Enhancement Of Lempel-Ziv Coding Using A Predictive Pre-Processor Scheme for Data Compression

RAJASVARAN LOGESWARAN
Faculty of Engineering
Multimedia University
63100 Cyberjaya,
MALAYSIA

Abstract: - In the implementation of lossless compression, the traditional Lempel-Ziv dictionary algorithm produces good compression performance, but at a relatively slow processing speed for use in real-time applications. This paper highlights a technique to speed up the overall compression process through introduction of a prediction pre-processor stage. A variety of predictors are applied to two variations of the Lempel-Ziv - LZSS and LZARI. It is shown that the proposed two-stage scheme significantly reduces the overall processing time taken for to encode raw telemetry data, as well as improves the overall compression performance achieved.

Key-Words: - Neural Networks, Lempel-Ziv, Prediction, Data Compression, Optimisation, Two-stage scheme

1 Introduction

The use of data compression is very significant in most fields that utilise computers and related machinery in one way or the other. The reduction of data size though the removal of redundancies in the data, possesses many benefits. Among these are : the ability for increased data storage in fixed capacity devices, quick manipulation of large data files, faster data transfer over fixed bandwidth and reduced total transmission error (assuming constant probability of transmission error along a medium, reduced data transmitted implies reduced total error incurred by that data during transmission).

In terms of accuracy, a number of applications require lossless data compression, as opposed to the higher compression capabilities of lossy compression. Applications such as binary files, satellite telemetry data and medical images have to be lossless as lossy effects could render the files inaccessible, may pose difficulties in diagnosis / analysis, or in extreme cases, may even give rise to "false" data due to inaccuracies and missing parts of the data.

There exists many algorithms for lossless data compression, ranging from primitive strategies of null suppression, to well known statistical techniques such as Huffman and arithmetic coding, as well as combination techniques which integrate a number of strategies within the algorithm, e.g. Sixpack [1]. One technique that produces good compression performance, but at the expense of relatively slow processing is the dictionary method, such as that employed by Lempel-Ziv (LZ) coding schemes [2].

This paper seeks to improve the performance of two LZ variations, especially in terms of processing time. This is done through the application of a two-stage scheme, which utilises classical predictors as well as artificial neural networks as a prediction-based pre-processor to the LZ encoder.

2 The Lempel-Ziv Algorithms

The basic concept of a dictionary-type encoder is that the encoder builds a table of characters / words / phrases that have been encountered in the input stream, assigning a corresponding codeword to each entry in the table. When an instance in the input matches an entry in the table (dictionary), it is replaced with the corresponding codeword. Dictionaries may be static (pre-defined) or dynamic (built and updated at run-time, with the possibility of initialisation with a pre-defined mini-dictionary of common matches at start-up). A description of the LZ algorithm as well as the two variations used in this paper is given below.

2.1 Lempel-Ziv

The LZ coding (also known as Ziv-Lempel and LZ77) [2] has a text window divided into two parts. The first part is the body of the text window (containing recently encoded text), and the second is a look-ahead buffer that contains the new input to be compressed. The algorithm tries to match the contents of the look-ahead buffer to a string in the dictionary (body of text window). Compression is achieved by replacing variable-length text with fixed size tokens. Each token has three parts : a pointer into the dictionary, the length of the phrase and the first symbol in the look-ahead buffer that follows the phrase, as given in Fig. 1.

Position	Length	Next character

Fig. 1 : Format of LZ77 encoded token

An example of the algorithm is illustrated in Fig. 2. During compression, the very first character (say *a*) is represented as 0, 0, '*a*'. During decompression, a similar text window will be created. The token is read, the indicated phrase is output and the remaining character (i.e. *a*) is appended. After this, the dictionary and look-ahead buffer is used, and the process is repeated until the end of the input. The implementation of LZ77 can cause a bottleneck, as string comparison has to be done at every position in the text window. When matching strings are not found, the cost of using this algorithm is very high as the 24-bit token is used to encode each unmatched 8-bit character, thus resulting in possible data expansion.

Contents of the Text Window (processed input in dictionary)	**Contents of the Look-Ahead Buffer**
d; for (i=0; i<MAX-1;i++)\r for(j=i+1;j	<MAX ; j++)\r a[i

(a) LZ77 Text window and look-ahead buffer at a particular instance

(i=0; i<MAX-1;i++)\r for(j=i+1;j<MAX	;j++)\r a[i]=100

(b) Text window and look-ahead buffer shifted left by 4 places after encoding the string '<MAX' into the LZ77 encoded symbol *13,4,';'* using the format given in Fig. 1

Fig. 2 : Example of Lempel-Ziv Coding

The LZ class of algorithms has many variations which implement various adaptations and improvements. Among the popular variations are : LZSS, LZ78, Lempel-Ziv-Welch (LZW), LZJ, LZT, LZC, Lempe-Ziv with Huffman coding (LZH) and Lempel-Ziv with arithmetic coding (LZARI) [1][3][4]. Two are evaluated in this paper : LZSS and LZARI.

2.2 LZSS

LZSS by Storer & Szymanski in 1982 [3] is an improvement of the LZ77, which updates a binary search tree with the phrases that are moved out of the look-ahead buffer into the dictionary. This allows string comparison to be done quickly, thus reducing the performance bottleneck. LZSS also reduces wastage by modifying the token. It uses a 1-bit prefix to indicate whether an offset (length pair) or a single symbol for out put is being sent. When a match is not found, the dummy position 0 and length 0 will not be sent.

At the start of compression, performance is poor as there is no data in the text window for comparison. A further improvement to the implementation would be to find probable data beforehand to be pre-loaded into the body of the text window. Therefore, meaningful tokens can be used right from the start, thus improving compression

2.3 LZARI

LZARI is a multilevel coding technique that uses a two-pass operation [1]. The first pass uses one of the better LZ algorithms, followed by the second pass in which the tokens (code pointers) are encoded using arithmetic coding. The arithmetic coding [5] encodes the codewords into a floating point number f_i (such that $\forall i: 0 \le f_i < 1.0$). Each codeword occupies a different non-overlapping range between 0 and 1.0, such that $\forall i \forall j: i=j$ $\text{low}_i \le f_i < \text{high}_i$, where low_i and high_i are the lower and upper bound of the range occupied by f_i.

The dictionary (text window) building and look-up phases cause the LZ algorithms to be relatively slow during implementations, as opposed to other popular schemes such as the statistical Huffman and arithmetic coding. However, the compression performance, in terms of compression ratio achieved by the LZ tends to be better than the simpler schemes, even when applied for compression of data which handicaps it (such as numeric data with very few repeated values, as used to test the schemes reported in this paper - see Section 4) [6].

3 Two-Stage Scheme

It is proposed in this paper that a pre-processing stage be integrated with the encoders to improve performance in terms of processing time of the LZ implementation. Generally, it is desired that the number of stages in any process for data manipulation be minimised, in order to minimise total processing time. However, it will be shown in Section 4 that the introduction of an additional pre-processing stage can in fact bring about significant benefits in terms of processing time of the LZ, especially when dealing with numeric data.

The two-stage scheme basically integrates a predictor (coupled with a residue generator) as a pre-processor to the encoder, as shown in Fig. 3. The first stage reduces the dynamic range of the input by predicting the current value (X_n) at each iteration. Prediction is achieved using the *p* past input values. At each iteration, the output generated is the corresponding residue (error between the actual and predicted values). By transmitting the residue, the receiver then utilises the same predictor and adds the residue to the predicted value to restore the original input.

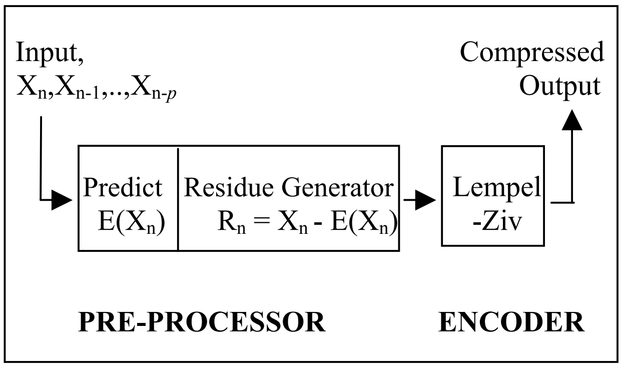
Input, $X_n, X_{n-1}, .., X_{n-p}$
Compressed Output
Predict $E(X_n)$
Residue Generator $R_n = X_n - E(X_n)$
Lempel -Ziv
PRE-PROCESSOR
ENCODER

Fig. 3 : Two-stage Scheme

A good predictor would remove the redundancy in the input thus minimising the number of unique patterns passed on to the encoder. As such, a dictionary type encoder benefits from the reduced size of the dictionary / lookup table it needs to build and use. In the case of numeric data, the mean and mode value of the residues of a good predictor would be 0, thus most of the input to the encoder would be encoded with the same codeword. This in turn allows the size of the codeword to be minimal, thus achieving higher compression.

4 Performance Evaluation

Evaluation of the implementation is carried out with a data set of 6 satellite launch vehicle telemetry data files of varying magnitude and distribution patterns, each containing a different type of telemetry measurement (e.g. pressure, temperature etc.).

4.1 Characteristics of the Test Data

The data files generally consists of pairs of values, (each converted to a 16-bit integer value). The first parameter is a reference value with increasing linear distribution, while the second the actual measurement reading which varies in pattern and distribution. Table 1 gives the characteristics of the test data [7]. The total number of samples and sampling rate is equal for both parameters. There are no repeated values in the first parameter, so the maximum frequency of a symbol is 1 and the number of distinct levels is equal to the number of samples. This produces high entropy, and in turn, high source information rate. Entropy, calculated using Eq. (1), gives the minimum number of bits required to encode the data, assuming that each distinct symbol is represented by a different bit pattern.

Table 1 : Characteristics of the test data files

				Parameter 1				**Parameter 2**			
Test File	**File Size (bytes)**	**Total number of symbols**	**Sampling rate (symbols per sec.)**	**Number of distinct symbols**	**Max. freq. of a symbol**	**Max. value of a symbol**	**Source Entropy (bits per symbol)**	**Number of distinct symbols**	**Max. freq. of a symbol**	**Max. value of a symbol**	**Source Entropy (bits per symbol)**
Tdata1	252305	28324	520	28324	1	159.9	14.790	157	7131	66.135	3.128
Tdata2	139571	11631	65	11631	1	368.0	13.506	12	7484	1070.249	1.017
Tdata3	55365	6778	65	6778	1	119.9	12.727	43	1438	76.105	4.644
Tdata4	131841	16052	130	16052	1	139.9	13.970	191	3985	50.894	5.387
Tdata5	184774	17232	65	17232	1	349.9	14.073	240	349	4960.000	7.614
Tdata6	74915	8662	65	8662	1	149.9	13.080	6	2840	124.250	2.121

$$H = -\sum_{i-1}^{n} P_i \log_2(P_i) \qquad (1)$$

The value obtained by Eq. (1) using probability of occurrence (P_i) of symbol i, would be representative of a fixed model. Adaptive models are capable of greater compression (i.e. total compressed size less than the product of the entropy and total number of symbols).

For the second parameter, it is observed that the least number of distinct levels is 6 (for 'Tdata6') whilst the highest is 240 (for 'Tdata5'). The file 'Tdata5' has equi-probable symbols, and as such, also has the highest source entropy of 7.614.

4.2 Predictors

A variety of predictors may be used in the first (pre-processor) stage of the two-stage implementation. A sample of 3 classical predictors as well as 2 artificial neural network models are reported here :

- a 5th-order fixed finite impulse response (FIR) filter given by Eq. (2) [8],

$$E(X_n) = 4X_{n-1} - 7X_{n-2} + 7X_{n-3} - 4X_{n-4} + X_{n-5} \qquad (2)$$

- an adaptive FIR with normalised least mean squared (NLMS) algorithm,
- a 2nd-order adaptive recursive least squares lattice filter with a-priori estimation errors and error feedback (RLSL) [9].
- the 4th-order single layer perceptron (SLP) neural network, and
- the 2nd-order multilayer perceptron (MLP) [10] neural network.

The neural network models were trained adaptively using block of training data, such that 20% of the block was used for training the remainder values were predicted. More details may be found in [11].

4.3 Processing Time

The processing time taken by the encoders in the single- and two-stage schemes is summarised in Table 2. The tabulations are simulation results on the Sun Ultra-10 platform, with the results for the neural predictors estimated based on internal operations and I/O performance on a parallel version of the same platform. An appropriate neural chip, ASIC or FPGA may be used.

From the table, it is shown that the processing time of the LZSS and LZARI are significantly shortened when the pre-processing stage is introduced. Amongst the predictors evaluated, the neural network SLP aids both encoders the most, with respect to reduction in total processing time.

It would be noticed from the results that the surprising performance in processing time brought about by the additional stage. Instead of slowing the implementation, as is usually the case for additional phases, the pre-processor has contributed to significantly reduce the processing time from approximately 8.5 seconds for the LZ algorithms by themselves, to less than 1 second for all the two-stage schemes. The processing time for the SLP-LZARI of 0.48 seconds (approximately 6% of the original LZARI implementation) brings about an increase of efficiency of more than 15 times.

The reason for this significant difference is due to the fact that the predictors generally performed well on the test data. As such, resulting in small residue values with a small dynamic range, i.e. the frequency of each value was high. As the LZ algorithms only had to encode the residue stream, the size of the dictionary was kept minimal, thus the gains in speed.

Table 2 : Overall Processing Time for Encoder-only and Two-stage schemes

Test File	**LZSS**						**LZARI**					
	LZSS only	**FIR – LZSS**	**NLMS -LZSS**	**RLSL –LZSS**	**SLP – LZSS**	**MLP – LZSS**	**LZARI only**	**FIR - LZARI**	**NLMS-LZARI**	**RLSL-LZARI**	**SLP-LZARI**	**MLP-LZARI**
Tdata1	7.18	1.52	0.98	0.84	0.56	0.58	7.25	1.53	0.98	0.88	0.56	0.58
Tdata2	12.59	0.65	0.44	0.62	0.50	0.52	12.70	0.67	0.43	0.61	0.49	0.51
Tdata3	4.09	0.46	0.31	0.28	0.38	0.40	3.98	0.44	0.33	0.25	0.38	0.40
Tdata4	7.45	0.92	0.78	0.60	0.44	0.46	7.56	0.90	0.78	0.55	0.43	0.45
Tdata5	8.71	1.57	1.56	0.86	0.67	0.69	8.81	1.44	1.93	0.78	0.66	0.68
Tdata6	10.95	0.61	0.51	0.41	0.38	0.40	10.78	0.58	0.48	0.38	0.38	0.40
average	**8.50**	**0.96**	**0.76**	**0.60**	**0.49**	**0.51**	**8.51**	**0.93**	**0.82**	**0.58**	**0.48**	**0.50**

Table 3 : Compression Ratio achieved by Encoder-only and Two-stage schemes

	LZSS Encoder						LZARI Encoder					
Test File	**LZSS only**	**FIR – LZSS**	**NLMS -LZSS**	**RLSL –LZSS**	**SLP – LZSS**	**MLP – LZSS**	**LZARI only**	**FIR - LZARI**	**NLMS- LZARI**	**RLSL- LZARI**	**SLP- LZARI**	**MLP- LZARI**
Tdata1	7.83	5.22	21.09	32.32	21.99	23.15	21.31	5.22	21.09	32.32	24.63	25.38
Tdata2	6.49	5.10	22.03	42.88	18.42	28.70	16.45	5.10	22.03	42.88	25.34	34.99
Tdata3	5.54	2.82	11.75	38.31	38.93	27.77	11.57	2.82	11.75	38.31	49.57	45.61
Tdata4	6.53	4.47	10.94	39.78	36.65	22.24	16.02	4.47	10.94	39.78	40.32	27.80
Tdata5	5.54	2.75	3.09	16.36	5.45	5.25	9.58	2.75	3.09	16.36	7.40	6.34
Tdata6	5.79	3.58	44.46	72.17	55.08	70.15	13.10	3.58	44.46	72.17	102.06	132.36
average	**6.29**	**3.99**	**18.89**	**40.30**	**29.42**	**29.54**	**14.67**	**3.99**	**18.89**	**40.30**	**41.55**	**45.41**

4.4 Compression Performance

As desirable as speed is, when dealing with compression, it is important to ensure good compression performance. The results, in terms of compression ratio achieved for each test file, is presented in Table 3.

In general, it is observed that there is performance gain in terms of compression. This is again due to the fact that the size of the codewords for the LZ were reduced as the dynamic range of the residues is lesser than the original input stream, in additional to the fact that the magnitude of residue values are also much reduced as compared to the original input. This is not always the case though.

From table 3, it is observed that the compression performance actually deteriorates with the introduction of the FIR first stage. The fixed predictor was not an efficient predictor as it was unable to adapt to the changing data, and produced large residues. This can cause data expansion (resulting compression ratios smaller than those achieved by the single stage LZSS and LZARI implementations), instead of compression. Sequences of similar large residues would still reduce the dictionary building and lookup time, but coding the residues into the dictionary would result in poor compression. This is observed when comparing the performance of the FIR in Tables 2 and 3.

5 Conclusion

In this paper, a two-stage scheme was proposed to improve the performance of the Lempel-Ziv (LZ) dictionary-type lossless compression algorithm. A pre-processor stage utilising a predictor and residue generator was integrated with the LZ encoder. The scheme was tested using two variations of the LZ algorithm, namely LZSS and LZARI, combined with a variety of classical and artificial neural network predictors.

As a general rule, an increased number of stages or processes in data manipulation leads to increased processing time. This paper, however, shows that this is not necessarily the case. In effect, the total processing time may be significantly reduced with the introduction of a suitable pre-processing stage, albeit the likelihood of additional resource requirements (processing power, storage buffer etc.) for implementing the additional stage. Through empirical or analytical means, the number and type of pre- / post-processing stages used should be determined by taking into account the overall performance gains vs. the resources requirements for the integration. Simulation results provided for some small known predictors show that performance gains of up to 17 times may be achieved by the two-stage scheme as opposed to single-stage encoder-only implementations.

In addition to processing speed gains, the results provided also shows that the introduction of a pre-processor may also enhance compression performance significantly, even up to more than 10 times that of the single-stage encoder implementation. This certainly merits the use of the two-stage scheme for lossless data compression. A similar idea may also be ported for lossy compression.

It must be noted that the test data sets used in this paper were numeric telemetry data, which is not best suited for LZ compression. The intentional use of such data was to portray a non-optimum scenario for better clarity in the empirical results. The predictors used were of low complexity for practical implementation of a small setup with limited resources.

The result presented in this paper recommend that with a suitable pre-processor stage, performance of the encoder can be improved significantly, when dealing with different data. This finding may also hold true for other types of encoders. Other tests with Huffman and arithmetic coding did not render better speed as these algorithms are essentially fast, but the respective two-stage schemes did exhibit significant improvement in compression performance [6] at the cost of a slight delay in processing efficiency. Performance (compression as well as speed) relies on the combination of both stages, so the suitability of the predictor in generating an appropriate residue stream for the particular encoder must be emphasised [6].

References:

[1] M. Nelson and J.L. Gailly, *The data compression book*, New York : M&T Books, 1996.

[2] J. Ziv and A. Lempel, “A universal algorithm for sequential data compression”, *IEEE Transactions of IT*, Vol. IT-23, No.3, 1977, pp. 337-343.

[3] J.A. Storer and T.G. Szymanski, “Data compression via textual substitution”, *Journal of the ACM*, Vol.29, No.4, 1982, pp. 928-951.

[4] J. Ziv and A. Lempel, “Compression of individual sequences via variable-rate coding”, *IEEE Transactions of IT*, Vol. IT-24, No.5, 1978, pp. 530-536.

[5] G.G. Langdon, “An introduction to arithmetic coding”, *Journal of IBM R&D*, March 1984.

[6] R. Logeswaran and C. Eswaran, “Effect of encoders on the performance of lossless two-stage data compression schemes”, *IEE Electronics Letters*, Vol.35, No.18, 1999, pp. 1515-1516.

[7] F. Rahaman, Adaptive entropy coding schemes for telemetry data compression, *Project Report No. EE95735*, Madras, India : IIT, 1997.

[8] J.W. McCoy, N. Magotra and S. Stearns, “Lossless predictive coding”, *IEEE Midwest Symposium on Circuits and Systems*, 1994, pp. 927-930.

[9] S. Haykin, *Adaptive Filter Theory*, NJ : Prentice Hall International, 1991.

[10] R. Logeswaran and C. Eswaran, “Neural network based lossless coding schemes for telemetry data”, *IEEE International Geoscience and Remote Sensing Symposium*, Vol.4, 1990, pp. 2057-2059.

[11] R.Logeswaran, “Transmission issues of artificial neural networks in a prediction-based lossless data compression scheme”, *IEEE International Conference on Telecommunications*, Vol.1, 2001, pp. 578-583.