Eleventh International Workshop on Algebraic and Combinatorial Coding Theory
June 16-22, 2008, Pamporovo, Bulgaria *pp. 229-235*

Recent results in combined coding for word-based PPM

Radu Radescu rradescu@gmail.com
George Liculescu
Polytechnic University of Bucharest, ROMANIA

Abstract. In this paper it is presented the lossless PPM (Prediction by Partial string Matching) algorithm and it is studied the way the extended alphabet can be used for the PPM encoding so it will allow the use of symbols which are not present in the alphabet at the beginning of the encoding phase. The extended alphabet can contain symbols with the size larger than a byte and at the decoding external words absent at decoding are combined with the adaptive-generated words. The arithmetical algorithm is used to encoding of words with the statistic model generated by the PPM algorithm. Some experimental results on various types of files and important interpretations deducted from these results are presented.

1 Introduction

Let us presume that a file contains a string of bytes (characters), which appears many times in the file. PPM must encode independently every byte from the string with a probability (which is preferable to have large value). Every time the character was not found in the past (the string preceding the current context), an escape symbol is send to decrease the level, leading to increment the information from the compressed data stream. The alphabet used by the PPM algorithm has 256 characters (all the characters that can be formed on 8 bits). If the regular alphabet is extended adding a new symbol (the string mentioned above) the algorithm could perform a good compression.

An extended alphabet is an enriched known alphabet with a series of symbols that will not be presented in the alphabet offered to the decoder. The symbols that extend the alphabet need to be obtained in the decoding phase through different methods, so while decoding the alphabet will be enriched with new symbols. In the coding phase, the symbols that will extend the alphabet are known, but at the decoding these will be deduced gradually.

In this paper, we consider that internal words are present at the decoding phase because they are internally generated, and they can be reproduced at the decoding. The external words that could be present at the decoding are inserted externally at both coding and decoding stages. It is considered an optimization of the data tree, so it can be used on the purpose of word-based coding (strings of octets).

In order to minimize the searching time, an optimized algorithm must be used. The red-black tree is used for searching. The red-black tree is a binary tree, which keeps inside of every node an extra-information - the color of the node - that can be red or black. Through the constrain of the way the nodes can be colored with

every line that starts at the base and ends in a leaf, the red-black tree ensures there is no other way which is longer than the other keeping the tree approximately balanced. The procedures that can be performed are that of a classic binary tree. The method used here is dedicated only to the PPM algorithm and it performs the adaptive search of the words.

2 PPM encoding with the extended alphabet

The extended alphabet encoding is similar to the basic alphabet one (made up of every 8 bits symbols). In order to determine which word is next coded, we need to check all the words, which can be made up based on the text bytes, starting from the current position from the considered coding stage. The length of words that can be formed with the text bytes must be smaller or equal to the maximum admitted length and smaller or equal to the maximum length present in the source alphabet. Every formed word is searched in the alphabet and if it is found, a gain is associated to it. In order to compute the gain, we must have information regarding the current state of every word from the alphabet (number of appearances, length, etc.).

The gain can be calculated in many ways, but here it will be calculated as a function of number of appearances and length. Usually, a formula for computing the gain should be used, and this would depend on the context where the word is situated.

In order to know the value of the word from the context point of view, a search in the tree must be performed. This search must be made for every word that has a chosen potential, this being a very big extra task. For this reason, in order to compute the gain, we will use the real number of appearances (imposed on inside basis), in the case of internal words, or the maximum between the real and false number of appearances (imposed on outside basis), for external words. The number obtained from the zero level node gives the real number of appearances. Therefore, every word will have to keep a reference to the correspondent zero-level node to find out the real number of appearances.

The external words, which are present at decoding, and the internal ones are coded using a regular PPM model. In this case, all the tasks that were executed on bytes must be executed on words. Thus in the tree a word and not a byte will be inserted, and the context will be one with words and not with bytes. The saving queue of the 2,048 symbols from the past for the actualization of the tree after the cleaning (if we want to use it) will contain also words and not bytes.

In order to reduce the time of adding and searching within the tree, all the words that are in other structures will be references to words from the used alphabet. In this way, all the comparisons between words could be made based on reference, but a comparison between elements will not be made. The only task that involves comparison between words at byte level is when a word must be searched inside the alphabet.

3 Combining the external words with the adaptive generated words

The encoder and decoder must always keep the same alphabet. When the -1 order is reached, we must encode a word with the probability (the alphabet length)-1. If the length of the alphabet is not identical at encoding and at decoding at that step, then the decoder will not be able to follow the coder's steps and the decoding will fail. In the alphabet, there can be external words present at decoding, external words that are not present at decoding, and internal words (automatically present at decoding). At the encoding step we know all the words of any type but at decoding we will not have at any moment all the words that were considered at encoding external and absent at decoding. For this reason, when we encode a symbol with a probability dependent on the alphabet length, we will consider only the words marked as being present at decoding. This is why it is important that every word which was external and absent at decoding to be marked as being extern and present at decoding only after this word has been encoded character by character and was followed by a special word and a counter type word. The disadvantage of combining external words, which are not present at decoding, with internal words is that the internal ones have priority, replacing the external ones absent at the decoding step. The external words are the result of other algorithms or of user's experience and many times this can be a useful information, which may improve the encoding. At the occurrence of an internal word, which replaces an external one, this useful information is ignored. The problem is that in the case of external words the lifetime is unlimited while the lifetime of internal words is limited if they have not been seen a few times in the past. (the internal words with zero appearance number are erased periodically). The advantage of this combination is that an absent word at decoding can be replaced with an adaptive generated one, which is seen many times until the end of the survival period. Because the adaptive generated word is encoded regularly, and the word absent at decoding is encoded character by character, the result is a gain.

4 Experimental results

The next two tables contain the best results obtained in two different experiments, using both plain and complex test files. The last line from every table represents the number of bits per character obtained on a compression with the standard RAR application.

We can remark that the adaptive mode is efficient when the text contains words that appear repeatedly in the text. For example, it is obvious that **aaa** looks all the same and **limit_comp.xmcd** is an XML format that contains elements of the same type in the tags. If the text has not a predefined structure, then the adaptive mode will generate words that initially can be good but later could be too long or too short. At first, a long word can be generated, but later we can find a piece of text that needs a shorter version of this word (it has a partial match with a text

fragment). This is why it is possible to generate a shorter word if the restrictions are matched. We can first generate a short word and then a longer word, which includes the short one. The words that includes other words are efficient if they are used. Unfortunately, for the files that do not have a well-defined structure it is likely to be generated words that later will not be used as it should. If the word was not seen many times in the past, it will have a small probability.

Parameter / File	aaa	limit_comp.xmcd	concertBach	ByteEditor.exe
Normal	0.01201	5.77123	4.2048	9.32427
Time [sec]	6.783	12.844	10.542	13.846
Adaptive	0.00451	5.57401	4.19925	9.17483
Time [sec]	0.078	17.260	10.330	13.867
gainTypeadaptive	Equal	Length	Equal	Length
clearPeriod [octets]	1500	1500	1500	1500
Adaptive / max.word	0.00471	5.51759	4.44394	9.17169
Time [sec]	0.060	18.622	10.392	14.049
gainTypeadaptive	Equal	Length	Equal	Appearances
clearPeriod [octets]	1500	1500	1500	1500
Search	0.00431	5.43396	-	9.20337
Time [sec]	0.073	19.289	-	13.478
gainTypesearch	Equal	Length	-	Appearances
gainTypecoding	Length	Length	-	Length
Search / max.word	0.00431	5.4443	-	9.20337
Time [sec]	0.070	16.257	-	14.092
gainTypesearch	Equal	Equal	-	Appearances
Adaptive / Search	0.00411	5.42187	-	9.18202
Time [sec]	0.401	20.542	-	13.809
gainTypeadaptive	Length	Appearances	-	Length
gainTypesearch	Appearances	Length	-	Appearances
clearPeriod [octets]	1500	1500	-	1500
Adaptive / Search / max.word	0.00521	5.4284	-	9.17326
Time [sec]	0.065	20.566	-	14.050
gainTypeadaptive	Appearances	Length	-	Appearances
gainTypesearch	Appearances	Equal	-	Appearances
clearPeriod [octets]	1500	1500	-	1500
RAR (”best” mode)	0.01141	3.7816	2.53007	7.30202

Table 1. Comparison of the best results (first experiment)

From the performed experiments this negative effect was not noticed. For every file, the adaptive search produced better results. The adaptive search of words cannot see in the future and cannot view which is the best word to choose. This is why the adaptive search is recommended only for files with a specific structure. We can remark that it is best to use the word with the biggest length when the text has a defined structure, because it is very likely that this will show in the future.

The most efficient from the compression point of view is the search of words that appear repeatedly, before the encoding. For the files that contain redundant words which can be seen in a period of existence of a word (*clearPeriod*) so they can be added in the alphabet, the adaptive searched is combined with that performed in a separate stage from the encoding.

The **aaa** file is compressed the best by using the adaptive search together with that performed before the encoding, because the search of words in a separate stage is limited to 255 (bytes), while the adaptive search is unlimited. The encoder uses words found by search, in a separate phase, and it adaptively extends them based on the data tree. For the experiments, the adaptive search was limited to the length of 1,000.

Parameter / File	paper1	progc	obj1	trans
Normal	3.78187	3.84196	5.38653	2.52419
Time [sec]	7.059	5.468	4.801	10.452
Adaptive	3.78157	3.82035	5.11496	2.51019
Time [sec]	6.880	5.359	4.636	11.340
gainTypeadaptive	Appearances	Appearances	Length	Appearances
clearPeriod [octets]	1500	1500	1500	1500
Adaptive / max.word	3.89955	3.8975	5.07924	2.60624
Time [sec]	7.047	5.411	4.704	12.266
gainTypeadaptive	Equal	Equal	Length	Equal
clearPeriod [octets]	1500	1500	1500	1500
Search	3.77736	3.80298	5.18936	2.45068
Time [sec]	9.549	4.952	4.056	21.223
gainTypesearch	Length	Appearances	Appearances	Equal
gainTypecoding	Equal	Length	Length	Length
Search / max.word	3.78006	3.80298	5.20945	2.45068
Time [sec]	9.560	4.940	4.115	21.196
gainTypesearch	Length	Appearances	Appearances	Equal
Adaptive / Search	3.7915	3.82076	5.12128	2.46476
Time [sec]	9.874	5.977	4.656	22.842
gainTypeadaptive	Appearances	Appearances	Equal	Appearances
gainTypesearch	Length	Equal	Appearances	Equal
clearPeriod [octets]	1500	1500	1500	1500
Adaptive / Search /max.word	3.8699	3.89488	5.07961	2.57713
Time [sec]	9.399	7.315	5.402	20.992
gainTypeadaptive	Appearances	Equal	Length	Appearances
gainTypesearch	Length	Length	Appearances	Equal
clearPeriod [octets]	1500	1500	1500	1500
RAR ("best" mode)	2.20748	2.23312	3.65365	1.26385

Table 2. Comparison of the best results (second experiment)

The **ByteEditor.exe** file is an executable file, which is extended and not compressed. The problem comes from the PPM classic encoder and not from the extended alphabet. A smaller size is obtained with the help of adaptive search.

The **concertBach** file did not contain words that will match the rules imposed by the search in other stage than that of coding, so for this reason any experiment that had this type of search was not performed.

We can observe that the best performances of the **aaa, limit_comp.xmcd** and **progc.cs** compression is obtained by combining the adaptive search with that performed in a separate stage. This phenomenon is present because the coding of a missing word at the encoding means the coding of every of its character, while the coding of a present word does not have these disadvantages. If it is found a word with the adaptive search and this is present in the alphabet and is

absent at decoding, then it is replaced with the one adaptively generated. After the insertion of the word in the text, we find (until the end of the word's existing period) a match with this word and therefore we have a gain because the word has not been coded character by character. In the case when the external word (absent at decoding) was replaced by an adaptive generated word that will not be found in the text, then we have a loss.

The encoding time usually increases compared to the PPM with regular alphabet because the words represented by strings of bytes, not only by bytes, must be checked. The file **aaa** is a special case where the coding time drops because there are few words, of very big length, which are coded.

The **concertBach** and ByteEditor.exe files are better coded by using the adaptive search because the restrictions imposed to the search performed in a separate stage of that of coding are too strong. These texts contain short words that appear repeatedly, and the adaptive search manages to find some of them because its minimum length is 5, while for the search before coding the minimum length is 20. For all the test files, the same encoding parameters were used. For this reason, we cannot say these are the best results that can be obtained. Still, an improvement is obtained.

Table 3 presents the experimental results obtained by using the adaptive search encoding for a larger set of original file types. One can analyze the compression efficiency obtained when all the words are accepted in the alphabet and the value of the existing period of a word (*clearPeriod*) is big. (e.g., 20,000 bytes).

File	Bits/character	Time [sec]
aaa	0.00510	0.030
limit_comp.xmcd	5.08220	78.550
concertBach	3.80084	34.441
ByteEditor.exe	9.18898	18.288
paper1	3.57781	23.326
progc	3.58870	16.224
obj1	4.92113	9.789
trans	2.40917	40.403

Table 3. Parameters for adaptive search without restrictions

One can see that for the proposed files the use of the adaptive search combined with the adding of words with no restrictions has better compression results but less quality time results from the gain and minimum length point of view. The time increases because there are many words in the alphabet and their search lasts a long time. The compression ratio is better because the words are early discovered and used. Although the alphabet has many words and the probability of a word at the -1 prediction level (the reverse value of the alphabet length) is small, the encoding is not strongly influenced by this because the -1 level situations are rare.

5 Conclusions

The most efficient is the search before the encoding together with the use of maximum length word at the encoding. The adaptive search can be performed

in the case of files with many repeating words and has the advantage that it is performed at the coding phase. The combining of the two procedures of search can be used only for a certain types of files that contain words that get repeated nearby (so that the adaptive search can find them) and words situated much apart in the text so they won't be included in the alphabet by the adaptive search. From the tests, it results that the gain function depends on the type of used files for most of the files. The difference between the extended alphabet PPM encoding and the WinRar compression is about 1.5 bits/character. The file **aaa** (plain text) is compressed better with the extended alphabet PPM. From the previous results, one can observe that the encoding with adaptive search without restrictions is the most efficient and most files are compressed better with extended alphabet PPM.

References

[1] A. Moffat, Implementing the PPM Data Compression Scheme, *IEEE Trans. Commun.* 38, 1990.

[2] Th. H. Cormen, *Introduction to Algorithms*, Ch. E. Leiserson , R. L. Rivest eds., The MIT Press. 1999.

[3] N. Abramson, *Information Theory and Coding*, McGraw-Hill, NY, 1963.

[4] T. Bell , I. H. Witten, J. G. Cleary, *Modeling for Text Compression*, ACM Computing Surveys 21, 1989.

[5] Pr. Skibinski, PPM with the Extended Alphabet, *Inform. Sci.* 176, 2006.

[6] J. Bentley, D. McIlroy, Data compression using long common strings, *Inform. Sci.* 135, Part 1-2, June 2001.

[7] A. T. Murgan, *The Principles of Information Theory in Information and Communication Engineering*, Romanian Academy Press, Bucharest, 1998.

[8] R. Radescu, *Lossless Compression - Methods and Applications*, Matrix Rom Press, Bucharest, 2003.

[9] M. Nelson, *The Data Compression Book*, 2nd Ed., J.-L. Gailly ed., M&T Books, 1995.

[10] *** *Data Compression - The Complete Reference*, 3rd Ed., D. Salomon ed., Springer-Verlag, 2004.

[11] *** *Lossless Compression Handbook*, 1st Ed., Kh. Sayood ed., Acad. Press, 2002.

[12] R. Radescu, C. Harbatovschi, Compression methods using prediction by partial matching, *Proc. 6th Intern. Conf. Commun.*, Bucharest, 2006, 65-68.

[13] R. Radescu, R. Popa, On the performances of symbol ranking text compression method, *Sci. Bull. "Politehnica" Univ. Timisoara, Romania, Trans. Electr. Commun., Electr. Telecomm. Symp.* 49, ETC 2004, 25-27.

[14] The Calgary Corpus:
ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus

[15] www.winrar.com