*Journal of Universal Computer Science, vol. 10, no. 9 (2004), 1199-1211*
*submitted: 9/11/03, accepted: 9/6/04, appeared: 28/9/04 © J.UCS*

# Fast Two-Stage Lempel-Ziv Lossless Numeric Telemetry Data Compression Using a Neural Network Predictor

**Rajasvaran Logeswaran**
(Multimedia University, Malaysia.
loges@ieee.org)

**Abstract:** Lempel-Ziv (LZ) is a popular lossless data compression algorithm that produces good compression performance, but suffers from relatively slow processing speed. This paper proposes an enhanced version of the Lempel-Ziv algorithm, through incorporation of a neural pre-processor in the popular predictor-encoder implementation. It is found that in addition to the known dramatic performance increase in compression ratio that multi-stage predictive techniques achieve, the results in this paper show that overall processing speed for the multi-stage scheme can increase by more than 15 times for lossless LZ compression of numeric telemetry data. The benefits of the proposed scheme may be expanded to other areas and applications.

**Keywords:** Lempel-Ziv, neural networks, prediction, lossless compression, two-stage
**Categories:** H.3.m, E.2

## 1 Introduction

Lossless data compression, emphasizing on the accuracy of compressed data, has many well known algorithms ranging from the primitive null suppression [Nelson 96], to well known statistical techniques such as Huffman [Huffman 52] and arithmetic coding [Langdon 84], and combination strategies, e.g. Sixpack, gzip, ARJ, LHA etc. [Nelson 96]. The popular Lempel-Ziv (LZ) [Ziv 77] dictionary algorithm scheme (and its many variants) traditionally produce good compression performance but at a relatively slow processing speed for use in many practical applications.

This paper proposes using a neural network predictor in a two-stage scheme to speed-up the LZ implementation for lossless data compression. It is known that multi-stage predictive schemes [McCoy 94] improve compression performance dramatically, especially for data that contain repetative sequences or vary gradually. Using such an implementation, it is shown by the results in this paper that in addition to improved compression performance, the proposed scheme of incorporating a neural predictive pre-processor to the LZ is capable of increasing processing speeds significantly, even for numeric multi-distribution telemetric data. As LZ algorithms are widely used in many applications, a fast and improved implementation would have far-reaching benefits. In addition, its incorporation with yet another popular scheme of a predictive pre-processor (i.e. neural networks), and the significant merits gained through the combination, should prove beneficial for an even wider range of applications.

## 2 Lempel-Ziv Algorithm Basics

A dictionary-type 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, possibly initialised with pre-defined common matches at start-up).

The LZ (also known as Ziv-Lempel and LZ77) [Ziv 77] has a text window divided into two parts : the body (containing recently encoded text), and a look-ahead buffer (that contains the new input to be compressed). The algorithm attempts to match the contents of the buffer to a string in the dictionary (body of text window), as shown in Fig. 1. Essentially, compression is achieved by replacing variable-length text with fixed sized tokens, each consisting of 3 parts : a pointer into the dictionary, the length of the phrase and the first symbol in the buffer that follows the phrase (example, Fig. 1(b)). The token is read, the indicated phrase is output and the remaining character is appended. The process is repeated until the end of the input.

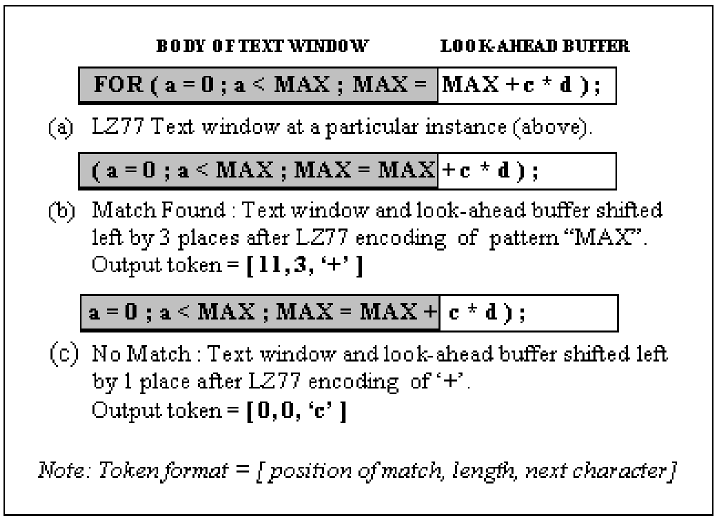
BODY OF TEXT WINDOW
LOOK-AHEAD BUFFER
FOR ( a = 0 ; a < MAX ; MAX =
MAX + c * d ) ;
(a) LZ77 Text window at a particular instance (above).
( a = 0 ; a < MAX ; MAX = MAX
+ c * d ) ;
(b) Match Found : Text window and look-ahead buffer shifted left by 3 places after LZ77 encoding of pattern "MAX".
Output token = [ 11, 3, '+' ]
a = 0 ; a < MAX ; MAX = MAX +
c * d ) ;
(c) No Match : Text window and look-ahead buffer shifted left by 1 place after LZ77 encoding of '+'.
Output token = [ 0, 0, 'c' ]
Note: Token format = [position of match, length, next character]

*Figure 1: Example of Lempel-Ziv Coding (LZ77)*

The implementation of LZ77 can cause a bottleneck due to relatively slow string comparison that has to be done at every position in the text window. When matching strings are not found, the compression cost of using this algorithm is high as the 24-bit token (three 8-bit characters) is used to encode each unmatched 8-bit character (see Fig. 1(c)). Many improved variants of the LZ class of algorithms have been created, including LZSS, LZ78, LZW, LZJ, LZT, LZC, LZH and LZARI [Nelson 96, Storer 82, Ziv 78].

Two LZ variations are implemented in this paper. LZSS [Storer 82] updates a binary search tree with the phrases moved out of the buffer into the dictionary, allowing quick string comparison, thus reducing the performance bottleneck. Wastage is reduced by modifying the token to use a 1-bit prefix to indicate whether an offset (length pair, e.g. 17-bit token [0, (11,3)]) or a single symbol (e.g. 9-bit token [1, '+']) is sent as output. When unmatched, the dummy position 0 and length 0 will not be sent, thus saving 15 bits as compared to the implementation in Fig. 1(c). LZARI [Nelson 96] is a multilevel coding technique that uses a two-pass operation - the first uses one of the better LZ algorithms, followed by arithmetic coding [Langdon 84] of the tokens (code pointers).

## 3 Enhanced Two-Stage Lempel-Ziv

The proposed two-stage LZ scheme integrates a predictor (coupled with a residue generator) as a pre-processor to the LZ encoder, as shown in Fig. 2. The 1st stage is a $p^{th}$ order predictor which reduces the dynamic range of the input by predicting the current input value ($X_n$) at each iteration, and outputs the corresponding residue, $R_n = X_n - E(X_n)$, which is then LZ encoded in the 2nd stage to produce the transmitted compressed residue ($Y_n$).

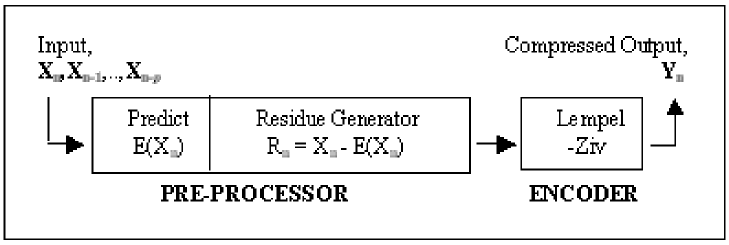

*Figure 2: Two-stage Scheme*

## 4 Performance Results

The simulation model in Fig. 3 is set up for performance evaluation, catering for the training requirements of the pre-processor to allow for adaptive prediction of the varying input patterns. A variety of predictors are evaluated, as described in Section 4.2. As the LZ algorithm has undergone many improvements over the years since its conception, the popular LZSS and LZARI are used instead of the original LZ77, for a more realistic implementation. As LZARI is already a multi-stage encoder, its use is suitable for also studying the performance effects of further incrementing the number of stages in an algorithm.

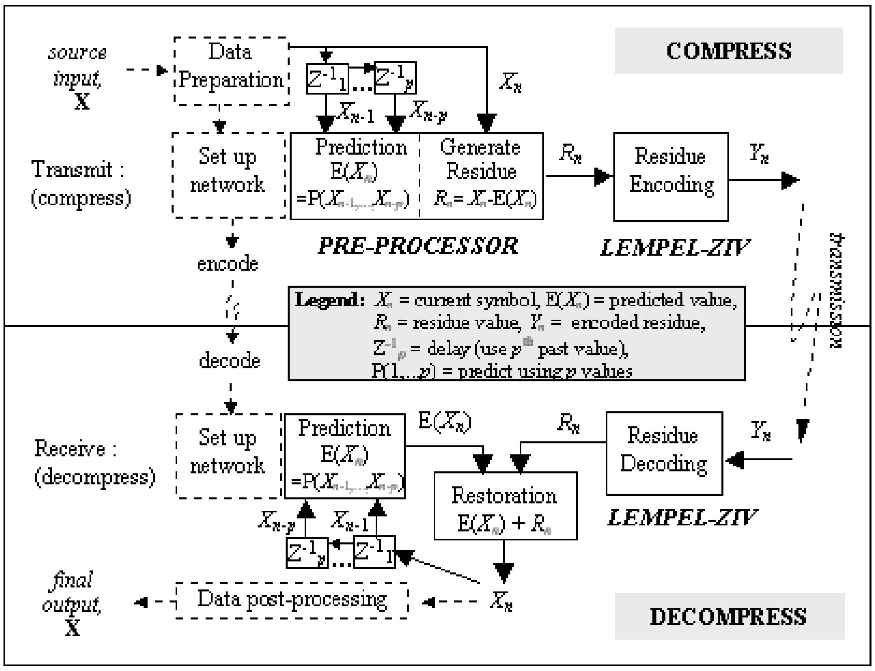

*Figure 3: Implementation schematic of the two-stage scheme*

## 4.1 Test Data

Evaluation of the scheme is carried out with a data set of typical 1-D numeric telemetry data files acquired from the various sensors aboard satellite launch vehicles, which contain data of varying magnitude and mixed distribution patterns. General characteristics of the test files are provided in Tab. 1 [Rahaman 97].

| | | | | **Parameter 1** | | | | **Parameter 2** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Test File** | **File Size (bytes)** | **Total no. of symbols (each param.)** | **Sample rate (symbols / 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 |

*Table 1: Characteristics of the test data files*

Each test file consists of pairs of values – the first parameter being a reference value (e.g. coordinates, time etc.) with semi-linear distribution, and the latter the actual measurement of varying distributions, as signified by the frequency and entropy ($H = -\sum_{i=1}^{n} P_i \log_2(P_i)$) information in Tab. 1. As a reference, an example of the data distribution of the separate parameters of one of the test files is given in Fig. 4. This test set enables more robust testing of the algorithm, as the distribution patterns within the input are more difficult to detect and adapt to.

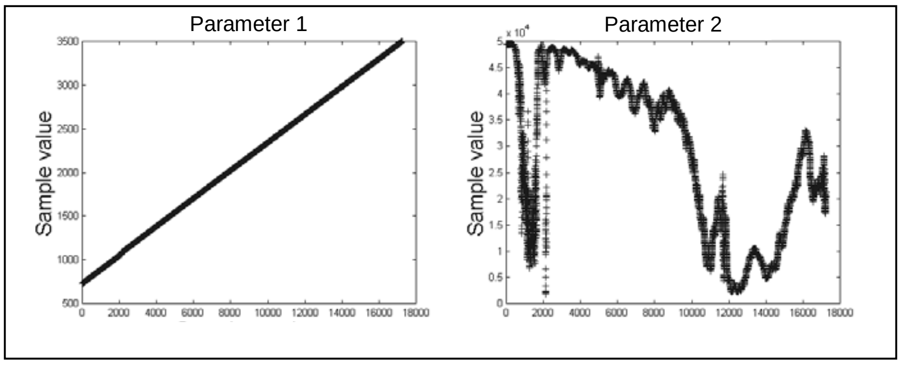

*Figure 4: Distribution of the separate parameters of test file tdata5*

## 4.2 Pre-processor Predictor Selection

A variety of predictors may be used in the 1st stage. The results for some classical and artificial neural network (ANN) predictors evaluated are given in this paper, namely: ANN - 4th-order single layer perceptron (SLP) and 3rd-order multi-layer perceptron (MLP) with 2 hidden nodes [Logeswaran 01]; Classical – 5th-order fixed FIR filter, adaptive FIR with normalized least mean squared (NLMS) algorithm and 2-layer recursive least squares lattice filter with a-priori estimation errors and error feedback (RLSL) [Haykin 91, McCoy 94]. Tests with other predictors were found to produce similar conclusions [Logeswaran 02]. The ANN were trained using the common Backpropagation with a learning rate of 0.1. The topology of the pre-processors setup were intentionally unoptimized to test the scheme for simple implementation on small encoders with minimal overheads.

To handle the dual distribution of the test data, two predictors are used in the 1st-stage - one per parameter, to produce the best match of the input pattern. The output of the 1st stage is interleaved such that the output (i.e. input to the LZ encoder in the 2nd stage) is again pairs of numbers of residues of the parameters, in the sequence of the original input files. NLMS and RLSL are fully adaptive algorithms, retrained (adaptive coefficients) as each new value is presented. For the ANN, training is conducted via a block-adaptive technique [Logeswaran 02], where the incoming input is split into blocks and the networks are retrained on-the-fly for each block using the first 20% of samples in the block. This technique prevents the over-fitting of the ANN (thus preventing rigid predictors), yet allows some adaptability to the input patterns.

The static FIR filter is non-adaptive and requires no training - its coefficients are determined experimentally [McCoy 94].

### 4.3 Processing Time Results

The processing time of the encoders in the single-stage (LZ only) and two-stage (predictor-LZ combination) schemes is summarized in Tab. 2. It is shown that the processing time of the LZSS and LZARI by themselves is large (more than 8 seconds), but is significantly reduced (down to 5.41 % of the original time) when the pre-processing stage is introduced, making the proposed multi-stage implementation faster by up to 17.5 times. The training times are not included in the table as in conventional implementations, the predictors would be intitialized to optimum settings, and adaptive training is handled concurrently during the prediction process.

| **Test File** | **LZ only** | | **Classical – LZ (Two-stage)** | | | | | | **ANN – LZ (Two-Stage)** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **LZSS** | **LZARI** | **FIR - LZSS** | **FIR - LZARI** | **NLMS - LZSS** | **NLMS - LZARI** | **RLSL - LZSS** | **RLSL - LZARI** | **SLP - LZSS** | **SLP - LZARI** | **MLP - LZSS** | **MLP - LZARI** |
| *tdata1* | 7.18 | 7.25 | 1.52 | 1.53 | 0.98 | 0.98 | 0.84 | 0.88 | 0.61 | 0.54 | 0.58 | 0.58 |
| *tdata2* | 12.59 | 12.70 | 0.65 | 0.67 | 0.44 | 0.45 | 0.62 | 0.61 | 0.56 | 0.47 | 0.52 | 0.51 |
| *tdata3* | 4.09 | 3.98 | 0.46 | 0.44 | 0.32 | 0.33 | 0.28 | 0.25 | 0.44 | 0.36 | 0.40 | 0.40 |
| *tdata4* | 7.45 | 7.56 | 0.92 | 0.90 | 0.78 | 0.78 | 0.60 | 0.55 | 0.50 | 0.41 | 0.46 | 0.45 |
| *tdata5* | 8.71 | 8.81 | 1.57 | 1.44 | 1.56 | 1.93 | 0.86 | 0.78 | 0.73 | 0.64 | 0.69 | 0.68 |
| *tdata6* | 10.95 | 10.78 | 0.61 | 0.58 | 0.51 | 0.48 | 0.41 | 0.38 | 0.44 | 0.36 | 0.40 | 0.40 |
| **Avg. Time (sec.)** | **8.50** | **8.51** | **0.96** | **0.93** | **0.77** | **0.83** | **0.60** | **0.58** | **0.55** | **0.46** | **0.51** | **0.50** |
| **% of original time** | 100 | 100 | 11.29 | 10.93 | 9.06 | 10.31 | 7.06 | 6.82 | 6.47 | 5.41 | 6.00 | 5.88 |
| **Speed up (no. of times faster)** | 0 | 0 | 7.6 | 8.1 | 10.0 | 8.7 | 13.2 | 13.7 | 14.5 | 17.5 | 15.7 | 16.0 |

*Table 2: Overall encoding processing time (seconds) for the LZ schemes*

From Fig. 5, it is observed that the processing time for the LZSS and LZARI (and their two-stage schemes) are similar, but processing time achieved vary between the type of predictors used, where the ANN schemes are seen to perform better than the classical predictors. It is also important to note that through the use of the pre-processor, the total processing time for the different files are relatively consistent and low, as opposed to the large differences observed for the LZ-only implementations across the different distribution patterns in the test files. These patterns do also affect predictors in the two-stage scheme, but as the chosen predictors are fast and take up only approximately 0.2-0.3 seconds in processing time, the influence on overall performance of the two-stage scheme is minimal.

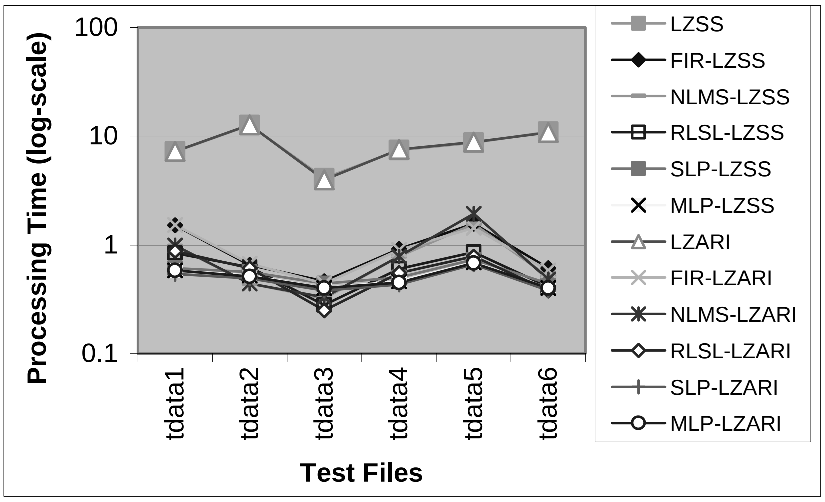

*Figure 5: Processing time achieved by the encoder-only and two-stage schemes for the different test files*

| Test File | LZSS | | | LZARI | | |
|---|---|---|---|---|---|---|
| | Parameter 1 (semi-linear) | Parameter 2 (varied) | **Total Time** | Parameter 1 (semi-linear) | Parameter 2 (varied) | **Total Time** |
| *tdata1* | 2.01 | 0.73 | 2.74 | 7.23 | 2.43 | 9.66 |
| *tdata2* | 2.50 | 0.36 | 2.86 | 4.97 | 1.48 | 6.45 |
| *tdata3* | 1.17 | 0.29 | 1.46 | 1.86 | 0.80 | 2.66 |
| *tdata4* | 2.18 | 0.80 | 2.98 | 7.58 | 2.04 | 9.62 |
| *tdata5* | 3.54 | 1.33 | 4.87 | 6.99 | 2.54 | 9.53 |
| *tdata6* | 1.75 | 0.19 | 1.94 | 2.78 | 0.48 | 3.26 |
| **Average time for separate parameter encoding (seconds)** | **2.19** | **0.62** | **2.81** | **5.24** | **1.63** | **6.86** |
| Average time for combined encoding (seconds) | - | - | 8.50 | - | - | 8.51 |
| **Time gained (seconds)** | | | 5.69 | | | 1.65 |

*Table 3: Processing time (seconds) for each parameter compressed separately using only the LZ algorithm*

Incidently, results of processing the individual parameters separately are given in Tab. 3 to compare the LZ performance for the different types of distribution of the input parameters. Whilst the results in Tab. 2 show similar total processing times by both the LZSS and LZARI schemes, Tab. 3 shows that there is significant difference in speed when the parameters are treated separately. The split allows much faster encoding of the each parameter and in the total processing time as a whole, more so in LZSS than in LZARI. The separate parameter implementation works faster since the matches in the dictionary are located faster as the dictionary for each parameter is smaller than the whole put together. The results also suggest that there are more unique values in the 1st parameter (based on higher processing time) than in the 2nd parameter, in accordance with Tab. 1. In the case of LZARI, being multi-stage (LZ with arithmetic coding), it has larger intrinsic overheads, thus its gain in processing speed is comparably less than that of the LZSS.

| **Test File** | **LZ only** | | **Classical - LZ (Two-stage)** | | | | | | **ANN – LZ (Two-stage)** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | **FIR -** | | **NLMS -** | | **RLSL -** | | **SLP -** | | **MLP -** | |
| | **LZSS** | **LZARI** | **LZSS** | **LZARI** | **LZSS** | **LZARI** | **LZSS** | **LZARI** | **LZSS** | **LZARI** | **LZSS** | **LZARI** |
| *tdata1* | 7.83 | 21.31 | 5.22 | 5.22 | 21.09 | 21.09 | 32.32 | 32.32 | 21.99 | 24.63 | 23.15 | 25.38 |
| *tdata2* | 6.49 | 16.45 | 5.10 | 5.10 | 22.03 | 22.03 | 42.88 | 42.88 | 18.42 | 25.34 | 28.70 | 34.99 |
| *tdata3* | 5.54 | 11.57 | 2.82 | 2.82 | 11.75 | 11.75 | 38.31 | 38.31 | 38.93 | 49.57 | 27.77 | 45.61 |
| *tdata4* | 6.53 | 16.02 | 4.47 | 4.47 | 10.94 | 10.94 | 39.78 | 39.78 | 36.65 | 40.32 | 22.24 | 27.80 |
| *tdata5* | 5.54 | 9.58 | 2.75 | 2.75 | 3.09 | 3.09 | 16.36 | 16.36 | 5.45 | 7.40 | 5.25 | 6.34 |
| *tdata6* | 5.79 | 13.10 | 3.58 | 3.58 | 44.46 | 44.46 | 72.17 | 72.17 | 55.08 | 102.06 | 70.15 | 132.36 |
| **Average CR** | **6.29** | **14.67** | **3.99** | **3.99** | **18.89** | **18.89** | **40.30** | **40.30** | **29.42** | **41.55** | **29.54** | **45.41** |

*Table 4 : Compression ratio performance achieved by LZ and LZ two-stage schemes*

## 4.4 Compression Performance Results

When dealing with compression, it is important to ensure high compression ratios (CR) are obtained. The results in Tab. 4, in terms of CR achieved for each test file, show that the two-stage scheme produces significant improvement in compression, as expected from a multi-stage implementation. In the case of the fixed FIR, compression performance deteriorates because the fixed predictor is not able to cope with the input pattern well and produced large residues with low frequency distribution, thus requiring larger codewords and causing data expansion (where the resulting CR are less than those achieved by the single stage LZSS and LZARI implementations). The other two-stage schemes performed well, with the classical RLSL producing the best results for LZSS and the ANN MLP the best for LZARI, obtaining CR of above 40. In addition, as with the experiment for processing time, each parameter was coded separately using the LZ algorithms, but no performance gains in terms of CR was achieved due to the very little overlap between the values of the 1st and 2nd parameters. The number of entries in the dictionary were not observably affected, and neither were the codewords nor the total compression performance.

## 5 Discussion

The compression performance results achieved for the multi-stage scheme was higher, as expected. However, the performance of the proposed schemes in terms of processing time is rather surprising, especially as it was significantly faster despite the addition of an additional stage to the LZ algorithm. This was possible as the output of the 1st stage is a binary stream in which each input value ($X_n$) is represented by a residue ($R_n$) that is generally significantly smaller in magnitude than $X_n$. The LZ algorithms used assume input of 16-bit integers, whilst each symbol in the residue stream is $r$ bits (residue word size, can be as low as 2 bits long, the 1st being a sign bit). In effect, in the two-stage scheme, the LZ algorithms process a number of residue symbols at a time to make up the 16 bits, so the total number of iterations of the LZ algorithm is reduced, thus speeding up processing.

A good predictor removes redundancy in the input such that the probability density function of the residues has a mean of 0 and a small standard deviation. As the range of values of $R_n$ is small, the number of unique patterns passed on to the encoder is minimal. The LZ benefits from the reduced size dictionary / lookup table it needs to build and use to store these patterns. The processing time for building, lookup and and associated I/O processes involved with the dictionary, and in producing output, is thus dramatically reduced, especially for long streams of similar residues values. As such, the total processing time is significantly reduced through the introduction of a good predictive pre-processor. A frequency histogram may be plotted to illustrate the performance of the predictor, and the proposed implementation. Fig. 6 shows that the number of unique values of data is dramatically reduced, supported by the significant increase in frequency of a few values (e.g. from a maximum frequency of 7 per symbol to a frequency above 10000 for residue value 0 in the first parameter, and from an average frequency of about 100 to above 2500 for certain residue values in the second parameter). Improving the predictor implementation would enable the pre-processor to achieve an even smaller range of $R_n$, and improve the compression performance further.

The above observations also explain the increases in the compression ratio gained by the proposed scheme as a smaller number of symbols are encoded, and a smaller number of tokens are output. Since the LZ algorithms do not utilize the distribution pattern of the input directly, but rather the actual $X_n$ values, the predictor pre-processor complements it well and produces the marked performance increase in terms of processing time and compression ability achieved.

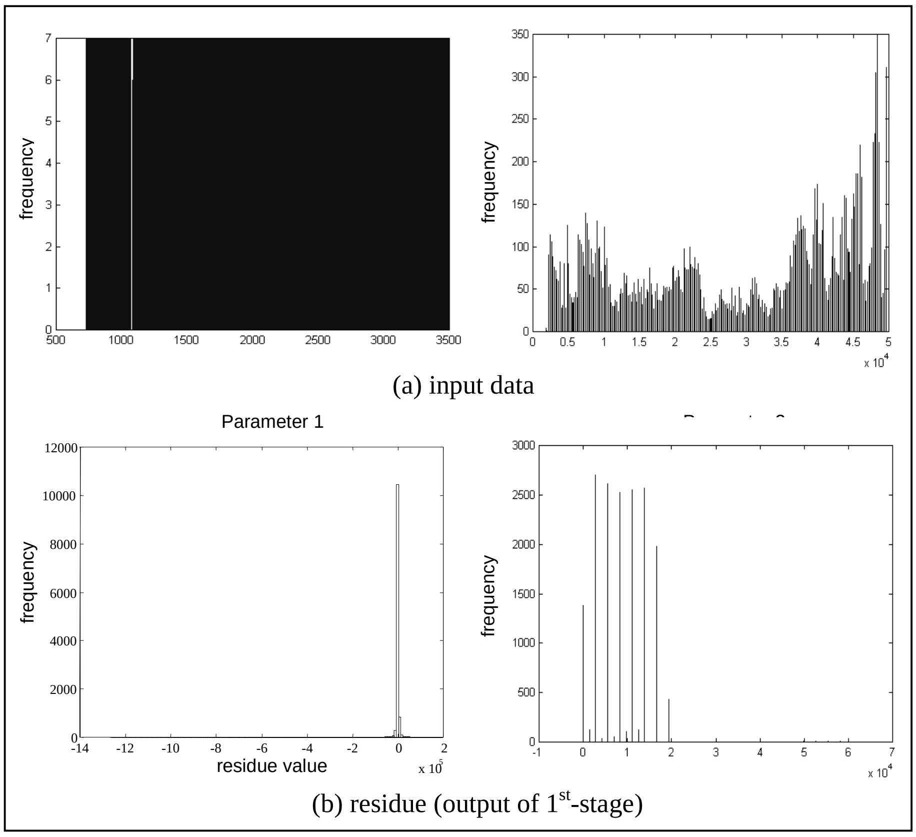

*Figure 6: Frequency histogram of input and the generated residue values of the separate parameters of file tdata5*

The performance of the two-stage scheme is subject to certain overheads. Additional processing power and machine cycles are needed in the 1st stage, along with buffer space for storing the *p* past values. Adaptive predictors require resources for training, thus causing a slight increase in processing time. Identical predictors must be set up at both the transmitter and receiver ends in order to produce the identical predicted values that will be added to the residue to restore the original input values. Certain set up information, such as the number of parameters, pseudo-random generator key, chosen training algorithm, initial *p* value to initiate prediction etc. may need to be transmitted as preamble for implementation of flexible predictors. Such implementation may reduce the compression performance slightly, depending on the available resources and amount of hardware and software customisation and flexibility required. In most cases, a system for data compression tends to have a specific application (e.g. in transmitting telemetry data for a certain application frome a remote sensor) and such dedicated systems would be customised, incurring minimum processing and setup overheads.

Implementing a dynamic residue size for each residue value is costly as there has to be an indicator of the number of bits to be read at the receiver end to restore the value. To minimize this overhead, past experience with similar types of data is used to select an "optimum" word-size (*r* bits) for the residues (e.g. 3 bits), as undertaken in this simulation. If a residue requires a larger word-size (e.g. in the event of signal impulse or very irregular input values), the residue is discarded. Instead, the actual input value (*v* bits) is transmitted preceded by an *r* bit flag (e.g. -0, zero with the sign bit set to negative). Each value transmitted is either *v* or *r* bits, thus minimizing the chances of data expansion (although expansion occurs if *r* is chosen poorly as many flags would be transmitted).

It must also be noted that the test data sets used 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 displaying the capabilities of the LZ algorithms in compressing such numeric data, and for ease of implementation of the predictors. The data chosen was also dual parameter and of varying distribution to better test the proposed scheme, as the input values would not follow a predictable sequence due to the interleaving of the dramatically different parameter values. For general-purpose testing, standard test sets such as those in the cantebury corpus may be used. In addition, the predictors used were of low complexity (only containing less than 5 nodes) for practical implementation of small systems (which would be expected for remote sensors acquiring the telemetry data).

The adhoc training mechanism used was to present the predictors with incoming data on-the-fly, training with just the first 20% of a block of data. Such testing of the system with no prior knowledge of the data (through random initialization) was undertaken to simulate a robust implementation that learns and adapts to non-standardized input. In conventional systems, there is usually an abundance of past data and systems usually need to only cope with a limited set of input patterns. As such, initializing the predictor through preparation of a proper training dataset containing a good sample of various expected input (and target, for supervised training) patterns, extracted through feature analysis, would certainly enable the predictor, and thus, the two-stage scheme to preform better.

## 6 Conclusion

In this paper, a two-stage scheme is successfully implemented to improve the performance of the Lempel-Ziv (LZ) dictionary-type lossless compression algorithm. The method employed is via integration of a predictive pre-processor stage with the LZ algorithm. Tests conducted with a number of classical and neural predictors, with two LZ algorithms, shows that the multi-stage predictive LZ scheme can significantly speed up processing in addition to improving compression performance. The simulation results for some small known predictors show that processing speeds of up to almost 18 times faster, with compression ratio of more than 45 can be achieved by the proposed two-stage scheme as compared to single-stage LZ-only implementations. 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 LZ encoder must be mentioned [Logeswaran 01, 02].

Through empirical or analytical means, the number and type of pre- (and possibly post-) processing stages used could be determined by the overall performance gains versus the resources requirements for the additional integration. The generalized scheme presented here may be expanded to larger systems and is expected to produce better results with the use of customised predictors. The results recommend that with a suitable pre-processor stage, performance of the encoder can be improved significantly, when dealing with different data. This is true not just for the test set of telemetry data used in this paper, selected for their varying paterns in data distribution, but for any data that has patterns in the input that may be predicted. Based on the popularity of the LZ algorithms and the predictor-encoder schemes, the proposed scheme may prove beneficial for a large domain of applications. It is also noteworthy that ANNs tend be faster and more error tolerant than their classical counterparts [Logeswaran 00], making then also the more suitable predictors for critical systems.

**Acknowledgements**

The author acknowledges the contributions and guidance of Prof. C. Eswaran and M.U. Siddiqi of Multimedia University, Malaysia in conducting this work.

## References

[Haykin 91] Haykin, S.: “Adaptive filter theory”; Prentice Hall International, New Jersey (1991)

[Huffman 52] Huffman, D.A.: “A method for the construction of minimum redundancy codes”; Proc. of the IRE, 40 (1952), 1098-1101.

[Langdon 84] Langdon, G.G.: “An introduction to arithmetic coding”; J. IBM R&D (1984)

[Logeswaran 00] Logeswaran, R. and Siddiqi, M.U.: “Error Tolerance in Classical and Neural Network Predictors”; IEEE TenCon, 1 (2000), 7-12

[Logeswaran 01] Logeswaran, R. : “Transmission Issues of Artificial Neural Networks in a Prediction-based Lossless Data Compression Scheme”; IEEE Int. Conf. on Telecommunications, 1 (2001), 578-583

[Logeswaran 02] Logeswaran, R.: “A Prediction-Based Neural Network Scheme for Lossless Data Compression”; IEEE Trans. on Systems, Man, and Cybernetics - Part C: Applications and Reviews, 32, 4 (2002), 358-365

[McCoy 94] McCoy, J.W., Magotra, N. and Stearns, S.: “Lossless predictive coding”; IEEE Midwest Symp. on Circuits and Systems (1994), 927-930

[Nelson 96] Nelson, M. and Gailly, J.L.: “The data compression book”; M&T Books, New York (1996)

[Rahaman 97] Rahaman, F.: “Adaptive entropy coding schemes for telemetry data compression”; Project Report No. EE95735, Indian Institute of Technology, Madras, India (1997)

[Storer 82] Storer, J.A. and Szymanski, T.G.: “Data compression via textual substitution”; J. ACM, 29, 4 (1982), 928-951

[Ziv 77] Ziv, J. and Lempel, A.: “A universal algorithm for sequential data compression”; IEEE Trans. on Information Technology, 23, 3 (1977), 337-343

[Ziv 78] Ziv, J. and Lempel, A.: “Compression of individual sequences via variable-rate coding”; IEEE Trans. for Information Technology, 24, 5 (1978), 530-536