy•Do"•was•recorded•on•11•June •1964•and•released•on•10•July.	
Grammar: $S \rightarrow S_1\ S_2\ S_3\ S_4\ S_5\ S_5\ S_6\ S_5\ S_3\ S_7\ S_8\ S_9\ S_D\ S_{10}\ S_D\ S_{11}\ S_{12}\ S_9\ S_D\ S_{13}$ $V \rightarrow S_1\ S_2\ S_3\ S_4\ S_5\ S_6\ S_7\ S_8\ S_9\ S_{10}\ S_{11}\ S_{12}\ S_{13}$ $D \rightarrow D_1\ D_2\ D_3$ $P \rightarrow P_1\ P_2\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_3\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_4$ $S_1 \rightarrow$ "The" $S_2 \rightarrow$ "song" $S_3 \rightarrow$ "Do" $S_4 \rightarrow$ "Wah" $S_5 \rightarrow$ "Diddy" $S_6 \rightarrow$ "Dum" $S_7 \rightarrow$ "was"	$S_8 \rightarrow$ "recorded" $S_9 \rightarrow$ "on" $S_{10} \rightarrow$ "June" $S_{11} \rightarrow$ "and" $S_{12} \rightarrow$ "released" $S_{13} \rightarrow$ "July" $D_1 \rightarrow$ "11" $D_2 \rightarrow$ "1964" $D_3 \rightarrow$ "10" $P_1 \rightarrow$ "•" $P_2 \rightarrow$ "•"" $P_3 \rightarrow$ """•" $P_4 \rightarrow$ "."

Table 4 shows the same process for a sample Arabic text (Fig. 2) which translates into English as follows: “The number of shares traded in the market, ‘Saudi’ were more than 277 thousand shares, and the number of transactions were more than 132 thousand transactions.” However, in this case the n-grams are generated from right to left instead. Each unique Arabic unigram has a non-terminal symbol associated with it. For instance, words “وبلغ”, “عدد” and “الأسهم” are replaced by non-terminal symbols S_1 to S_3, respectively.

In the grammar examples, the S rule is used to represent the word and digit symbols sequence. Separate rules (S_1, S_2, S_3 …) are used, one for each word, to specify each symbol’s contents directly using a non-terminal (denoted by characters surrounded by “ ’s). The V rule enumerates each of these words in order; it is used to represent the vocabulary (the sequence of unique words as they occur in the text). Each digit sequence is encoded within the S sequence by using a special symbol to indicate the positions of the digits in the sequence (as represented by S_D in the above examples). The actual contents of each digit symbol is specified by the D rule and encoded separately to the word and digit symbols. We also process spaces and any punctuation characters in order to be able to fully decode the original text back. These are represented by the P rules for the grammars in the above examples and are similarly encoded separately to the word and digit unigram symbols. Moreover, the grammar will be transmitted to the receiver once it has been constructed after all unigrams are substituted in the original text with their non-terminal symbols.

The grammar represents a complete description of the text and therefore it is possible to devise a lossless text compression scheme by directly encoding it in some manner since it is possible for the decoder to regenerate the complete source text losslessly once the grammar has been decoded.

TABLE IV. ANOTHER EXAMPLE GRAMMAR GENERATED BY GRW-PPM FOR A SAMPLE ARABIC TEXT

Sequence:	
وبلغ•عدد•الأسهم•في•السوق•”السعودي“•أكثر•من•277• ألف •سهم•وبلغ•عدد•الصفقات•أكثر•من•132•ألف•صفقة.	
Grammar:	
$S \rightarrow S_1\ S_2\ S_3\ S_4\ S_5\ S_6\ S_7\ S_3\ S_7\ S_8\ S_D\ S_9\ S_{10}\ S_1\ S_2\ S_{11}\ S_7\ S_8\ S_D\ S_9\ S_{12}$	$S_8 \rightarrow$ “من”
$V \rightarrow S_1\ S_2\ S_3\ S_4\ S_5\ S_6\ S_7\ S_8\ S_9\ S_{10}\ S_{11}\ S_{12}$	$S_9 \rightarrow$ “ألف”
$D \rightarrow D_1\ D_2\ D_3$	$S_{10} \rightarrow$ “سهم”
$P \rightarrow P_1\ P_1\ P_1\ P_1\ P_2\ P_3\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_1\ P_4$	$S_{11} \rightarrow$ “الصفقات”
$S_1 \rightarrow$ “وبلغ ”	$S_{12} \rightarrow$ “صفقة”
$S_2 \rightarrow$ “عدد”	$D_1 \rightarrow$ “277”
$S_3 \rightarrow$ “الأسهم”	$D_2 \rightarrow$ “132”
$S_4 \rightarrow$ “في”	$P_1 \rightarrow$ “•”
$S_5 \rightarrow$ “السوق”	$P_2 \rightarrow$ ““•”
$S_6 \rightarrow$ “السعودي”	$P_3 \rightarrow$ “•””
$S_7 \rightarrow$ “أكثر”	$P_4 \rightarrow$ “."

As stated, we have found one effective means for encoding the grammar is to use PPM. Specifically, the grammar is encoded by using PPMD to separately encode the four main elements (words, vocabulary, digits and spaces/punctuation as represented by the S, V, D and P rules). For Rule S, we can encode the sequence of symbol numbers or letters that appear in the rule. For example, in Table 3, the sequence of symbol numbers/letters for Rule S is as follows: 1 2 3 4 5 5 6 5 3 7 8 9 D 10 D 11 12 9 D 13. This represents the sequence of id numbers assigned to each unique word with id numbers starting from 1 and incrementing by one whenever a new word is encountered. The letter D indicates when a digit sequence has occurred. Clearly, the sequence for rule S will be highly repetitive for long sequences of natural language text because of the presence of repeated words and frequent function words (such as “*the*” and “*and*” for English and “*من*” and “ في” for Arabic as shown in Table 1). More specifically, we have found PPMD to be very effective at encoding this sequence.

However, unlike W|W (which uses similar PPM-like methods to encode word symbols in this manner), our method simply uses PPMD with a fixed maximum alphabet size (since this is known when the grammar has been fully constructed for the whole text). Also, our method does not need to encode an escape down to a separate character-level as W|W does in order to encode novel words when they occur.

Instead, it uses the standard PPMD encoding mechanism (where a novel symbol will be encoded using a default order -1 model where all symbols are equiprobable).

For practical purposes, rule V and rules S_1, S_2, S_3, … can simply be represented as a string of text that contains all the unique words as they appear in the source text one after another with a separator (such as a space character) used to indicate the end of the previous word and the beginning of the next one. Similarly, we can use the same encoding technique for the digit sequences for rule D and rules D_1,D_2,D_3,… and for the spaces and punctuation for rule P and rules P_1,P_2,P_3,…. That is, both the digits and punctuation can be encoded effectively by using PPMD to encode one text string that contains all the unique digit sequences and another text string that contains the unique space and punctuation sequences respectively. A space character can be used as a separator for the digits, but for the punctuation, a different separator is needed. We use the letter “W” as the separator in this case to mark where the words are.

As an illustration, Table 5 presents the symbols or text that is being encoded for the four elements (symbols, vocabulary, digits, spaces and punctuation) for the beginning of the Brown corpus. All are encoded directly by PPMD as text except for the Symbols element which is treated as a sequence of numbers instead.

The decompression process first uses PPMD to decode the four separate elements and then re-constructs the full grammar from them. During the subsequent regeneration phase, the grammar is then used to exactly regenerate the original source text character for character (i.e. the method is completely lossless). Whenever a previously unseen symbol is encountered as the sequence specified by the S rule is being processed, the current word is read from the sequence specified by the V rule and then the position is moved along to the next word.

TABLE V. WHAT THE DIFFERENT TEXT ELEMENTS LOOK LIKE FOR THE BEGINNING OF THE BROWN CORPUS

Brown Corpus (text at the start of the corpus): The Fulton County Grand Jury said Friday an investigation of Atlanta's recent primary election produced "no evidence" that any irregularities took place. The jury further said in term-end presentments that the City Executive Committee, which had over-all …			
Symbols	**Vocabulary**	**Digits**	**Spaces & Punctuation**
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 2 25 26 7 27 28 29 30 20 31 32 33 34 35 36 37 38 39 11 31 16 40 31 41 42 43 11 31 32 11 12 44 31 45 27 35 31 16 46 47 2 48 49 28 25 36 50 51 52 3 53 54 55 56 57 58 59 60 11 61 …	The Fulton County Grand Jury said Friday an investigation of Atlanta s recent primary election produced no evidence that any irregularities took place jury further in term end presentments the City Executive Committee which had over all charge …	1 1 2 2 1913 71 74 637 1937 1923 1 13 1962 8 1961 100 30 3 4 1958 50 10 87 31 29 5 13 1 119 402 18 17 63 31 300 000 6 13 451 500 157 460 88 000 182 17 000 1 000 12 3 81 65 4 22 1 4 250 114 4 5 000 000 15 000 000 24 12 30 24 4 150 13 1961 62 10 …	W W W W W W W W W W W'W W W W W "W W" W W W W W. W W W W W W-W W W W W W W, W W W-W W W W W, "W W W W W W W W W W" W W W W W W W W W. W W-W W W W W W W W W W W W W W W W W W "W" W W W-W W W W W W W-W W W W&. "W W W W W W W W W", W W W, "W W W W W W W, W W W W W W W W W W". W W W W W W W…

The P rule is used to insert the punctuation between the word and digit symbols as they are encountered in the S rule. Whenever a digit is signified by the S_D symbol for this rule, the current digit symbol is read from the sequence specified by the D rule, which is then inserted into the decoded output sequence and the position then moves along to the next digit symbol.

Algorithm 1 summarizes the algorithm using pseudo-code. Lines 1 through 15 are for the n-gram tokenizer. Line 3 starts the **for** loop to read the n-grams in the input file. Lines 4 through 9 check if the n-gram is a word; if it is, it prints the n-gram to the Grammar file, assigns each id numbers with ids for unique n-grams increasing with each new n-gram that is found and also prints a W to the Spaces & Punctuation file. Lines 10 through 13 checks if this n-gram is a digit; if it is, it adds this digit to the digit file and prints W to the Spaces & punctuation file. Lines 14 and 15 checks if this n-gram is punctuation or space; if so these are added to the Spaces & Punctuation file. Line 16 compresses the final text for the four files by using PPMD.

Algorithm 1: Pseudo-code for the GRW-PPM.

```
Input: The source text file
Output: Compressed text

1  Open Grammar, SYMBOLS, DIGITS and SPACES &
   PUNC. files
2  id ← 2
3  for each n-gram in input file do
4      if ngram is a new word then
5          print n-gram to Grammars file
6          Assign id to each n-gram
7          print id to SYMBOLS file
8          print W to SPACES & PUNC. file
9          id ← id + 1
10     else if n-gram is a digit then
11         print n-gram to DIGITS file
12         print number 1 to SYMBOLS file
13         print W to SPACES & PUNC. file
14     else
15         print n-gram to SPACES & PUNC. file
16 Use PPMD to encode the four files.
```

A further improvement of our approach, both in terms of compression and execution speed, can be gained by further processing the files in the following manner. The main disadvantage of the Symbols file is that it consists of many singletons that occur only once in the text and doubletons that occurs only twice [18]. Singletons and doubletons are detrimental to the encoding efficiency because they do not give any useful reference information [19]. In addition, singletons incur an unnecessary extra cost in our scheme because their symbol numbers are unique and therefore cause the alphabet size to be incremented by 1 each time they occur (which is frequently due to the Zipf's Law-like nature of natural language text). As a result, the alphabet size can be substantially higher when these are present. A large alphabet for PPM is undesirable when using the full exclusions mechanism [1] that PPM uses for its encoding as it substantially slows down execution speeds due to the need to exclude symbols already seen in the higher orders from lower order predictions.

In order to overcome these problems and therefore improve our new method, we process the Symbols file to replace all singletons in the Symbols file with the same special symbol wherever they occur. For example, for the Symbols stream "1 6 7 6 7 7 4 5" there are three singletons – 1, 4 and 5. These singletons get replaced by a special symbol (Φ, say) and the Symbols sequence being encoded becomes "Φ 6 7 6 7 7 Φ Φ'. Each singleton can be readily decoded once the special symbol is encountered in the Symbols stream which signals to the decoder to read the characters for the word from the next set of characters in the Vocabulary stream up until the next word separator character. For our example, let's say that the characters in the Vocabulary stream are "*one six seven four five*". When replacing just singletons in the Symbols stream, there is no need to change this Vocabulary stream since the decoder will have all the necessary information to decode each word since singletons only occur once. The only effect is that the Symbols stream becomes slightly more compressible with a much smaller alphabet which significantly speeds up compression speeds when performing full exclusions as shown below.

We also have an option to replace doubletons and tripletons (and so on) wherever they occur in the Symbols file if we wish. However, when replacing non-singletons in this case, there is no way to decode the characters when the word is being replaced the second time or subsequent times (for tripletons etc.) so a simple expedient is to repeat the word character for character in the Vocabulary stream whenever it occurs again. Using the previous example again, if we were to replace singletons and doubletons (but not tripletons), then the Symbols sequence would now be encoded as "Φ Φ 7 Φ 7 7 Φ Φ" since the symbol 6 appears twice (i.e. it is a doubleton) but symbol 7 appears three times (i.e. it is not a singleton or doubelton). In the Vocabulary stream in this case, the characters for symbol 6 would appear twice, i.e. it would now become "*one six seven six four five*" since the word "*six*" is a doubleton and therefore appears again in this sequence. Clearly, the size of the Vocabulary stream now will grow because of the presence of the repeated words and this can affect the overall compression, but this is offset by the significantly faster processing since the alphabet size in the Symbols stream is much smaller.

In the experimental results below, we use the following labels for the variants of our algorithm: GRW-PPM for our standard algorithm; GRW1-PPM for when singletons are replaced by the special symbol; GRW2-PPM for when both singletons and doubletons are replaced; GRW3-PPM for when all the singletons, doubletons and tripletons are replaced; and GRW4-PPM for when all the singletons, doubletons, tripletons and quadrupletons are replaced.

IV. Experimental Results

This section discusses experimental results using GRW-PPM and its variants described above for compression of various text files. We compare our new method with other compression schemes. Also, we discuss in this section the encoding execution times for GRW-PPM with and without using the full exclusions mechanism that PPM uses for its encoding.

In this experiment, the GRW-PPM encoding is divided into four parts. The four parts are for the Grammar, the Symbols, the Digits and the Spaces and Punctuation. Order 5 PPMD is used for the Grammar, order 1 PPMD for the Symbols, order 4 PPMD for the Digits and for Spaces and Punctuation, order 4 PPMD is used. Experiments showed these different orders were the most effective at compressing the different text elements.

Table 6 illustrates the compression ratio for the four parts. The compression ratio is calculated by multiplying the compressed output size in bytes times 8 divided by the original input file size in order to determine the contribution each part has to the overall encoding cost. As shown in the table, the Digits part has the smallest compression rate for the different languages. Also, the compression rate for Grammar and Spaces and Punctuation are small compared to the Symbols part for the Brown, LOB, CEG, Hamshahri and BACC corpora.

As shown in Table 7, order 1 GRW3-PPM significantly outperforms order 1 GRW-PPM as it has the best compression ratio for the corpora being compressed. The improvement of GRW3-PPM over GRW-PPM occurs for all texts and ranges from over 2% to 4.2% for the BACC corpus of Arabic text.

From our experiments as shown in Tables 7 and 10 for different text files, we found that full exclusions improves the compression rate. However, this increases the execution time slightly because for full exclusions all symbols are removed for prediction in the lower order level if they have already been seen in the higher order. (There may be many symbols needing to be excluded depending on the context.) The configuration of our test machine is 4 GB GHz intel Core i5, with 4GB internal memory.

It is clear from Tables 8 and 9 that not using full exclusions result in a worse compression rate. The improvement of GRW1-PPM and GRW2-PPM with full exclusions over GRW1-PPM and GRW2-PPM without using full exclusion ranges on average from just over 4% to 5.4% for all texts. However, the advantage in not performing full exclusions is that this runs on average 3% to 20% more quickly for different texts.

Table 11 shows an interesting result when comparing GRW-PPM and GRW3-PPM with PPMD and W|W. It is clear that GRW3-PPM on average significantly outperforms W|W. GRW3-PPM shows an average 7.1% improvement over W|W. Also, it illustrates that there are significant differences between each of the compression methods for different languages.

TABLE VI. Compression Ratios for GRW-PPM In The Four Elements For the Different Sample Texts

File	**Language or Dialect**	**Size**	**Symbols (bpc)**	**Vocabulary (bpc)**	**Digits (bpc)**	**Spaces & Punct. (bpc)**	**Overall (bpc)**
Brown	American English	5968707	1.698	0.226	0.014	0.278	2.21
LOB	British English	6085270	1.628	0.217	0.016	0.191	2.05
BACC	Arabic	31018167	1.078	0.143	0.006	0.173	1.40
Hamsh.	Persian	1120834	0.982	0.311	0.042	0.101	1.43
CEG	Welsh	6753317	1.284	0.147	0.089	0.214	1.73

TABLE VII. COMPRESSION RATIOS FOR GRW-PPM WITH FULL EXCLUSIONS COMPARED WITH GRW1-PPM, GRW2-PPM AND GRW3-PPM PERFORMANCE FOR DIFFERENT NATURAL LANGUAGES

File	GRW-PPM (bpc)	GRW1-PPM (bpc)	GRW2-PPM (bpc)	GRW3-PPM (bpc)	GRW4-PPM (bpc)
Brown	2.21	2.16	2.15	**2.14**	**2.14**
LOB	2.03	1.99	**1.98**	**1.98**	**1.98**
BACC	1.40	1.35	**1.34**	**1.34**	**1.34**
Hamsh.	1.43	1.41	1.40	**1.39**	**1.39**
CEG	1.73	1.70	**1.69**	**1.69**	**1.69**
Average	1.76	1.72	**1.71**	**1.71**	**1.71**

TABLE VIII. GRW-PPM WITHOUT FULL EXCLUSIONS COMPARED WITH GRW1-PPM, GRW2-PPM AND GRW3-PPM PERFORMANCE FOR DIFFERENT NATURAL LANGUAGES

File	GRW-PPM (bpc)	GRW1-PPM (bpc)	GRW2-PPM (bpc)	GRW3-PPM (bpc)	GRW4-PPM (bpc)
Brown	2.35	2.33	2.23	2.23	2.23
LOB	2.14	2.11	2.07	2.06	2.06
BACC	1.49	1.45	1.43	1.43	1.43
Hamsh.	1.52	1.48	1.46	1.46	1.46
CEG	1.80	1.76	1.76	1.75	1.75
Average	1.86	1.82	1.79	1.79	1.79

TABLE IX. EXECUTION TIMES FOR GRW1-PPM, GRW2-PPM, AND GRW3-PPM WHEN NOT USING FULL EXCLUSIONS

File	GRW1-PPM (seconds)	GRW2-PPM (seconds)	GRW3-PPM (seconds)	GRW4-PPM (seconds)
Brown	722.25	481.15	389.04	320.10
LOB	596.83	583.66	353.13	296.02
BACC	5655.20	4156.35	2339.16	3179.45
Hamsh.	2544.21	1375.30	965.34	843.35
CEG	275.82	198.56	193.31	138.03

TABLE X. EXECUTION TIMES FOR GRW1-PPM, GRW2-PPM, AND GRW3-PPM WHEN USING FULL EXCLUSIONS

File	GRW1-PPM (seconds)	GRW2-PPM (seconds)	GRW3-PPM (seconds)	GRW4-PPM (seconds)
Brown	760.83	600.05	471.92	342.59
LOB	670.20	436.98	328.12	329.57
BACC	6149.99	5292.56	3693.99	3320.88
Hamsh.	3260.91	2062.56	1268.33	916.22
CEG	302.71	264.58	239.56	173.57

TABLE XI. COMPARING THE PERFORMANCE OF THE PPMD, PPM WORD-BASED, GRW-PPM AND GRW3-PPM MODELS

File	Size	PPMD Order4 (bpc)	W|W Order4 (bpc)	GRW-PPM Order1 (bpc)	GRW3-PPM Order1 (bpc)
Brown	5968707	2.22	**2.13**	2.21	2.14
LOB	6085270	2.03	**1.96**	2.05	1.98
BACC	31018167	1.57	1.59	1.40	**1.34**
Hamsh.	1120834	1.75	1.79	1.43	**1.39**
CEG	6753317	**1.69**	1.70	1.73	**1.69**
Avg.		1.85	1.83	1.76	**1.70**

For instance, for American English text, W|W achieves the best compression rate compared with other models, with a 3.6% improvement over GRW-PPM and a 0.45% improvement over GRW3-PPM. For British English text, W|W achieves a 4.3% improvement over GRW-PPM and a 1.0% improvement over GRW3-PPM. For Welsh, GRW3-PPM and PPMD attain a 2.3% improvement over GRW-PPM and approximately a 1.0% improvement over W|W. For Arabic text, GRW3-PPM outperforms the other models, attaining a 14.6% improvement over PPMD and a 15.7% significant improvement over W|W. For Persian text, GRW3-PPM exceeds the other models, with a 22.3% improvement over W|W (see Fig. 3).

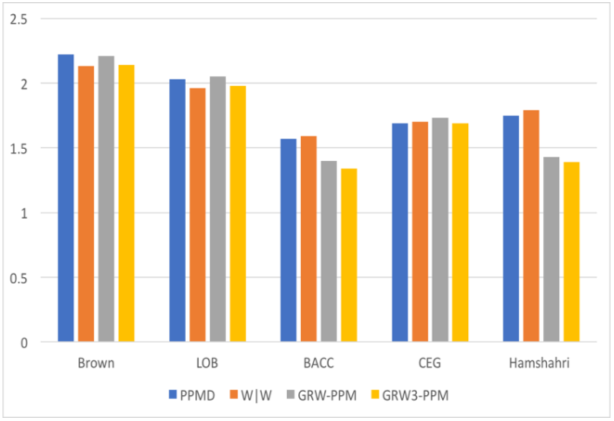


Fig. 3. Comparing the compression performance between the various methods for different languages.

V. Conclusions

In this paper, a new word-based grammar scheme (GRW-PPM) has been described for compressing natural language text. Our method creates a context-free grammar by replacing words and repeated sequences of digits, spaces and punctuation represented as non-terminal symbols in the text as it is processed from beginning to end in a single pre-processing pass. The PPM text compression algorithm is then used as the compression algorithm to encode the sequences of non-terminal sequences once they have been constructed for the whole text. Unlike PPM which is an online method, our method is off-line during the phase which generates the grammar.

In our experimental evaluation, GRW-PPM (and further such as variants GRW2-PPM and GRW3-PPM) have been compared with other well-known schemes on various language corpora for the English, Welsh, Arabic and Persian languages. The best performing scheme for the languages that use Arabic script (Arabic and Persian) is GRW3-PPM, followed by the previous best performing word-based PPM models (W|W) then the standard character-based PPMD scheme. For the English language, our experiments show that the word-based PPM models (W|W) is the best compared with standard PPM and GRW-PPM. For Welsh text, the best results are achieved using the standard character-based PPMD scheme and GRW3-PPM. Also, GRW3-PPM significantly outperforms GRW-PPM itself for different languages.

References

[1] J. Cleary and I. Witten, "Data compression using adaptive coding and partial string matching," Commun. IEEE Trans., vol. 32, no. 4, pp. 396–402, 1984.

[2] A. Moffat, "Implementing the PPM data compression scheme," IEEE Trans. Commun., vol. 38, no. 11, pp. 1917–1921, 1990.

[3] P. Howard, "The design and analysis of efficient lossless data compression systems," Ph.D. dissertation, Dept. Comput. Sci.,Brown Univ., Providence, RI, Jun. 1993.

[4] J. Cleary and Teahan, W. "Unbounded Length Contexts for PPM," Computing Journal, vol. 40, nos. 2 and 3, pp. 67–75, Feb. 1997.

[5] C. Bloom. "Solving the problems of context modeling." Informally published report, see http://www.cbloom.com/papers. 1998.

[6] D. Shkarin. "PPM: One step to practicality". Proc. Data Compression Conference, pp. 202-211, 2002. IEEE.

[7] I. H. Witten and T. C. Bell. The zero-frequency problem: Es- timating the probabilities of novel events in adaptive text compression. IEEE Transactions on Information Theory, 37(4):1085–1094, 1991.

[8] W. Teahan, "Modelling English text," Ph.D. dissertation, School of Computer Science, University of Waikato, 1998.

[9] W. Francis, W. and Kucera, H. "Brown corpus manual." Brown University. 1979.

[10] S. Johansson. "The tagged LOB Corpus: User´s Manual." 1986.

[11] W. Teahan and K. Alhawiti,"pre-processing for PPM: Compressing UTF-8 encoded natural language text," Int. J. Comput., vol. 7, no. 2, pp. 41–51, Apr. 2015.

[12] L. Al-Sulaiti and E. S. Atwell. The design of a corpus of contem- porary Arabic. International Journal of Corpus Linguistics, 11(2):135– 171, 2006.

[13] A. Ahmad et al., "Hamshahri: A standard Persian text collection," Knowledge-Based System, vol. 22, no. 5, pp. 382–387, 2009.

[14] C. Nevill-Manning and I. Witten,"Identifying hierarchical structure in sequences: A linear-time algorithm," J. Artif. Intell. Res.(JAIR), vol. 7, pp. 67–82, 1997.

[15] Teahan, W. J. and Aljehane, N. O., "Grammar-Based Pre-Processing for PPM," IJCSIT, vol. 9, no. 1, 2017.

[16] N. C. Ellis et al., "Cronfa Electroneg o Gymraeg (CEG): a 1 million word lexical database and frequency count for Welsh," 2001.

[17] T.C. Bell, J.G. Cleary, and I.H. Witten. Text Compression. Prentice Hall, New Jersey, 1990.

[18] ISO/IEC JTC1/SC29/WG1 N339. Xerox Proposal for JBIG2 Coding, June 1996.

[19] Y. Ye and P. Cosman, "Fast and memory efficient text image compression with JBIG2," IEEE Trans. image Process., 2003.