GRAMMAR-BASED PRE-PROCESSING FOR PPM

William J. Teahan[1]and Nojood O. Aljehane[2]

[1]Department of Computer Science,University of Bangor Bangor, UK
[2]Department of Computer Science,University of Tabuk,Tabuk, Saudi Arabia

ABSTRACT

In this paper, we apply grammar-based pre-processing prior to using the Prediction by Partial Matching (PPM) compression algorithm. This achieves significantly better compression for different natural language texts compared to other well-known compression methods. Our method first generates a grammar based on the most common two-character sequences (bigraphs) or three-character sequences (trigraphs) in the text being compressed and then substitutes these sequences using the respective non-terminal symbols defined by the grammar in a pre-processing phase prior to the compression. This leads to significantly improved results in compression for various natural languages (a 5% improvement for American English, 10% for British English, 29% for Welsh, 10% for Arabic, 3% for Persian and 35% for Chinese). We describe further improvements using a two pass scheme where the grammar-based pre-processing is applied again in a second pass through the text. We then apply the algorithms to the files in the Calgary Corpus and also achieve significantly improved results in compression, between 11% and 20%, when compared with other compression algorithms, including a grammar-based approach, the Sequitur algorithm.

KEYWORDS

CFG, Grammar-based,Preprocessing, PPM, Encoding.

1. INTRODUCTION

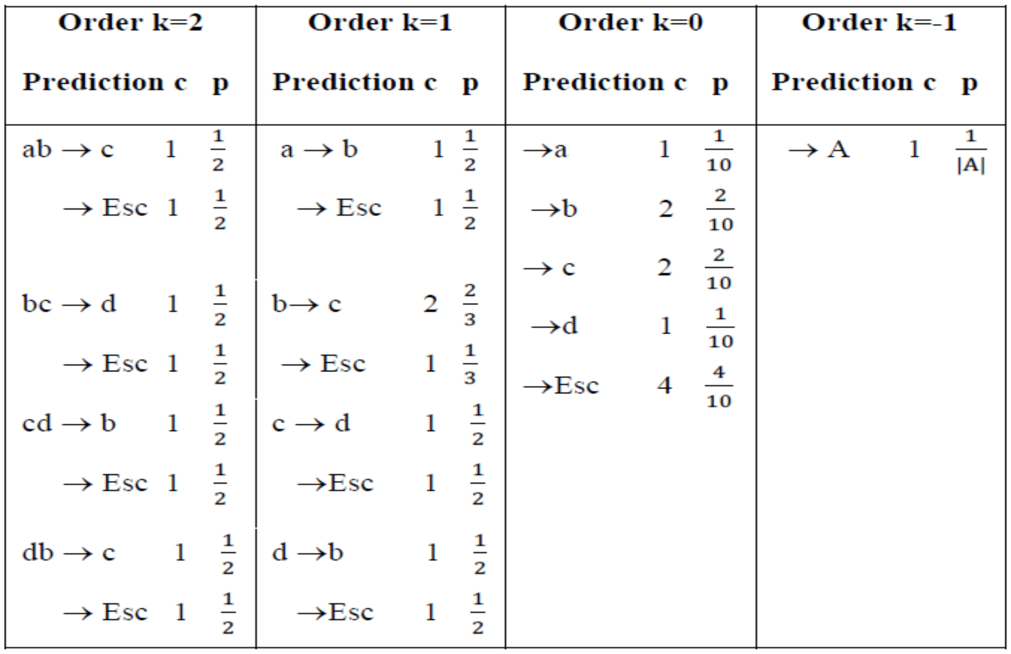
1.1. PREDICTION BY PARTIAL MATCHING

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]. The prediction of PPM depends on the bounded number of previous characters or symbols. In PPM, to predict the next character or symbol, different orders of models are used, starting from the highest orders down to the lowest orders. An escape probability estimates if a new symbol appears in the context [1, 2]. Despite the cost of the terms of memory and the speed of execution, PPM usually attains better compression rates compared with other well-known compression methods.

One of the primary motivations for our research is the application of PPM to natural language processing. Therefore, we report in this paper results on the use of PPM on natural language texts as well as results on the Calgary Corpus, a standard corpus used to compare text compression algorithms. PPM has achieved excellent results in various natural language processing applications such as language identification and segmentation, text categorisation, cryptology, and optical character recognition (OCR) [7].

Each symbol or character in PPM is encoded by using arithmetic coding using the probabilities that are estimated by different models [8].To discuss the operation of PPM, Table 1 shows the state of different orders for PPM, where *k=2, 1, 0 and -1* after input string *"abcdbc"* has been handled. In this exampleto estimate the probability for symbols in the contexts, the PPM algorithm starts from the highest order *k=2*. To estimate the probability of the upcoming symbol or character, if the context predicts the next symbol or character successfully, the associated probability for this symbol or character will be used to encode it. Otherwise, the probability of the escape will be estimated to let the encoder move down to the next highest order which is in this case is*k=1*until the encoder reaches (if needed) the lowest order which is *k=-1*. Then the probability for all symbols or characters will be estimated and encoded by$\frac{1}{|A|}$where*A* is the size of alphabets in the contexts. The experiments show the maximum order that usually gets good compression rates for English is five [1][8][7]. For Arabic text, the experiments show that order seven the PPM algorithm gives a good compression rate [9].

TABLE 1: PPMC model after processing the string "*abcdbc*".



Order k=2			Order k=1			Order k=0			Order k=-1		
Prediction	**c**	**p**	**Prediction**	**c**	**p**	**Prediction**	**c**	**p**	**Prediction**	**c**	**p**
ab → c	1	$\frac{1}{2}$	a → b	1	$\frac{1}{2}$	→ a	1	$\frac{1}{10}$	→ A	1	$\frac{1}{\vert A\vert}$
→ Esc	1	$\frac{1}{2}$	→ Esc	1	$\frac{1}{2}$	→ b	2	$\frac{2}{10}$			
						→ c	2	$\frac{2}{10}$			
bc → d	1	$\frac{1}{2}$	b → c	2	$\frac{2}{3}$	→ d	1	$\frac{1}{10}$			
→ Esc	1	$\frac{1}{2}$	→ Esc	1	$\frac{1}{3}$	→ Esc	4	$\frac{4}{10}$			
cd → b	1	$\frac{1}{2}$	c → d	1	$\frac{1}{2}$						
→ Esc	1	$\frac{1}{2}$	→ Esc	1	$\frac{1}{2}$						
db → c	1	$\frac{1}{2}$	d → b	1	$\frac{1}{2}$						
→ Esc	1	$\frac{1}{2}$	→ Esc	1	$\frac{1}{2}$						

For example, if *"c"* followed the string *"abcdbc"*, the probability for an order 2 model with the input symbol *"abcdbc"* would be $\frac{1}{2}$ because a successful prediction for the clause *"ab→c"*can be made. Suppose the string "*a*" follows the input string *"abcdbc"* instead.The probability of $\frac{1}{2}$ for an escape for the order 2 model would be encoded arithmetically, and the process of encoding downgrades from order 2 model to the next order 1 model. Moreover, in order 1, the encoder does not predict string *"a"*, so another escape (with probability $\frac{1}{2}$) is going to be encoded, and the process of encoding downgrades to order 0. In order 0, the probability is willbe $\frac{1}{10}$ for string *"a"*.Therefore, to encode string "*a*", the total probability is $\frac{1}{2} * \frac{1}{2} * \frac{1}{10} = \frac{1}{40}$, which is 5.2 bits.

If a previously unseen character new string *"n"* follows the input string "*abcdbc*", the process encodes down to the lowest order which is *k=-1* model. In this model *k=-1*, all strings are encoded with the same probabilities which is $\frac{1}{|A|}$ where A is size of alphabet. Supposing the size of the alphabet is 256, so the probability for the new string *"n"* is $\frac{1}{256}$ for the order -1 model. Therefore, the character "*n*" is encoded using the probability $\frac{1}{2} * \frac{1}{2} * \frac{4}{10} * \frac{1}{256}$ which is 11.4 bits.

As our method uses both grammar-based and text pre-processing techniques prior to compression, we will now briefly discuss related work in these two areas.

1.2.GRAMMAR-BASED COMPRESSION ALGORITHMS

Grammar-based compression algorithms depend on using a context-free grammar (CFG) to help compress the input text. The grammar and the text is compressed by arithmetic coding or by different statistical encoders [10]. Two examples of grammar-based compression schemes are the Sequitur algorithm [11] and the Re-Pair algorithm [12].

Sequitur is an on-line algorithm that was developed by Nevill-Manning and Witten in 1997 [11]. Sequitur uses hierarchical structure as specified by a recursive grammar that is generated iteratively in order to compress the text. Sequitur depends on repeatedly adding rules into the grammar for the most frequent digram sequence (which may consist of terminal symbols that appear in the text or non-terminal symbols that have been added to the grammar previously). The rule for the start symbol *S* shows the current state of the corrected text sequence as it is being processed. For instance, a text *"abcdbcabc"* is converted into three rules, which are *S* as the start symbol and *A* and *B* as nonterminal symbols: *S→BdAB, A→bc, B→aA*.

In contrast, the Re-Pair algorithm proposed by Larsson and Moffat [12] in 2000 is off-line. Like Sequitur, Re-Pair replaces the most frequent pair of symbols with a new symbol in the source message essentially extending the alphabet. The frequencies of symbol pairs are then re-evaluated and the process repeats until there are no longer any pair of symbols that occur twice. Through this off-line process, what can be considered to be a dictionary has been generated, and then an explicit representation of this dictionary is encoded as part of the compressed message.

1.3.TEXT PRE-PROCESSING FOR DATA COMPRESSION

Abel and Teahan [13] discuss text pre-processing techniques that have been found useful at improving the overall compression for the Gzip, Burrows-Wheeler Transform (BWT) and PPM schemes. The techniques work in such a way that they can easily be reversed while decoding in a post-processing stage that follows the decompression stage. The methods discussed include a long list of prior work in this area and various new techniques, and presents experimental results that show significant improvement in overall compression for the Calgary Corpus. The methods most similar to our method described in this paper are the bigraph replacement scheme described by Teahan [7] and the token replacement scheme described by Abel and Teahan [13].

The main contribution of the work described in this paper is the improved pre-processing method for PPM. This is due to the discovery that instead of using a fixed set of bigraphs for replacement from a standard reference source (such as the Brown corpus),significantlybetter compression results can be achieved by using bigraphs obtained from the text being compressed itself.

The rest of the paper is organised as follows. Our new approach is discussed in the next section. Then we discuss experimental results on natural language texts and the Calgary Corpus by comparing how well the new scheme performs compared to other well-known methods. The summary and conclusions are presented in the final section.

2. GRAMMAR BASED PRE-PROCESSING FOR PPM (GR-PPM)

In this section, a new off-line approach based on Context Free Grammars (CFG) is presented for compressing text files. This algorithm, which we call *GR-PPM* (which is short for *Grammar based pre-processing for PPM*), uses both CFGs and PPM as a general-purpose adaptive compression method for text files.

In our approach, we essentially use the *N* most frequent *n*-graphs (e.g. bigraphs when *n*=2) in the source files to first generate a grammar with one rule for each n-graph. We then substitute every time one of these *n*-graphs occurs in the text with the single unique non-terminal symbol as specified by its rule in the grammar in a single pass through the file. (Further schemes described below may use multiple passes to repeatedly substitute commonly occurring sequences of *n*-graphs and non-terminal symbols). This is done during the pre-processing phase prior to the compression phase in a manner that allows the text to be easily regenerated during the postprocessing stage. For bigraphs, for example, we call the variant of our scheme *GRB- PPM (Grammar Bigraphs for PPM)*. Our new method shows good results when replacing the *N* most frequent symbols for bigraphs (for example, when *N=100*) but also using trigraphs (when *n=3*) in a variant which we have called *GRT-PPM (Grammar Trigraphs for PPM).*

Each natural language text contains a high percentage of common *n*-graphs which comprise a significant proportion of the text [7]. Substitution of these *n*-graphs 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 two characters (bigraphs) that often repeat in the same order in many different words, such as in the English *"th", "ed"*and *"in"*, and for the Arabic language, such as " ال", "في" and "لا " and so on.

The frequencies of common sequences of characters in reference corpora (such as the Brown Corpus for American English [14] and the LOB Corpus for British English [15]) can be used to define the *n*-graphs that will be substituted (without the need to encode the grammar separately, making it possible to have the algorithm work in an online manner rather than offline). However, although this method can be quite effective, what we have found to be most effective for our scheme is to determine the list of *n*-graphs that define the grammar in a single or double pass through the text being compressed prior to the compression phase, and then encoding the grammar separately along with the corrected text which is encoded using PPM.

Our method replaces common sequences similar to both Sequitur and Re-Pair, but unlike them, this is not done iteratively on the current most common digram sequence or phrase, but is done by replacing the most common sequences as the text is processed from beginning to end in a single pass (although further passes may occur later). Also, the PPM algorithm is used as the encoder once the common sequences have been replaced whereas Re-Pair uses a dictionary based approach for the encoding stage. Like Re-Pair, our method is only off-line during the phase which generates the grammar.

Our approach adapts the bigraph replacement text pre-processing approach of Teahan [7] by using an offline technique to generate the list of bigraphs first from the source file being compressed. This approach is considered within a grammar-based context, and the approach is further extended by considering multiple passes and recursive grammars.

Figure 1 shows the whole process of GR-PPM. First, the CFG will be generated from the original source files by taking the *N* most frequent *n*-graphs and replacing them with the non-terminal symbols as defined by their rules in the grammar. After rules are produced, the sender will do grammar-based pre-processing to correct the text. Then, the corrected text is encoded by using PPMD, and the resulting compressed text is then sent to the receiver. The receiver then decodes the text by using PPMD to decompress the compressed file that was sent. Grammar-based post-processing then facilitates the reverse mapping by replacing the encoded non-terminal symbols with the original *n* characters or *n*-graphs.

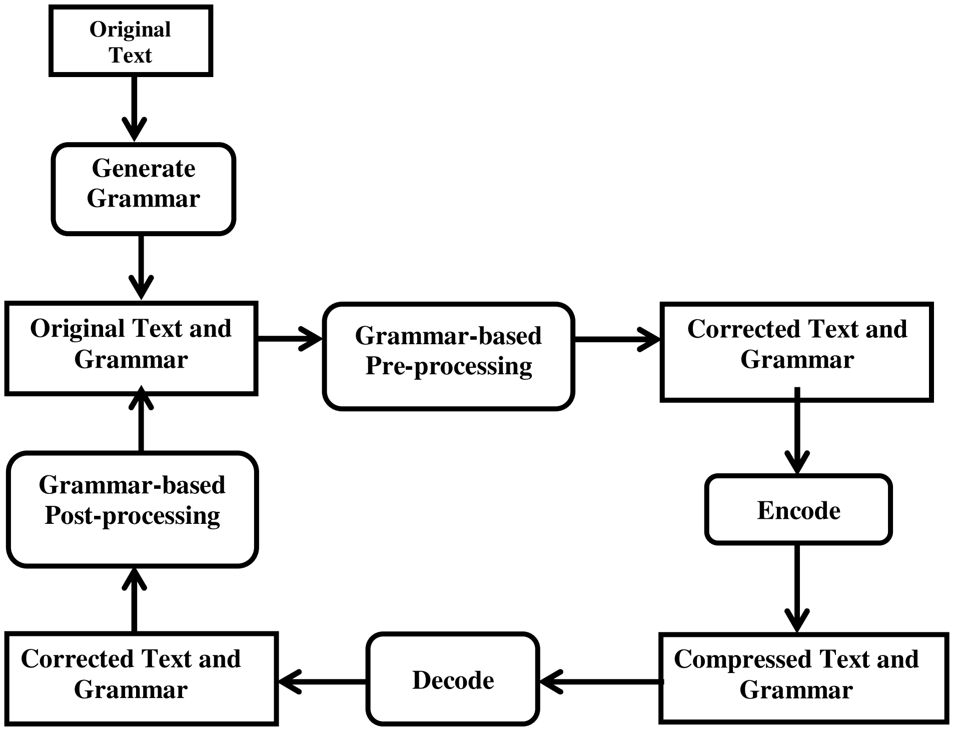

Original Text
Generate Grammar
Original Text and Grammar
Grammar-based Pre-processing
Corrected Text and Grammar
Encode
Compressed Text and Grammar
Decode
Corrected Text and Grammar
Grammar-based Post-processing

Figure 1: The complete process of grammar based pre-processing for Prediction by Partial Matching (GR-PPM).

One final step is the need to encode the grammar so that it is also known by the receiver since, as stated, we have found we can achieve better compression by encoding separately a grammar generated from the source file itself rather than using a general grammar (known to both sender and receiver) that was generated from a reference corpus. We choose to use a very simple method for encoding the grammar—simply transmitting the lists of bigraphs or trigraphs directly. (So encoding a grammar containing N = 100 rules for the GRB-PPM scheme where n = 2 and character size is 1 byte (for ASCII text files, say), this will incur an overhead of $N \times n$ bytes or 200 bytes for each file being encoded).

Table 2 illustrates the process of GRB-PPM using a line taken from the song by Manfred Mann (one published paper uses this song as a good example of repeating characters [12]). The sequence is *"do wah diddy diddy dum diddy do"*. For GRB-PPM, for example, there are five bigraphs which are repeated more than once in the first pass through the text: *in, do, di* and *dd* . Thus, these bigraphs will be substituted with new non-terminal symbols, say *A, B, C* and *D,* respectively, and included in the grammar (for example, if we choose N = 4). Note that substitution is performed as the text is processed from left to right. If a bigraph is substituted, the process will move to the character following the bigraph before continuing.

Table 2: An example of how GR-PPM works for a sample text.

Pass	Grammar	String & Corrected Strings
		singing.do.wah.diddy.diddy.dum.diddy.do
1st	A→in, B→do, C→di, D→dd	sAgAg.B.wah.CDy.CDy.dum.CDy.B
2nd	E→Ag, F→CD	sEE.B.wah.Fy.Fy.dum.Fy.B

In *GRBB-PPM (Grammar Bigraph to Bigraph for PPM),* new rules are formed in a second pass through the text in the corrected text that resulted from the first pass for the further bigraphs, *Ag, CD*. These will then be represented with non-terminals *E* and *F* that are added to the expanded grammar. After all bigraphs have been substituted, the message is reduced to the new sequence *sEE.B.wah.Fy.Fy.dum.Fy.B.*

Note that we ignore spaces and any punctuation characters because based on our experiments, including these symbols decreases the compression rate. Moreover, the grammar will be transmitted to the receiver with the original text after all bigraphs are substituted in the original text with their non-terminal symbols. In the above example, it is clear that the number of symbols is reduced from 31 symbols in the original text to 23 symbols in the first pass (GRB-PPM) and to 20 symbols in the next pass (GRBB-PPM).

The grammar in both GRB-PPM and GRT-PPM share the same characteristic, which is that no pair of characters appears in the grammar more than once. This property ensures that every bigraph in the grammar is unique, a property called *non-terminal uniqueness* using the same terminology proposed by Neville-Manning and Witten [11]. 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. These two features are in the grammar that GR-PPM generates and are discussed in more detail next.

2.1. NON-TERMINAL UNIQUENESS

For the more general case (i.e. considering *n*-graphs, not just bigraphs), each *n*-graph has to appear only once in the grammar and is also substituted once by a non-terminal symbol. To prevent the same *n*-graph from occurring elsewhere in the corrected text sequence, each *n*-graph is substituted based on the *n*-graph that will be generated by the algorithm. In the example of Table 2, the list of most frequent bigraphs that form the grammar are added at the same time as the text is processed in each pass. For example, when *di* appears in the text more than once, the new non-terminal symbol *A* is substituted. On the other hand, in the Sequitur algorithm [11], only the most frequent digram (i.e. bigraph using our terminology) are added incrementally to the grammar.

2.2.RULE UTILITY

Every rule in the grammar should be used more than one time to ensure that the rule utility constraint is applied. When *di* appears in the text more than once, the new non-terminal symbol *C* is substituted. In our approach, rule utility does not require the creating and deleting of rules, which makes the rules more stable. This method retains the tracking of long files, thus avoiding the requirement of exterior data structures. However, in the Sequitur algorithm, when the new letter in the input appears in first rule *S*, the grammar creates a new rule and deletes the old digram [11]. The process of deleting and creating of the rules in the grammar each time when the new symbol appears is unstable and inefficient. In the Sequitur approach, the grammar is being dynamically created. To avoid separate data structures, we apply a multi-pass approach for GR-PPM and add multiple symbols to the grammar at the same time and this causes a greater stability and efficiency.

2.3.Hierarchical Grammatical Structures

In order to further illustrate our method, Figures 2a and 2b show the hierarchical grammatical structures that are generated by our approach for two different languages, English and Arabic. The hierarchical grammatical structures are formed based on the most frequent two characters or bigraph in each text. For example, in Figure 2a, the word *the* is split into *th* and *e (th* is the bigraph in GRB-PPM, and the non-terminal that represents it and the letter *e* forms the second bigraph for the next pass for GRBB-PPM), and so on for other words in the texts.

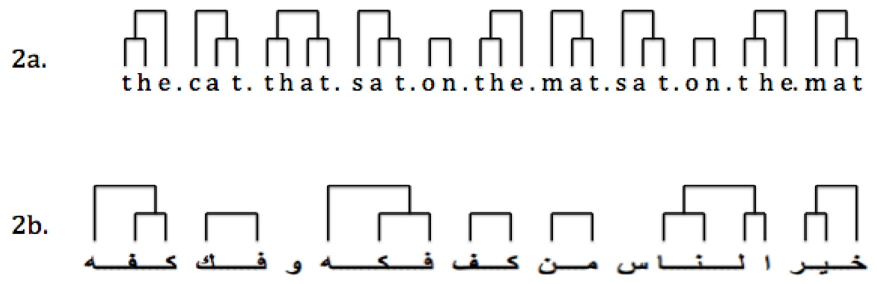

2a.
t h e . c a t . t h a t . s a t . o n . t h e . m a t . s a t . o n . t h e . m a t
2b.
خ ي ر ا ل ن ا س م ن ك ف ف ك ه و ف ك ك ف ه

Figure 2: Hierarchical structure in the grammars generated by our algorithm for sample sequences in two languages: (a) English (b) Arabic.

The same algorithm generates the Arabic version in Figure 2b, where the word كفهis also split into كف and هin a similar way (كف is the bigraph and هis the second bigraph). We use bullets for spaces to make them more visible, but nevertheless, spaces will not be considered in our algorithms and structures. On the other hand, in the Sequitur algorithm, spaces are part of the algorithm, which means it is also part of the grammatical structures generated by the algorithm.

```
1  For each pass up to P
2     Find the N most frequent bigraphs in the current sequence
       that do not contain a space or a punctuation character
3     For each of the N most frequent n-graphs do
4         Substitute the new n-graph with its non-terminal symbol
5     End for
6  End for
7  Use PPMD to encode the corrected sequence
```

Figure 3: Pseudo-code for the GR-PPM algorithm.

Figure 3 summarises the algorithm using pseudo-code. Line 1 is for a loop to define how many passes the algorithm performs (from 1 up to a maximum of *P* passes). Lines 2 to 4 are for a loop to find the *N* most frequent bigraphs and substitute them with non-terminal symbols. Lines 3 through 5 implement the bigraphs utility constraints. Line 7 compresses the final text file by using PPMD after substituting *N* bigraphs.

3. EXPERIMENTAL RESULTS

This section discusses experimental results for variants of our method GR-PPM de-scribed above for compression of various text files compared with other well-known schemes.

We have found that variants of the GR-PPM algorithm achieve the best compression ratio for texts in different languages, such as English, Arabic, Chinese, Welsh and Persian. Also, we have compared the results with those of different compression methods that are known to obtain good results, including Gzip and Bzip2. BS-PPM uses order 4 PPMD to compress UTF-8 text files and is a recently published variant of PPM that achieves excellent results for natural language texts [9]. For both GRB-PPM and GRT-PPM, we use the 100 most frequent bigraphs or trigraphs and order 4 PPMD for the encoding stage.

Table 3 compares the results of using GRB-PPM and GRT-PPM (using N = 100 and order 4 PPMD) with different versions of well-known compression methods, such as Gzip, Bzip2 and BS-PPM order 4. (PPMD has become the de facto standard for comparing variants of the PPM algorithm, and so is included with our results listed here.) We do these experiments with different text files in different languages, including American English, British English, Arabic, Chinese, Welsh and Persian. We use the Brown corpus for American English and LOB for British English. For Arabic, we use the BACC [16]. For Persian, the Hamshahri corpus is used [17]. The LCMC corpus is used for Chinese [18] and the CEG corpus is used for Welsh [19].

It is clear that GRB-PPM achieves the best compression rate (shown in bold font) in bits per character (bpc) for all cases in different languages. Also, GRB-PPM is significantly better than various other compression methods. For instance, for Arabic text, GRB-PPM shows a nearly 45% improvement over Gzip and approximately 15% improvement over Bzip2. For Chinese, GRB-PPM shows a 36% improvement over BS-PPM and 38% improvement over Gzip. For the Brown corpus, GRB-PPM shows nearly a 35% improvement over Gzip and approximately 15% improvement over Bzip2. For the LOB corpus, GRB-PPM shows a 36% improvement over Gzip and 15% improvement over Bzip2. For Welsh, GRB-PPM shows a 31% improvement over BS-PPM and 48% improvement over Gzip. For Persian, GRB-PPM shows nearly a 50% improvement over Gzip and approximately 22% improvement over Bzip2.

GRBB-PPM and GRTT-PPM, which are the second passes of GRB-PPM and GRT-PPM respectively, achieve better compression ratios than their single pass variants for all the different language texts, such as English, Arabic, Welsh and Persian. In GRBB-PPM and GRTT-PPM, we use the 100 most frequent bigraphs for both passes and order 4 PPMD for the encoding stage as before.

Table 3: GRB-PPM and GRT-PPM compared with other compression methods for different natural language texts.

File	**Language**	**Size**	**Gzip**	**Bzip2**	**PPMD4**	**BS-PPM**	**GRT-PPM**	**GRB-PPM**
			(bpc)	**(bpc)**	**(bpc)**	**(bpc)**	**(bpc)**	**(bpc)**
Brown	American English	5968707	3.05	2.33	2.14	2.11	2.03	**1.99**
LOB	British English	6085270	2.95	2.22	2.03	2.10	1.92	**1.88**
LCMC	Chinese	5379203	2.58	1.64	1.61	2.49	1.61	**1.59**
BACC	Arabic	69497469	2.14	1.41	1.68	1.32	1.29	**1.21**
Hamshahri	Persian	53472934	2.58	1.64	1.68	1.25	1.31	**1.22**
CEG	Welsh	6753317	2.91	1.95	1.69	2.20	1.57	**1.51**
Average			2.14	1.86	1.80	1.91	1.62	**1.56**

Table 4 shows the results of the different single pass and double pass variants of GR-PPM. The results for the single pass variants GRB-PPM and GRT-PPM have been included from Table 3 for ease of comparison as well as the results of the grammar-based compression algorithm Sequitur. It is clear that GRBB-PPM achieves the best compression rate (bpc) for almost all cases in the different language texts with only the single result on the Chinese text, the LCMC corpus, being better for the Sequitur algorithm. For Arabic, GRBB-PPM shows 29% improvement over PPMD and a nearly 19% improvement over the Sequitur algorithm. For American English, Brown GRBB-PPM shows 8% improvement over PPMD and a nearly 23% improvement over Sequitur. For British English, LOB GRBB-PPM shows a 20% improvement over the Sequitur algorithm. For Welsh, GRBB-PPM shows 12% improvement over PPMD and 27% improvement over the Sequitur algorithm. For Persian, GRBB-PPM shows 27% improvement over PPMD and a nearly 14% improvement over Sequitur.

Table 4: Variants of GR-PPM compared with other compression methods for different natural language texts.

File	**PPMD Order4**	**Sequitur**	**GRT-PPM**	**GRTT-PPM**	**GRB-PPM**	**GRBB-PPM**
	(bpc)	**(bpc)**	**(bpc)**	**(bpc)**	**(bpc)**	**(bpc)**
Brown	2.14	2.55	2.03	2.00	1.99	**1.97**
LOB	2.03	2.34	1.92	1.90	**1.88**	**1.88**
LCMC	1.61	**1.45**	1.61	1.61	1.59	1.59
BACC	1.68	1.47	1.29	1.29	1.21	**1.20**
Hamshahri	1.68	1.42	1.31	1.31	**1.22**	**1.22**
CEG	1.69	2.04	1.57	1.54	1.51	**1.49**
Average	1.80	1.87	1.62	1.60	1.56	**1.55**

Table 5 shows the compression rate for PPMC, Sequitur [20], Gzip, GRB-PPM and GRBB-PPM on the Calgary corpus. Overall, the GRBB-PPM algorithm outperforms all the well-known compression methods. For the Sequitur algorithm, GRBB-PPM shows on average a nearly 19% improvement and on average a 17%, 12% and 1% improvement over Gzip, PPMC and GRB-PPM, respectively. Although GRBB-PPM achieves similar results on the *book1, book2, news* and *pic* files compared to GRB-PPM, GRBB-PPM is better than GRB-PPM for the other files.

Table 5: Performance of various compression schemes on the Calgary Corpus.

File	**Size**	**PPMC**	**Gzip**	**Sequitur**	**GRB- PPM**	**GRBB-PPM**
	(bytes)	**(bpc)**	**(bpc)**	**(bpc)**	**(bpc)**	**(bpc)**
bib	111261	2.12	2.53	2.48	1.87	**1.85**
book1	768771	2.52	3.26	2.82	**2.25**	**2.25**
book2	610856	2.28	2.70	2.46	**1.91**	**1.91**
news	377109	2.77	3.07	2.85	**2.32**	**2.32**
paper1	53161	2.48	2.79	2.89	2.34	**2.32**
paper2	82199	2.46	2.89	2.87	2.29	**2.26**
pic	513216	0.98	0.82	0.90	**0.81**	**0.81**
progc	39611	2.49	2.69	2.83	2.36	**2.33**
progl	71646	1.87	1.81	1.95	1.66	**1.61**
progp	49379	1.82	1.82	1.87	1.70	**1.64**
trans	93695	1.75	1.62	1.69	1.48	**1.45**
Average		2.14	2.29	2.32	1.91	**1.88**

4. SUMMARY AND CONCLUSIONS

In this paper, we have described new algorithms for improving the compression for different natural language texts. These algorithms work by substituting a repeated symbol (bigraph or trigraph) with a non-terminal symbol from a grammatical rule in a CFG before using PPM to compress the text files. These algorithms are maintained by two constraints, which are non-terminal uniqueness and rule utility. These techniques also work well as good compression methods for general text files.

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

[8] Witten, I., Neal, R. & Cleary, J. "Arithmetic coding for data compression". Communications of the ACM, vol. 30 Issue 6, June 1987.

[9] W. Teahan and K. M. Alhawiti, "Design, compilation and preliminary statistics of compression corpus of written Arabic," Technical Report, Bangor University, School of Computer Science, 2013.

[10] J. Kieffer and E. Yang, "Grammar-based codes: a new class of universal lossless source codes," Inf. Theory, IEEE Trans., vol. 46, no. 3, pp. 737–754, May. 2000.

[11] 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.
[12] N. Larsson and A. Moffat, "Off-line dictionary-based compression," Proc. IEEE, vol. 88, pp. 1722–1732, Nov. 2000.
[13] J. Abel and W. Teahan, "Universal text pre-processing for data compression," IEEE Transactions on Computers, 54.5: 497-507, 2005.
[14] W. Francis, W. and Kucera, H. "Brown corpus manual." Brown University. 1979.
[15] S. Johansson. "The tagged LOB Corpus: User ́s Manual." 1986.
[16] 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.
[17] Ale Ahmad et al., "Hamshahri: A standard Persian text collection," Knowledge-Based System, vol. 22, no. 5, pp. 382–387, 2009.
[18] A. McEnery and Z. Xiao, "The Lancaster Corpus of Mandarin Chinese: A corpus for monolingual and contrastive language study," Religion, vol. 17, pp. 3–4, 2004.
[19] N.C. Ellis et al., "Cronfa Electroneg o Gymraeg (CEG): a 1 million word lexical database and frequency count for Welsh," 2001.
[20] C. Nevill-Manning and I. Witten, "Compression and explanation using hierarchical grammars," Comput. J., vol. 40, no. 2/3, pp 103–116, 1997.