5 S − 1

BSTW Data Compression with PPM

Damras WONGSAWANG *and* Masayuki OKAMOTO
Shinshu University

1 Introduction

Algorithm BSTW developed by Bentley, Sleator, Tarjan and Wei [1] is one of the most efficient data compression algorithms. This algorithm takes an advantage of locality of reference, the tendency of words to occur frequently for a short period of time and then fall into long period of disuse. The scheme is based on a self-organizing list, for maintaining code table. PPM (Prediction by Partial Matching) in conjunction with Arithmetic coding, first investigated by Cleary and Witten [2] and later modified and implemented by Moffat [4] had been also reported of high performance data compression. The algorithm bias probabilities assigned to symbols according to the immediately preceeding text. In this paper we propose the using of BSTW's idea and employ PPM to fomulate adaptive data compression model.

2 BSTW With PPM Data Compression Model

BSTW method is a defined-word scheme that attemps to exploit locality of reference. The algorithm uses a self-organizing list as an auxiliary data structure and shorter encoding for words near the beginning of the list. BSTW employs "move-to-front" heuristic for maintaining the code table. During the encoding process, code table will be created and reorganized by keeping current symbol at the first position. Encoding integers will be the numbers representing their positions in current code table. Our data compression model is similar to BSTW but different in the method of maintaining code table. We try to exploit the interdependency of occurrence of source messages for maintaining code table. The reorganization of code table will be based on the prediction from most recent symbols seen, ie. PPM, as the following condition.

If $p(s_i|c_k) > p(s_j|c_k)$ then $d(s_i) < d(s_j)$

where $p(s_i|c_k)$ is a probability of next occurrence of symbol s_i given a preceeding context c_k,
and $d(s_i)$ is a distance of symbol s_i from the beginning of code table to its position.

The preceeding contexts c_k may be fixed or variable length. For fixed length context model, they will be referred to as $1st, 2nd, 3rd, ...$ order model according to the length of contexts. The encoding processes are performed in the same way as BSTW that each symbol will be converted into an integer representing its position in current code table. For integer encoding we used optimal universal Elias prefix codes [3]. This code maps an integer x onto binary value of x prefixed by $\lfloor \log x \rfloor$ zeros. For example, 1 will be maped onto binary digits as "1", 2 as "010", 3 as "011", 4 as "00100" and etc.

3 Algorithm and Implementation

The algorithm of our model can be explained as follow.

1. Set default previous context.
2. Input a data symbol to be encoded.
3. Look for this symbol in code table (initialize with null). If exist, encode by sending its current position, else encode by sending $n + 1$ (n is no. of symbols in code table) follow by new symbol and add new symbol to code table.
4. Accumulate frequency count table depending on previous context and current symbol.
5. Set new previous context.
6. Reorganize code table based on previous context and corresponding symbol frequency counts.
7. Repeat from step 2 to step 6 until end of file.

The computer programs have been developed in C language with no purpose of optimal time processing. The various types of text files have been tested with various order context models.

4 Experimental Results

A simple model using 1st order context has been tested with text files, eg. C source programs (files 1-3), UNIX manual pages (files 4-9), UNIX OS and Utilities document (file 10). The results are shown in Table 1, comparing among the other methods, BSTW move-to-front (byte level), "compact", arithmetic coding [5] and our algorithm. The numbers show the percentage of reduction in size after compression, ie.

*((original-compressed)/original*100).*

No	Size KB	BSTW MTF	BSTW PPM (1st)	compact	Arith
1	23	30.5	62.4	42.5	33.2
2	34	41.7	68.3	50.7	39.4
3	36	12.1	47.8	36.2	28.9
4	35	12.2	43.5	36.5	28.8
5	41	10.0	46.7	38.3	30.1
6	57	11.7	46.9	37.5	30.2
7	58	12.4	50.8	36.9	30.0
8	68	14.7	49.4	39.9	33.3
9	72	13.4	49.7	39.2	32.6
10	199	27.7	52.1	42.5	35.1

Table 1 : Experimental results comparing among various methods with our 1st order model.

The higher order model, upto 5th, have been also tested with the same set of text files. The results, comparing with UNIX utility "compress", are shown in Table 2.

No	compress	BSTW with PPM			
		2nd	3rd	4th	5th
1	67.4	71.3	72.9	72.9	72.7
2	73.9	74.0	75.6	75.6	75.6
3	55.0	59.4	63.4	63.7	63.8
4	48.7	51.5	55.6	54.3	53.6
5	54.6	56.8	59.8	58.9	58.2
6	55.0	57.3	60.8	59.7	58.9
7	61.4	64.4	68.4	68.8	68.5
8	57.4	58.8	63.7	63.0	61.9
9	60.9	62.8	67.6	67.9	67.6
10	62.3	62.5	67.2	66.9	66.2

Table 2: Experimental results of higher order models.

5 Conclusion

The theoretical behind our compression model is that if successive input data are dependent the entropy becomes less per unit than an individual input. The first-order entropy is defined as :

$H_1 = -\sum_i p(s_i)\log p(s_i)$

where $p(s_i)$ is the probability of occurrence of symbol s_i.

The second-order entropy is also defined as :

$H_2 = -\sum_j \sum_i p(s_i, s_j)\log p(s_i, s_j)$

where $p(s_i, s_j)$ is the join probability density function. The higher order of entropy can be defined in the same way as second-order. They can be shown that

$H_1 \geq H_2 \geq H_3 \geq \dots$

This means that we can reduce the entropy or increase the redundancy of data that give more possibly to compress.

We have described a simple data compression model but capabality of high compression rate. From experimentals, they suggested that the method may be useful in practice, especially for data having high interdependency, eg., text files. However, this method can not compress well enough when apply to non-text files. Another problems still unresolved are processing time and memory space. These problems require further study and development.

References

[1] Bentley, J. L., Sleator, D. D., Tarjan, R. E. and Wei, V. K.; *A locally Adaptive Data Compression Scheme*, Communications of the ACM, 29, 4 (Apr 1986), 320-330.

[2] Cleary, J. G. and Witten I. H.; *Data Compression Using Adaptive Coding and Partial String Matching*, IEEE Trans Comm., COM-32, 4 (Apr 1984), 396-402.

[3] Elias P.; *Universal Codeword Sets and Representation of Integers*, IEEE Trans Inf. Theory, 21, 2 (Mar 1975), 194-203.

[4] Moffat A.; *Implementing of PPM Data Compression Scheme*, IEEE Trans Comm, 38, 11 (Nov 1990), 1917-1921.

[5] Witten I., Neal R. and Cleary J.; *Arithmetic Coding for Data Compression*, Communications of the ACM, 30, 6 (Jun 1987), 520-541.