

# Compressed Context Modeling for Text Compression

M. Oğuzhan Külekci

TÜBİTAK – BİLGEM – UEKAE

National Research Institute of Electronics & Cryptology, Turkey.

kulekci@uekae.tubitak.gov.tr

## Abstract

In text compression, statistical context modeling aims to construct a model to calculate the probability distribution of a character based upon its context. The order- $k$  context of a symbol is defined as the string formed by its preceding  $k$  symbols. This study introduces compressed context modeling, which defines the order- $k$  context of a character as the sequence of  $k$ -bits composed of the entropy compressed representations of its preceding characters. While computing the compressed context of a symbol at some position in a given text, enough number of characters are involved in the compressed context so as to produce  $k$ -bits of information. Thus, instead of certain number of characters, certain amount of *information* is considered as the context of a character, and this property enables the prediction of each character to be performed with nearly uniform amount of information. Experiments are conducted to compare the proposed modeling against the classical fixed-length context definitions. The files in the large Calgary corpus are modeled with the newly introduced compressed context modeling and with the classical fixed-length context modeling. It is observed that on the average the statistical model with the proposed method uses 13.76 percent less space measured according to the number of distinct contexts, while providing 5.88 percent gain in empirical entropy measured by the information content as bits per character.

## Keywords

Context Modeling, PPM, Text Compression, Compressed Context Modeling

## I. INTRODUCTION

Prediction with partial string matching (PPM) method, which was originally introduced by Cleary & Witten [1], has been studied extensively during the last two decades yielding the most successful compression rates. The basic idea is to predict the next symbol based on the preceding symbols. The algorithms originating from this main idea are generally classified as context-based compression algorithms [2]. Some sophisticated extensions of the PPM methodology have reported the best compression rates especially on natural language texts [3], [4].

The two phases of context-based compression algorithms are *context modeling* and *coding*. Context modeling process [5], [6], [3] aims to construct a statistical model to calculate the probability distribution of a character based upon its context. The character is then encoded according to its probability in that distribution via a selected coding scheme, such as the arithmetic coding [7]. The compression rate is highly related with the success of the context modeling since better description of the source data yields more accurate statistics, which in turn improves the efficiency of the coding.

Given text  $T$  of length  $n$  as  $T = t_1 t_2 \dots t_n$ , the order- $k$  context of the symbol  $t_p$ , where  $k < p \leq n$ , is defined as the string  $t_{p-k} \dots t_{p-2} t_{p-1}$  formed by its preceding  $k$  symbols. Context modeling approaches differ due to the length of context used in the compression. The compressors using finite-length context models [8], [9], [10] set  $k$  to a predetermined value at the beginning of the compression process. The frequencies of the characters succeeding the order- $k$  context up to the current symbol determines the value generated by the encoder for that symbol. The statistics is updated after each encoding step.

On the other hand, some compressors such as PPM\* [11], [12] and PPMZ [13] use *deterministic* context, which was introduced by Cleary *et al.* [11], [12]. For some position  $p \leq n$ , the  $u$  character long string  $D = t_{p-u} \dots t_{p-2} t_{p-1}$ ,  $p > u$ , is the deterministic context of  $t_p$ , if and only if, for all positions in  $T$  up to  $p$ ,  $D$  is always followed by the same symbol  $t_p$ . According to this definition,  $u$  can take any value between 1 to  $p$ . Thus, the deterministic context is unbounded. The compressors based on *unbounded-length* context tries to find the deterministic context at each position. If no such deterministic context exists for the current  $t_p$ , they switch to regular fixed-length context procedure.

This study proposes a novel context modeling technique, which defines the context of the  $t_p$  as the sequence of  $k$ -bits that is composed of the entropy compressed representations of its preceding characters, where  $k$  is a fixed predetermined value. More formally, the first  $k$ -bits of  $comp(t_{p-1} t_{p-2} \dots t_{p-z})$  is proposed to be used as the order- $k$  context of  $t_p$ , assuming  $comp(S)$  corresponds to the bit stream holding the entropy compressed representation of string  $S$  and  $t_{p-z} \dots t_{p-1}$  is long enough to have more than  $k$  bits in the compressed form. The novelty comes from the fact that; unlike the other modeling techniques that are based on the previous  $k$ -**symbols**, the introduced compressed context modeling considers  $k$ -**bits** of previous information. The amount of information in the preceding symbols is measured according to Shannon's information theory [14]. We can easily compute the information content of the previous characters by examining their compressed representations since compression by itself aims to represent sequences with least number of bits according to their entropy. In other words, compressed context modeling (CCM) scheme no more defines the context of a symbol a sequence of its preceding characters, but instead a sequence of bits that encode the previous characters according to their entropy.

**The Contribution:** The advantages of the CCM technique may be investigated under two issues as gain in entropy and gain in space. On equal context length, the average empirical entropy of an input file measured under CCM is less than the one calculated under the corresponding fixed-length context modeling (FLCM). As an example, experiments conducted on large Calgary corpus<sup>1</sup> showed that, the average empirical entropy of a file under order-16 *bit* CCM is approximately 50% less than that of the same file under order-2 FLCM, which indicates the compression is expected to be better if CCM is used. This is due to the fact that CCM stores much more information than the fixed-length model in two bytes (16 bits).

On the other hand the number of observed distinct contexts in a file, which greatly determines the space consumption of a PPM scheme, is larger in CCM than that of in FLCM when equal context lengths are considered. For example, on the above settings (context length = 2-bytes = 16-bits) the number of distinct contexts under CCM is four times larger than that of under the FLCM model. Thus, the next question is whether it is profitable to trade entropy with space. To answer that question, a second measurement is done on the test files. Order- $k$  FLCM is compared with order- $\ell$  CCM, where  $\ell$  is the *largest* number chosen such that the number of distinct contexts observed in order- $\ell$  CCM is *less* than that of the order- $k$  FLCM. As an example, the  $\ell$  value computed against order-2 FLCM on file *book1* of Calgary corpus is 11 since the number of distinct contexts are 1775 and 1826 on CCM and FLCM schemes respectively. The averages computed over large Calgary corpus showed that CCM introduces a 13.76 percent gain in space, while still preserving an advantage of 5.88 percent in entropy over the FLCM.

**Outline:** After describing the basic motivation in section 1, Section 2 surveys problems regarding to the context modeling in fixed-length and unbounded-length models. Section 3 describes the proposed compressed context modeling concept in detail, followed by the section 4 giving the experimental results

<sup>1</sup> Available at <http://www.data-compression.info/Corpora/CalgaryCorpus/index.htm>

obtained while comparing CCM against FLCM on the Calgary corpus. The study ends with concluding remarks on the new approach and future research directions.

## II. MOTIVATION

The main motivation behind the compressed context modeling is to have approximately equal amount of information while performing a prediction on each symbol. Sometimes just one character may say more on the value of the next than a sequence of characters. This central idea may be best explained on an example. Assume we are examining the context of letter 'u' at some position in an English text. If the previous character is 'q', it might be enough for a good statistical distribution as 'q' is generally followed by 'u' in English and most probably we will see that the distribution of characters following 'q' estimates a high probability for letter 'u', which will give us an opportunity to encode 'u' with less number of bits in this context. On the other hand, if the preceding character is 'o', we can not rely on the statistics collected up to current position from the succeeding letters of 'o', since it will not assign a probability to 'u' as high as in the 'q' context case. Hence, we need to consider more characters, and maybe three letter long context 'vio' will be merely able to give enough high probability to 'u'.

The amount of information contained in a sequence is measured using Shannon's theory [14] and compression algorithms compete to squeeze a given text to its information content. Based on this fact, the natural solution for the stated aim of this study is to use the compressed form of the previous context in the design of the statistical model. This might seem a bit tricky in the beginning since it is proposed to benefit from compression during the actual compression of a file.

The number of characters included in the compressed context of a symbol is neither fixed nor unbounded. Furthermore, it is possible that the length might be fractional, e.g., 2.3 characters. Assuming a simple zeroth order entropy model to measure the information content of the context of a character, if the preceding symbols carry high entropy, meaning that they are rare, then the number of characters involved in the context will be low. When the symbol is preceded by frequent symbols, the number of characters involved in the compressed context will be high, since frequent characters will correspond short code words.

Note also the difference between unbounded-length deterministic context and the compressed context that the later does not require the context to be *deterministic*, but tries to find enough long context where the probability distribution is skewed enough for an efficient encoding of the next symbol. This effort is in much comfort than looking for the deterministic context as compressing a couple of characters is easier than investigating the whole file until a specific point.

## III. CONTEXT MODELING PROBLEMS IN CONTEXT-BASED COMPRESSION

It will be better to first remember the general mechanism in PPM type compression, and then focus more on the difficulties arise in practice. Assume that we have finished the encoding of  $t_1 t_2 \dots t_{p-1}$  in the given sequence  $T$ , and now trying to compress next symbol  $t_p$ .

The first thing we need to examine is whether  $t_p$  has been previously preceded by its order- $k$  context  $t_{p-k} \dots t_{p-2} t_{p-1}$  up to current point  $p$ . If we find out that  $t_p$  has been previously observed to follow the string  $t_{p-k} \dots t_{p-2} t_{p-1}$ , then we use the frequency counts of this context and compute the probability of  $t_p$ . This probability is send to the coder, the frequency of  $t_p$  in the  $t_{p-k} \dots t_{p-2} t_{p-1}$  context is incremented by one, and the process is repeated for the next  $t_{p+1}$  until the end of the input text. In case of failure in locating  $t_{p-k} \dots t_{p-2} t_{p-1}$ , the probability of an *escape* symbol is estimated due to this context, and this probability is sent to the coder. After emitting the escape symbol indicating  $t_p$  has never followed this context up to position  $p$ , the statistics of the related context is updated, the context length is decremented by 1, and the same procedure is repeated with order- $(k-1)$  context  $t_{p-k-1} \dots t_{p-2} t_{p-1}$ . The coder keeps

emitting escape symbols until a match is reported by any context length down to 0. If the symbol has not been observed even in order-0 context, meaning that this is the first time symbol  $t_p$  occurs in the text, special action is taken to code its actual value.

Most of the PPM variants such as the PPMA and PPMB [1], PPMC [8], PPMP and PPMX [9], and FastPPM [10] use finite-length context modeling. The main difference among these alternative implementations is actually the way they handle the zero-frequency problem [15], which is out of the scope of this study.

On the other hand, if we prefer using unbounded-length deterministic context instead of a fixed-length context, then we need to begin with searching the deterministic context of  $t_p$ . In case such a context is detected for some length  $u$ , the coder uses the statistics of that order- $u$  context. Otherwise, the compressor switches to regular fixed-length order- $k$  execution again with a predetermined maximum  $k$  value.

The main point in all context-based compression schemes is the necessity to access the frequency counts of characters in the context  $t_{p-k} \dots t_{p-2}t_{p-1}$  for some  $p$  and  $k$  values. There are two ways to accumulate this:

The first one is to scan the previous text  $t_1t_2 \dots t_{p-1}$  and compute the frequencies dynamically at each attempt of coding  $t_p$ . This brings a time-complexity of  $O(n^2)$  since for each  $p$ ,  $1 < p \leq n$ , the previous  $p - 1$  positions should be checked resulting to the sum of  $1 + 2 + 3 + \dots + (n - 1) = (n - 1) \cdot n/2$ . The good thing about this dynamic approach is that there is no need for space as we do not store anything. On the other hand, the compression will be very slow. Although it is possible to calculate the frequencies faster with more sophisticated pattern matching techniques [16], this approach indeed becomes infeasible with the growing value of text size  $n$ .

The second way is to use a data structure to store all frequency counts corresponding to possible distinct contexts. One such structure is a table that each row corresponds to a specific context, and the columns include the frequency counts of characters in this context. After encoding of each character  $t_p$ , the related updates are performed on the table.

For the encoding process of  $t_p$  described above, the related entry in the table should be located first. This requires a search process on the table, which may benefit from various techniques such that the rows of the table may be kept sorted according to context strings, and a binary search procedure may be run to accomplish this task. Although this seems better than the  $O(n^2)$  dynamic computation approach, it still needs considerable effort to find the correct row in the table besides the need for memory. Keeping the entries sorted is another issue also.

Context-tries are used to overcome that problem. Instead of a table, a trie of depth  $k$  for order- $k$  context modeling may be used [13]. While passing over the text for the compression process, the trie is updated after encoding of each character. The nodes are composed of a label indicating the character, the frequency count due to the trie structure, and pointers to parent and children nodes to be able to traverse the trie. Such a trie gives access to an order- $k$  context in  $O(k)$  complexity, since it takes at most  $k$  steps to reach a node in a depth- $k$  trie. The disadvantage is the difficulty in programming and maintaining the complex data structure [17] besides the increase in required memory.

In practice, a third solution is more widely preferred that a linked list is used to store the statistics. Each distinct context is saved in a list node along with its statistics, where the nodes in the linked list are kept sorted. A relatively small table holding the hash values of the order- $k$  contexts is also maintained. Whenever a context is to be searched, first its hash is computed and the traversal of the linked list begins from the node pointed by the hash table. This approach does not eliminate the search process of the context, but serves a good practical solution to a faster performance. A similar implementation was given in [18].

The implementation of PPM compressors depending on unbounded-length deterministic context ([12], [13]) is much more difficult as the solutions described above are not enough to locate deterministic contexts. In such cases, the suffix tree [19] is used. However, the demand of the suffix trees for huge memories inhibits their practical usage.

In spite of these discussions, the basic difficulties faced in implementing a PPM scheme is the complex data structures demanding large memories and the search process in locating a context when linked lists or tables are preferred in storing the frequencies. Thus, space-efficient context modeling approaches enabling time-efficient access to frequency counts seem valuable. Special attention should be paid not to loose efficiency in compression rate while designing such a model. The proposed compressed context modeling scheme is an attempt to achieve these goals.

#### IV. COMPRESSED CONTEXT MODELING

Let  $B = b_1b_2 \dots b_z$  be the bit stream resulted from compressing a symbol sequence  $S = s_1s_2 \dots s_r$  of length  $r$  as  $B = comp(S)$ . Assuming text  $T = t_1t_2 \dots t_n$  is given, the order- $k$  compressed context of symbol  $t_p$  is the first  $k$  bits of the bit sequence  $B = comp(t_{p-1}t_{p-2} \dots t_{p-w})$ . The window size variable  $w$  indicating the number of preceding characters involved in  $comp()$  function is determined with an incremental approach. First  $comp(t_{p-1})$  is calculated. If the length of the resulting bit stream is equal to or greater than  $k$ , then we stop. Else, we continue with  $comp(t_{p-1}t_{p-2})$ ,  $comp(t_{p-1}t_{p-2}t_{p-3})$ ,  $\dots comp(t_{p-1}t_{p-2}t_{p-m})$ , until the length constraint is satisfied or  $m$  becomes  $p$  indicating the beginning of the text is reached. If  $p = m$  and the length of the bit string is still less than  $k$ , the order- $k$  compressed context is undefined for  $t_p$ . Thus, it is not possible to model some small number of initial characters in a file because of the lack of enough previous information.

The compression function that is to be used in CCM needs to obey the restriction that for all substrings  $A$  and  $B$  in  $T$ , if  $A = B$ , then  $comp(A) = comp(B)$  should hold. That is because we don't want to mix the frequencies of different contexts. If  $comp(A) = comp(B)$  while  $A \neq B$ , then the the probability of character  $t_p$  will be estimated according to unified frequencies collected both from different  $A$  and  $B$  contexts.

On the other hand, when  $comp(t_{p-1}t_{p-2}) \dots t_{p-w}$  produces more and  $comp(t_{p-1}t_{p-2}) \dots t_{p-w-1}$  produces less than  $k$  bits, then the the bits added to the stream by the introduction of the  $t_{p-w}$  can only be added partially to the compressed context of  $t_p$ . For example lets assume the compressed context of 't' in text 'denote' is to be computed, and it is seen that  $comp('on')$  is shorter than  $k$  while  $comp('one')$  is larger. When the first  $k$  bits are extracted from  $comp('one')$ , some of those bits residing at the end are related with character 'e'. If some other context such as 'ona' has the same initial  $k$  bits in  $comp('ona')$ , then the frequencies for the compressed context of 't' at this position will both include those letters following 'ano' and 'eno'. The  $comp()$  function accepts inputs in the reverse order of appearance, and because of this property ambiguity can only be caused by the single last symbol that is leftmost to current  $t_p$ . Thus, it is accepted in the current proposal. Note that as long as the same characters are involved and statistics are extracted properly, information theoretically, reading from right-to-left is no different than reading left-to-right.

The compression function used in CCM need not be the same with the one being used in the *coding* phase of the PPM scheme. It might be the case that the coding phase is using, say, adaptive arithmetic coding while the compressed context models are constructed with a *static* zeroth order Huffman coding.

#### V. EXPERIMENTAL RESULTS

Experiments are conducted to compare compressed context modeling versus fixed-length context modeling on large Calgary corpus. The  $comp()$  function used in CCM was selected to be a simple

Large Calgary corpus File ID List: 1-paper1, 2-paper6, 3-paper5, 4-progl, 5-paper3, 6-paper4, 7-paper2, 8-book2, 9-bib, 10-book1, 11-geo, 12-progc, 13-news, 14-progp, 15-trans, 16-pic, 17-obj2, 18-obj1

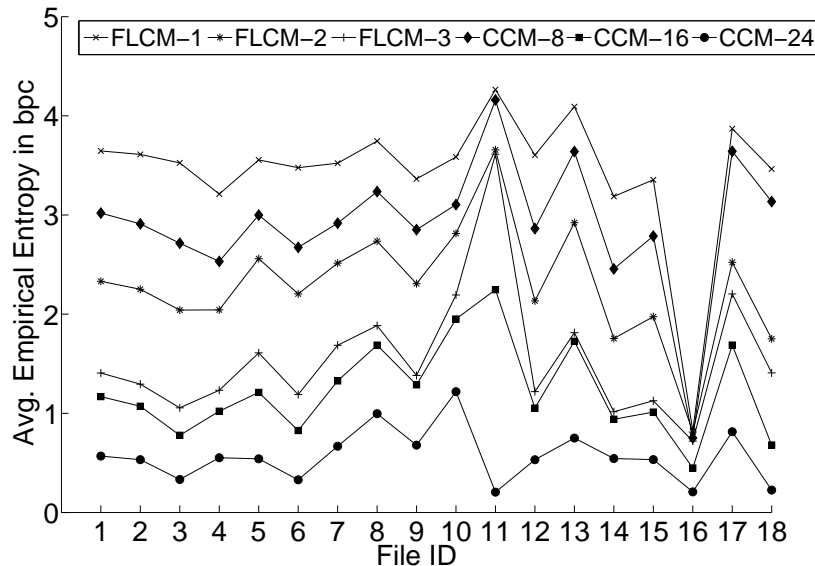


Figure 1. CCM against FLCM according to average empirical entropy.

zeroth order minimum variance static Huffman coding [20]. Thus, the compressed context of  $t_p$  is the bit sequence formed by the corresponding Huffman codes of the preceding characters. The single letter frequencies, which are required to assign static Huffman codes to each character, are counted with a prior pass over the target files.

Each of the 18 files in the corpus are modeled with CCM and FLCM according to different context lengths as 1 to 24 *bits* in CCM, and 1 to 10 *bytes* in FLCM. The average empirical entropies of the files are measured in *bits per character* (bpc) under each model. It is known that the actual compression rates cannot catch these theoretical values. However, *bpc* measurements indicate the descriptive power of the underlying statistical models.

First, the performances of CCM and FLCM are compared on equal context lengths. Since the unit of context is bits in CCM and bytes in FLCM, order- $k$  FLCM is compared with order- $(8 \cdot k)$  CCM. The average empirical entropies measured on all files for context lengths of one-byte (eight-bits), two-bytes (16-bits), and three-bytes (24-bits) are plotted in figure 1, where the length of the context is represented by the value after the dash sign, e.g., 2 in FLCM-2.

The grand average computed over all files in the corpus shows that the average empirical entropy measured under order-8 CCM is 15.38 percent lower than order-1 FLCM. Since lower entropies mean better descriptions of the data, the performances of order-16 and order-24 CCM are 47.44 and 61.67 percent better than corresponding order-2 and order-3 FLCM respectively. This is due to the fact that CCM stores much more information than FLCM in same context length, which is the basic motivation in the CCM proposal.

The amount of memory that a context-based compression scheme requires is mainly determined by the number of observed distinct contexts on the target file. The numbers of distinct contexts observed under CCM and FLCM are investigated in figure 2. It is observed that order-8 CCM generates 2.40 times more distinct contexts than order-1 FLCM. Similarly, order-16 and order-24 compressed context

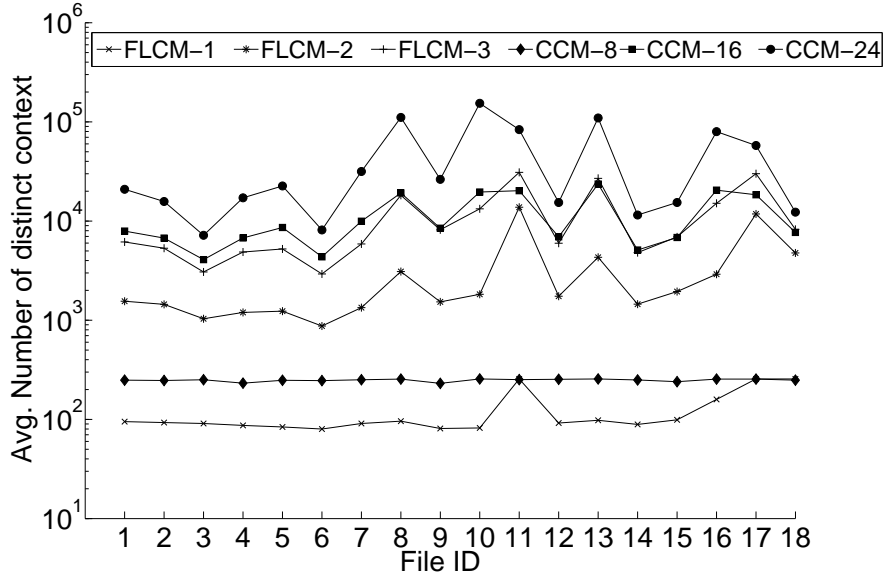


Figure 2. CCM versus FLCM according to number of observed different contexts.

modelings produce 4.96 and 3.79 times more unique contexts than order-2 and order-3 finite-length context modelings respectively.

Hence, a second measurement is performed to see how CCM behaves if we restrict its order to use less number of distinct contexts than the FLCM model. On each file in the corpus, the order- $\ell$  CMM is compared with order- $k$  FLCM such that  $\ell$  is chosen to be the greatest number, where number of distinct contexts in order- $\ell$  CMM is less than order- $k$  FLCM. The results of the comparisons for files *paper1*, *book1*, and *obj1* under this space restriction are depicted in table I. As an example, the selected length of the CCM context corresponding to order-2 FLCM on *paper1* is 11 bits since 11 is the biggest number that the number of contexts under order-11 CCM is less than that of the value under order-2 FLCM. Compressed context modeling claims to have better performance than FLCM in space and/or entropy. Hence, the percentages of gain both in space and entropy are computed on the measurements. For example, the space required by order-11 CCM is 5.08 percent less than the space required by order-2 FLCM (as  $(1556 - 1477)/1556 = 5.08$ ) and the entropy of order-11 CCM is 8.62 percent better than order order-2 FLCM (as  $(2.33 - 2.13)/2.33 = 8.62$ ) on *paper1*.

The percentages of average gains in space and entropy are summarized in table II for every file in the corpus. We observe that there is a loss in entropy on the natural language files if we want to save space with CCM. Note that the gain in space is always superior to loss in entropy in all circumstances, which makes sense in trading space versus entropy with CCM. On semi-structured files such as the geophysical data file *geo* or the *trans* file including the transcript of a terminal session file, CCM brings advantage both in space and entropy. The entropy gain is tremendous on files *pic1* *obj1* and *obj2*. Even when we exclude those files from the statistics, the average values become 14.48 percent for gain in space and 0.002 (negligible) percent for gain in entropy. This shows that space requirements in context-based compressors can be significantly reduced by replacing FLCM styles with the proposed CCM.

Context Length		Number of distinct contexts		Avg. Emp. Entropy in bpc		Percentage of Gain in Space	Percentage of Gain in Entropy
CCM (bits)	FLCM (bytes)	CCM	FLCM	CCM	FLCM		
6	1	64	95	3.61	3.64	32.63	0.9
11	2	1477	1556	2.13	2.33	5.08	8.62
15	3	4769	6155	1.47	1.40	22.52	-4.55
18	4	11324	12841	0.94	0.90	11.81	-4.84
22	5	19412	19841	0.61	0.63	2.16	2.09
24	6	20920	26074	0.56	0.44	19.77	27.74
						Avg.: 15.66	Avg.: -4.25

a) The results of the comparison on file *paper1*.

Context Length in Bits		Number of distinct contexts		Avg. Emp. Entropy in bpc		Percentage of Gain in Space	Percentage of Gain in Entropy
CCM	FLCM	CCM	FLCM	CCM	FLCM		
6	8	64	82	3.49	3.58	21.95	2.46
11	16	1775	1826	2.59	2.81	2.79	7.83
15	24	13187	13296	2.06	2.19	0.82	6.08
18	32	39145	49956	1.74	1.73	21.64	-0.32
22	40	107729	124119	1.38	1.37	13.21	-0.41
24	48	154136	227992	1.21	1.05	32.39	-15.99
						Avg.: 15.47	Avg.: -0.06

b) The results of the comparison on file *book1*.

Context Length in Bits		Number of distinct contexts		Avg. Emp. Entropy in bpc		Percentage of Gain in Space	Percentage of Gain in Entropy
CCM	FLCM	CCM	FLCM	CCM	FLCM		
6	1	249	256	3.13	3.46	2.73	9.49
13	2	3802	4766	1.26	1.75	20.23	27.85
16	3	7686	8246	0.68	2.40	6.79	51.67
18	4	9551	9793	0.46	1.25	2.47	62.74
19	5	10252	10614	0.39	1.24	3.41	68.06
20	6	10832	11147	0.34	1.16	2.83	70.52
						Avg.: 6.41	Avg.: 48.39

c) The results of the comparison on file *obj1*.

Table I  
COMPARISON OF CCM VERSUS FLCM UNDER SPACE RESTRICTION.

## VI. CONCLUSION

Although context based compression algorithms are known to achieve higher compression rates, they are slower than lexicon based approaches and demand for more memory [21], [2], [17]. This disadvantage lacks their usage especially in limited-resource environments such as the mobile devices. The main reason of high resource allocation is the complex data structures required to hold the various different statistics measured in different contexts [17]. Allocation of large memories also brings access problems and latency, which was investigated in section 2.

This study introduced compressed context modeling, which is an attempt of using compression in compression, as an alternative method to fixed-length or unbounded-length context modelings. The length of the context in CCM is not restricted with a number of previous characters, but instead a certain amount of information. This information is again computed by compressing the previous content.

File Name	Percentage of Average Gain in Space	Percentage of Average Gain in Empirical Entropy
paper1	15.66	-4.25
paper6	12.45	-3.30
paper5	11.66	-2.74
progl	18.69	-2.59
paper3	14.39	-2.45
paper4	11.91	-2.06
paper2	18.47	-2.00
book2	24.84	-0.82
bib	15.62	-0.69
book1	15.47	-0.06
geo	3.43	2.44
progc	11.63	2.93
news	15.01	2.96
progp	11.09	4.20
trans	16.89	8.44
pic	17.96	25.77
obj2	6.07	31.58
obj1	6.41	48.39
Average	13.76	5.88

Table II  
COMPARISON OF COMPRESSED CONTEXT MODELING VERSUS FIXED-LENGTH CONTEXT MODELING.

With this property, approximately uniform amount of information is used in predicting the next symbol based on its preceding context, which gives better estimations. Experimentally it is shown that the space requirement, which is directly proportional to the number of distinct contexts in a statistical model, can be lowered with CCM, and it is possible to catch the level of entropy achieved by high-order context lengths in classical methods with shorter context lengths.

The experiments represented in this study compute the *compressed context* of a symbol by compressing its preceding characters with a simple zeroth order Huffman coding. It is believed that using more sophisticated compression schemes such as the higher order Huffman codes, or arithmetic coding will lead to better results than the ones presented in this paper. Future research direction in CCM is to design and implement a new PPM-type compressor based on the compressed-context modeling scheme, and compare the resulting compression ratio with the other alternatives.

#### REFERENCES

- [1] J. G. Cleary and I. H. Witten, "Data compression using adaptive coding and partial string matching," *IEEE Transactions on Communications*, vol. 32, no. 4, pp. 396–402, 1984.
- [2] K. Sayood, *Introduction to Data Compression*, 2nd ed. Morgan Kaufmann Publishers, 2000.
- [3] M. V. Mahoney, "Adaptive weighing of context models for lossless data compression," Florida Tech., Tech. Rep., 2005.
- [4] "Hutter prize," compression competition, <http://prize.hutter1.net/>.
- [5] D. Hirschberg and D. Lelewer, "Context modeling for text compression," in *Image and Text Compression*, 1992, pp. 113–144.
- [6] T. Bell, I. H. Witten, and J. G. Cleary, "Modeling for text compression," *ACM Computing Surveys*, vol. 21, pp. 557–591, December 1989.

- [7] P. G. Howard and J. S. Vitter, "Practical implementations of arithmetic coding," in *Image and Text Compression*, 1992, pp. 85–112.
- [8] A. Moffat, "Implementing the ppm data compression scheme," *IEEE Transactions on Communications*, vol. 38, no. 11, pp. 1917–1921, November 1990.
- [9] I. H. Witten and T. C. Bell, "The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression," *IEEE Transactions on Information Theory*, vol. 37, no. 4, pp. 1085–1094, 1991.
- [10] P. G. Howard and J. S. Vitter, "Design and analysis of fast text compression based on quasi-arithmetic coding," *Journal of Information Processing and Management*, vol. 30, no. 6, pp. 777–790, 1994.
- [11] J. G. Cleary, W. J. Teahan, and I. H. Witten, "Unbounded length contexts for ppm," in *Proceedings of Data Compression Conference*, 1995, pp. 52–61.
- [12] J. G. Cleary and W. J. Teahan, "Unbounded length contexts for ppm." *Computer Journal*, vol. 40, no. 2/3, pp. 67–75, 1997.
- [13] C. Bloom, "Solving the problems of context modeling," 1998, available at <http://www.bloom.com/papers/ppmz.pdf>.
- [14] C. E. Shannon, "A mathematical theory of communication," *Bell Systems Technical Journal*, vol. 27, pp. 379–423, 623–656, 1948.
- [15] J. G. Cleary and W. J. Teahan, "Experiments on the zero frequency problem," in *Proceedings of the Conference on Data Compression*. IEEE Computer Society, 1995, p. 480.
- [16] G. Navarro and M. Raffinot, *Flexible pattern matching in strings: practical on-line search algorithms for texts and biological sequences*. New York, NY, USA: Cambridge University Press, 2002.
- [17] D. Salomon, *Data Compression: The Complete Reference*, 4th ed. Springer, 2007.
- [18] T. Raita and J. Teuhola, "Predictive text compression by hashing," in *Proceedings of 10th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*. ACM, 1987, pp. 223–233.
- [19] M. Farach, "Optimal suffix tree construction with large alphabets," in *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS'97)*. IEEE Computer Society, October 1997, p. 137.
- [20] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the Institute of Radio Engineers*, vol. 40, no. 9, pp. 1098–1101, 1952.
- [21] I. H. Witten, A. Moffat, and T. C. Bell, *Managing Gigabytes : Compressing and Indexing Documents and Images*, 2nd ed. Morgan Kaufmann, 1999.