Pattern Based Lossles Data Compression

Angel Kuri Morales
Instituto Tecnológico Autónomo de México
Río Hondo No. 1
México 01000, D.F.
akuri@itam.mx

Abstract. In this paper we discuss a method for lossless data compression (LDC) which relies on finding a set of patterns (each of these patterns will be called a *metasymbol*) in a set of data whose elements (which we will call *symbols*) are of arbitrary size and which is, itself, also of arbitrary size. This arbitrary data set will be called a *message*. In order to achieve LDC two things are necessary: a) A method to find the metasymbols and b) A scheme to represent the message as a function of these metasymbols. In the past, LDC has been attempted, among other methods, by using the probability that a given symbol or combination of symbols appear in the message (as in Huffman and PPM encoding schemes) or by keeping a record of the last K symbols in the message's stream and using references to this record to represent the data (as in the several variations of Lempel-Ziv compression schemes). In both of the aforementioned approaches to LDC the structure of the premium data structure on which the method is based is fixed *a priori*. Furthermore, the compression ratio of both of these approaches, changes even in the presence of similar patterns in the structure of the message. The structure of the metasymbols in our approach, however, does not depend on aprioristic considerations. In fact, the structure of every metasymbol is arbitrary and , in general, different from every other's one. We show that Metasymbolic Lossless Data Compression (MLDC) is dependent on the structure of the patterns of symbols and NOT on the symbols and under which conditions MLDC is superior to other LDC schemes.

Keywords. Data, compression, losslessness, encoding, information theory, ergodicity.

1 Introduction

In this paper we address the problem of achieving the most compact representation of an arbitrary set of data without losing any of the information contained in it, a goal that is generally called lossless data compression (LDC). Alternatively, one may attempt to compress data with a given acceptable (under some measure) loss of its contents; something we refer to as lossy data compression. LDC may be very broadly classified considering whether such compression is achieved by taking into account the probabilities of the symbols in a message or not. For this classification to make sense, we must define a *symbol*. Intuitively, a symbol is the smallest unit of information contained in a message. In this regard it is clear that any message composed of symbols may be reconstructed by orderly enumerating them. However, given the digital and usually binary nature of present day computers and data channels, it is clear that a symbol may be defined as an n-ary collection of bits and that, although not generally made explicit, the choice of n is arbitrary. Typically, however, the choice corresponds to n=8 where this selection corresponds to a byte (or character, when the analyzed data is a text). In some compression schemes this choice is taken one step forward by considering compression optimized to streams of characters or words, where the regularity present in a given language yields some purported advantage.

Furthermore, it is tacitly assumed in all LDC schemes up to date, that the sought for regularity is present in neighboring symbols and no attempt has been made to try to find such regularity in arbitrary clusters of symbols.

If, however, no a priori consideration is made on the size of a symbol we may take one further step and validly ask whether the structure of such symbol is to be taken for granted. That is, we may not only consider the size of groups of adjacent bits, but rather, groups of possibly non-adjacent bits. In what follows we assume, for the sake of simplicity and without loss of generality, that we arbitrarily set the size of a group to 8 (i.e. symbols are of length 8). Likewise we explore the possible advantages of attempting LDC by identifying clusters of symbols with arbitrary structure. These we call *metasymbols*. We

show that when such identification is possible, it is also possible to encode a given message by simply representing the message as a collection of metasymbols. From the analysis of the detailed representation it is easy to find the bounds on metasymbolic structure in order to achieve LDC. The rest of the paper is organized as follows: in section 2 we make a brief summary of previous approaches to LDC. In section 3 we introduce a novel approach to LDC based on the discovery of patterns of arbitrary structure. In section 4 we describe a set of tests which we performed in order to establish the relative behavior of traditional LDC techniques versus our own. Finally, in section 5 we offer some conclusions and point to future lines of research.

2 Previous approaches to LDC

Several previous attempts to LDC have been tried. In what follows we give a very brief account of some of these.

2.1 Huffman Coding and Related Techniques

2.1.1 Huffman and Shannon-Fano Coding

Huffman coding is a statistical data compression technique which gives a reduction in the average code length used to represent the symbols of an alphabet. The Huffman code is an example of a code which is optimal in the case where all symbols probabilities are integral powers of 1/2. A Huffman code can be built in the following manner:

(1) Rank all symbols in order of probability of occurrence.

(2) Successively combine the two symbols of the lowest probability to form a new composite symbol; eventually we will build a binary tree where each node is the probability of all nodes beneath it.

(3) Trace a path to each leaf, noticing the direction at each node.

For a given frequency distribution, there are many possible Huffman codes, but the total compressed length will be the same. It is possible to define a "canonical" Huffman tree, that is, pick one of these alternative trees. Such a canonical tree can then be represented very compactly, by transmitting only the bit length of each code. This technique is used in most archivers.

A technique related to Huffman coding is Shannon-Fano coding, which works as follows:

(1) Divide the set of symbols into two equal or almost equal subsets based on the probability of occurrence of characters in each subset. The first subset is assigned a binary zero, the second a binary one.

(2) Repeat step (1) until all subsets have a single element.

The algorithm used to create the Huffman codes is bottom-up, and the one for the Shannon-Fano codes is top-down. Huffman encoding always generates optimal codes, Shannon-Fano sometimes uses a few more bits. The interested reader is referred to [1].

2.1.2 Arithmetic Coding

It would appear that Huffman or Shannon-Fano coding is the perfect mean of compressing data. However, this is not the case. As mentioned above, these coding methods are optimal when and only when the symbol probabilities are integral powers of 1/2, which is usually not the case. The technique of arithmetic coding does not have this restriction. It achieves the same effect as treating the message as one single unit (a technique which would, for Huffman coding, require enumeration of every single possible message), and thus attains the theoretical entropy bound to compression efficiency for any source.

Arithmetic coding works by representing a number by an interval of real numbers between 0 and 1. As the message becomes longer, the interval needed to represent it becomes smaller and smaller, and the number of bits needed to specify that interval increases. Successive symbols in the message reduce this interval in accordance with the probability of that symbol. The more likely symbols reduce the range by less, and thus add fewer bits to the message.

2.1.3 DMC and PPM Coding

Using the symbol probabilities by themselves is not a particularly good estimate of the true entropy of the data: We can take into account intersymbol probabilities as well. Some of the best compressors available today take this approach: DMC (Dynamic Markov Coding) starts with a zero-order Markov model and gradually extends this initial model as compression progresses; PPM (Prediction by Partial Matching) looks for a match of the text to be compressed in an order-n context. If no match is found, it drops to an order n-1 context, until it reaches order 0. Both these techniques thus obtain a much better model of the data to be compressed, which, combined with the use of arithmetic coding, results in superior compression performance. For a more detailed description of PPM principles, see [2].

2.2 Substitutional Compressors

The basic idea behind a substitutional compressor is to replace an occurrence of a particular phrase or group of bytes in a set of data with a reference to a previous occurrence of that phrase. There are two main classes of schemes, named after Jakob Ziv and Abraham Lempel, who first proposed them in 1977 and 1978.

2.2.1 The LZ78 family of compressors

LZ78-based schemes work by entering phrases into a *dictionary* and then, when a repeat occurrence of that particular phrase is found, outputting the dictionary index instead of the phrase. There exist several compression algorithms based on this principle, differing mainly in the manner in which they manage the dictionary. The most well-known scheme (in fact the most well-known of all the Lempel-Ziv compressors, the one which is generally referred to as "Lempel-Ziv Compression"), is Terry Welch's LZW scheme, which was designed in 1984 for implementation in hardware for high-performance disk controllers. LZW starts with a 4K dictionary, of which entries 0-255 refer to individual bytes, and entries 256-4095 refer to substrings. Each time a new code is generated it means a new string has been parsed. New strings are generated by appending the current character K to the end of an existing string w.

The most remarkable feature of this type of compression is that the entire dictionary is transmitted to the decoder without actually explicitly transmitting the dictionary. At the end of the run, the decoder will have a dictionary identical to the one the encoder has, built up entirely as part of the decoding process.

LZW is more commonly encountered today in a variant known as LZC, after its use in the UNIX "compress" program. In this variant, pointers do not have a fixed length. Rather, they start with a length of 9 bits, and then slowly grow to their maximum possible length once all the pointers of a particular size have been used up. Furthermore, the dictionary is not frozen once it is full (as for LZW) the program continually monitors compression performance, and once it starts decreasing the entire dictionary is discarded and rebuilt from scratch. More recent schemes use some sort of least-recently-used algorithm to discard little-used phrases once the dictionary becomes full rather than discarding the entire dictionary.

2.2.2 The LZ77 family of compressors

LZ77-based schemes keep track of the last *n* bytes of data seen, and when a phrase is encountered that has already been seen, they output a pair of values corresponding to the position of the phrase in the previously-seen buffer of data, and the length of the phrase. In effect the compressor moves a fixed-size *window* over the data (generally referred to as a *sliding window*), with the position part of the (position, length) pair referring to the position of the phrase within the window. The most commonly used algorithms are derived from the LZSS scheme described by James Storer and Thomas Szymanski in 1982. In this the compressor maintains a window of size N bytes and a *lookahead buffer* the contents of which it tries to find a match for in the window.

Decompression is simple and fast: Whenever a (position, length) pair is encountered, go to that (position) in the window and copy (length) bytes to the output.

Sliding-window-based schemes can be simplified by numbering the input text characters mod N, in effect creating a circular buffer. The sliding window approach automatically creates the LRU effect which must be done explicitly in LZ78 schemes. Variants of this method apply additional compression to the output of the LZSS compressor, which include a simple variable-length code (LZB), dynamic Huffman coding (LZH), and Shannon-Fano coding (ZIP 1.x), all of which result in a certain degree of improvement over the basic scheme, especially when the data are rather random and the LZSS compressor has little effect.

An algorithm was developed which combines the ideas behind LZ77 and LZ78 to produce a hybrid called LZFG. LZFG uses the standard sliding window, but stores the data in a modified trie data structure and produces as output the position of the text in the trie. Since LZFG only inserts complete *phrases* into the dictionary, it should run faster than other LZ77-based compressors.

2.3 The Burroughs-Wheeler Transform

The BWT [3] is an algorithm that takes a block of data and rearranges it using a sorting algorithm. The resulting output block contains exactly the same data elements that it started with, differing only in their ordering. The transformation is reversible, meaning the original ordering of the data elements can be restored with no loss of information. The BWT is performed on an entire block of data at once. Most of the LDC algorithms operate in streaming mode, reading a single byte or a few bytes at a time. But with BWT operation on the largest sets of data is desirable.

The basic idea behind the BWT is to map the original message into an equivalent one which is better suited

than the original one for an efficient application of a typical encoding process (such as an arithmetic one as described above).

2.4 Problem Oriented Compression

2.4.1 Predictive Encoding

In predictive encoding [4], the idea is to have an "oracle" which adequately predicts the output from the information source on both sides of the communications channel. Once the oracle emits its prediction, it sends" the *number* of correctly predicted bits rather than the bits themselves. On the receiving end, the received data stream is matched versus the predicted one, corrections (if necessary) are made and, thus, the original message is recovered. Clearly, the main problem, in this case, is to ensure a minimum prediction accuracy (or *prediction ratio* ρ). If ρ is large enough, significant compression is achieved. In practice, a Neural Network (NN) may be used as the oracle.

The main drawback of this approach is that an oracle has to be found specifically for each particular data set.

2.4.2 Neural Networks

In [5] a predictive arithmetic encoder called P6 based on a 6-gram character model is described. This algorithm is specifically aimed (the NN is trained for) at texts of the English language. Prediction is done one bit at a time by a two layer neural network with 222 (about 4 million) inputs, and one output. The output is the probability that the next bit will be a 1. The inputs are "context detectors", where the context is the last 1 to 5 complete bytes, plus the 0 to 7 bits of the partially read current byte. An input unit is active (output of 1) when the context matches a particular value, otherwise it is 0. Because there are more than 222 possible context values, the context is hashed to a 22-bit number to select the active input. There are 5 contexts considered, with lengths of 1 to 5 complete bytes. Thus there are 5 out of 222 inputs active at any time. Using this methodology, 42-47% better compression than *gzip-9* is reported. Compression time for 600 KB is attained in 15 seconds (using a 475 MHz P6-II). As in the previous case, the main drawback of this approach is that the NN has to be trained specifically for a particular data set.

2.5 Theoretical Limitations

From the information theoretical point of view, the reason for statistical LDC schemes such as Huffman coding not to achieve optimal performance relates to the following three simple facts:

a) One is unable to work with the probabilities of the basic symbols (because they are, in general, unknown). Instead, one has to use proportions within the realm of the known data. For instance, if in a block of size n a byte (say β), stemming from a given information source S, appears m times, it is common to say "the probability that byte β appears in the data stream generated by source S is *(n-m)/n*", or $P(\beta) = (n - m)/n$. In general, symbols such as β are assumed to be statistically independent which is, of course, an unrealistic assumption. The efficiency of schemes such as PPM and its variations, on the one hand, or DMC, on the other, depends fundamentally on their ability to find second, third, …, n-th order correlations.

b) Once the proportions are calculated, the LDC technique "assumes" that the source exhibits an ergodic behavior and that the aforementioned probabilities are set, at least, for the duration of the data block.

c) The structure of the symbol is assumed to be of the simplest nature, i.e. a symbol is defined as an arbitrary collection of *adjacent* bits. This is not to be taken for granted if theoretical bounds are to be approached in LDC.

With this point of view in mind, substitutional compressors may be seen as methods which are simply trying to dynamically adjust themselves to varying probabilities and to take into consideration correlations of higher order. The order is, in practice, limited by the size of the dictionary.

3 Pattern Based Data Compression

If one abandons the idea of defining symbols as above, and rather searches for *patterns* whose structure is not defined *a priori* statistical independence of such patterns (or metasymbols) is guaranteed, the order of correlation is unbounded and the source (seen as a generator of metasymols) approaches an ergodic behavior.

As a simple example, let us assume that we are faced with a set of data (message 1) such as the one in figures 1a and 1b. Message 1 consists of 512 (32 X 16) arbitrary characters. For presentation purposes it has been split in two parts (16 X 16 each). Figure 1a consists of the leftmost part (columns 1 to 16) of the data set while figure 1b consists of the rightmost part (columns 17 to 32). For convenience, we refer to the ensemble of figures 1a and 1b as "Figure 1" (similar considerations apply to the rest of the figures). The displayed characters

are the printable ASCII characters corresponding to the 8 bits long number. It is not evident which patterns, if any, are to be found in the message. Let us assume that

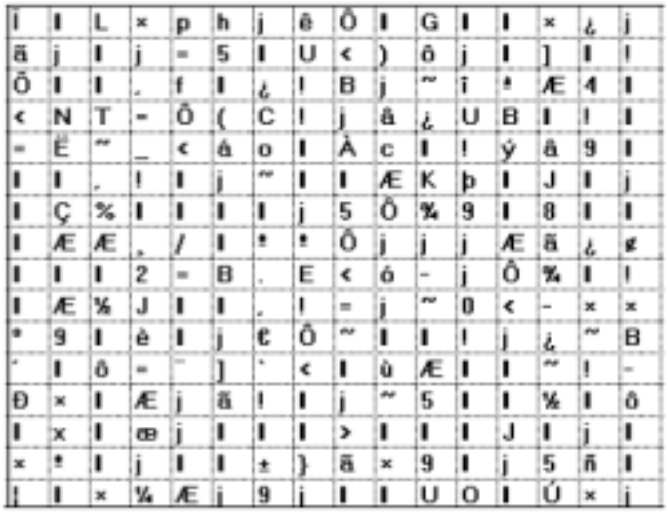

Fig. 1a. A Message

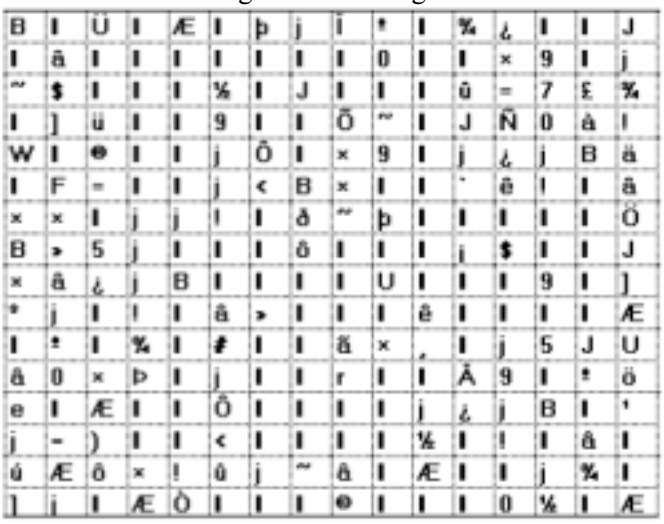

Fig 1b. A Message

we have an efficient algorithm which is capable of finding patterns which repeat themselves throughout the data. One such metasymbol is shown in figure 2.

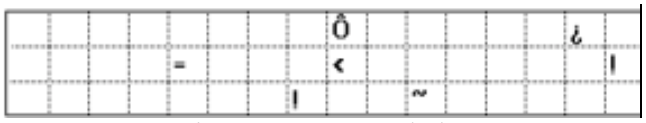

Figure 2a. Metasymbol 1.

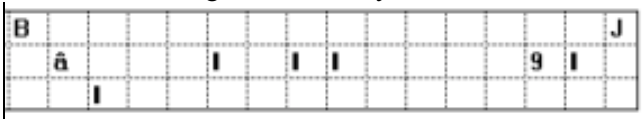

Figure 2b. Metasymbol 1.

The distribution of metasymbol 1 in the message is shown in figure 3. Metasymbol 1 consists of 16 characters. It appears in the message 8 times and accounts, therefore, for 25% of the data. Likewise, we are able to find 4 more metasymbols, shown in figures 4-7.

Metasymbol 2 consists of 10 characters. It appears 5 times and accounts for 9.76% of the data.

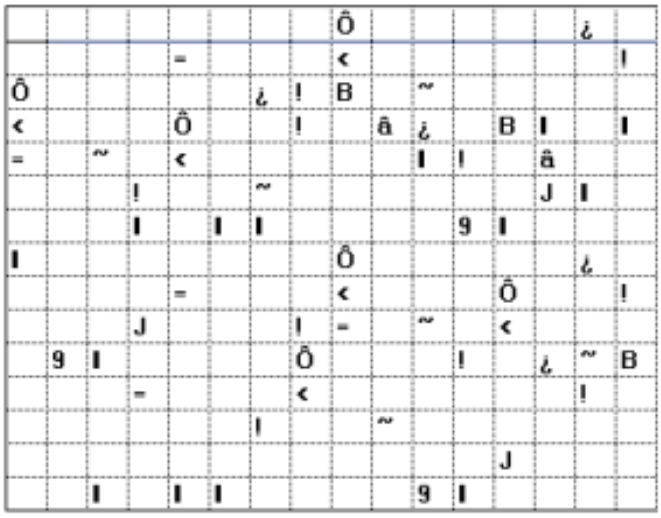

Figure 3a. Occurrences of Metasymbol 1

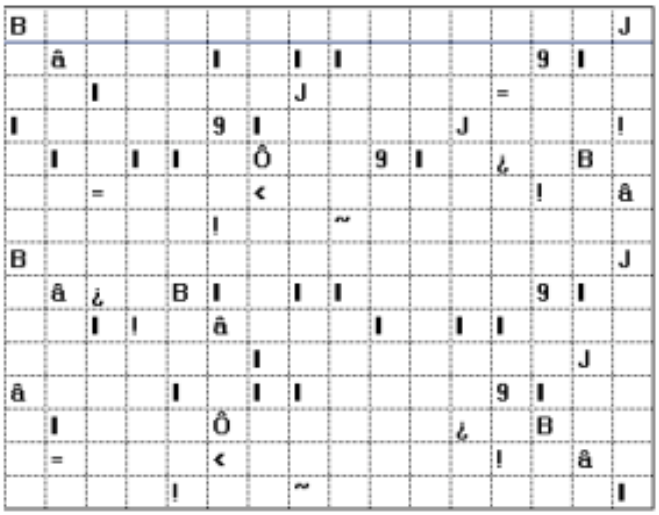

Figure 3b. Occurrences of Metasymbol 1

Fig.4a. Metasymbol 2

Fig.4b. Metasymbol 2

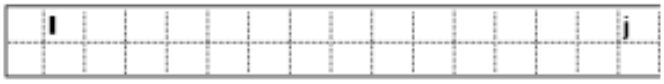

Fig. 5a. Metasymbol 3.

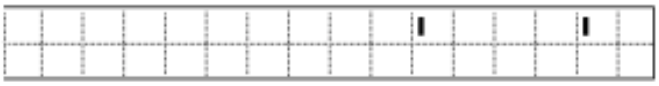

Fig. 5b. Metasymbol 3

Metasymbol 3 consists of 4 characters. It appears 30 times and accounts for 23.44% of the data.

Fig. 6a. Metasymbol 4

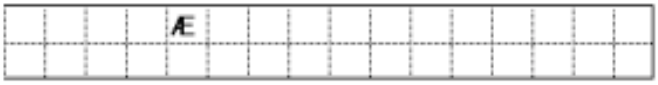

Fig. 6b. Metasymbol 4.

Metasymbol 4 consists of 3 characters. It appears 15 times and accounts for 8.79% of the data.

Fig. 7a. Metasymbol 5

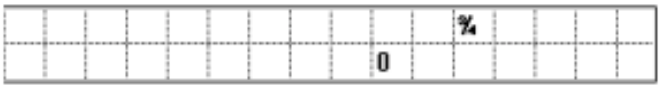

Fig. 7b. Metasymbol 5

Metasymbol 5 consists of 4 characters. It appears 5 times and accounts for 3.9% of the data.

The remaining 149 characters may not be accounted for in any metasymbol. They are shown in figure 8. This set of un-patterned symbols will be called the *filler*.
The reader may verify that, by superimposing all occurrences of metasymbols 1-5 plus the filler, we may reconstruct the message fully.
Even with the metasymbols at hand, it is not a simple task to find the places where they appear in the message. This is so because patterns are not pre-specified in either content or structure. For instance, metasymbol 5, which consists of 4 characters, appears for the first time in position 32, having intersymbolic gaps (or, simply, "gaps") of size 11, 5 and 11 respectively. In contrast, metasymbol 3, which consists of 4 symbols also, appears for the first time in position 2, having gaps of size 13, 10 and 3 respectively.

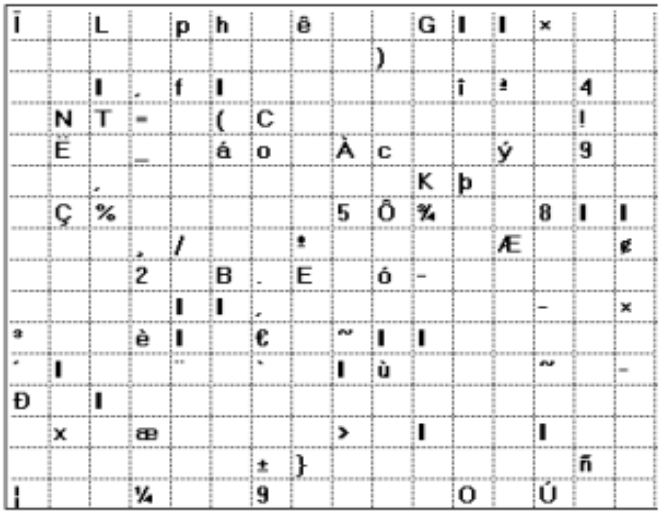

Fig 8a. Filler

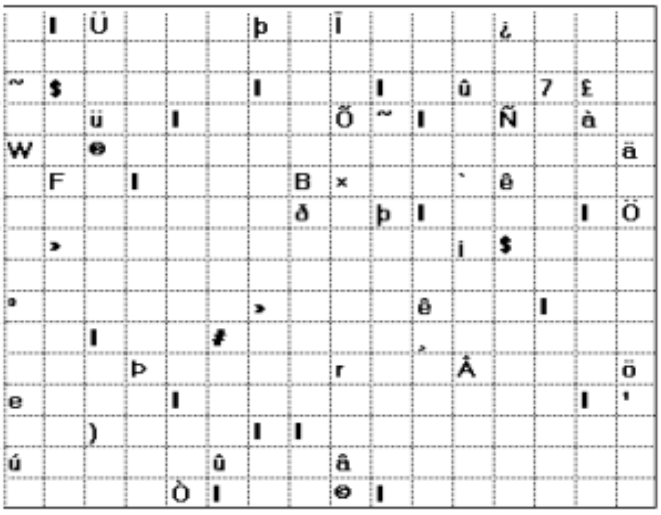

Fig 8b. Filler

We may represent the message above as a combination of metasymbols as follows.

3.1 Algorithm MT: Metasymbolic Transform

1. Create a Header.

The basic characteristics of the encoding process are encoded in a header (in this case, it is 16 bits long). In the header we define:

a) The symbol's length [λ]. As stated in the introduction, the length of a symbol is arbitrary. We use 4 bits to specify its length (i.e. symbols length goes from 1 to 16).

b) The number of bits needed to express the amount of metasymbols found [μ]. Again we use 4 bits (i.e. we provide for up to $2^{\mu-1}$ possible metasymbols).

c) The number of bits needed to express the maximum distance between two consecutive metasymbols [ω]. This distance is expressed relative to preceding metasymbol. By convention, the first occurrence of the first metasymbol is set relative to 0. We call this distance, an *offset*. Hence, we express a set of *offsets*.

d) The number of bits needed to express the maximum *gap* between two consecutive symbols in a given metasymbol [γ]. The position of the first symbol in the *i*-th metasymbol is obtained from (c). With exception of the first symbol, this distance (the *gap*) is expressed relative to the previous symbol.

2. Express the message in metasymbols. Once the metasymbols are identified we re-express the message as a list of metasymbols, plus the filler.
3. Describe the initial position of each one of the metasymbols.
4. Describe the structure of each of the metasymbols.
5. Describe the contents of each of the metasymbols.
6. Describe the contents of the filler

3.1 Higher Order Metasymbolic LDC (MLDC)

The mere re-expression of the message as described is enough to achieve data compression. But, the re-expressed message may be further subject to, say, arithmetic encoding, allowing for further improvements. The MLDC, as developed and tested is what we may call a 0-order MLDC Compression Scheme ($MLDC_0$, for short). In fact, $MLDC_0$ is a transformation method which, like BWT discussed above, may prepare the data for further compression. In this regard, we may say that application of $MLDC_0$ merely transforms a source data set to a target data set. If no compression by traditional schemes is performed on the target data set, $MLDC_0$ is achieved. If, on the other hand, we do some sort post-compression processing then $MLDC_1$ is achieved. Furthermore, second order MLDC ($MLDC_2$) may be achieved by determining distribution frequencies of metasymbols in sets of data whose nature is determined a priori. For instance, one may extensively analyze sets of English texts or sets of .BMP files. These sets may be indexed canonically. Thereafter, the most frequently appearing metasymbolic structures in every predefined set may be indexed canonically as well. The preceding information (the indices of the data sets, the indices of the metasymbolic structures and the structures themselves) is, therefore, stored in a catalog which has to be calculated only once, and made known to the potential receivers of the data sets. Once this is done, one achieves $MLDC_2$ by a) Identifying the proper set (i.e. the index of the set is encoded) and b) Encoding the indices of the metasymbols. This process eliminates the need for step 4 in algorithm MT above.

4 Preliminary test and discussion

In order to tests the behavior of $MLDC_0$, we performed the following test:

1. We randomly generated a patterned file 512 bytes long (the samples above correspond to one such data file). A maximum of 16 symbols and a maximum gap of 16 per metasymbol were allowed.

2. We considered:
   a) A uniform distribution, that is, one in which $P(S_i) = P(S_j) \quad \forall i, j$.
   b) We also obtained the frequencies of all 256 binary combinations in 4 sets of sample data:
   c) An excerpt of Cervantes' novel "El Amante Liberal" (61,233 bytes).
   d) An excerpt of Shakespeare's play "Macbeth" (64,000 bytes).
   e) A .JPG image of Iguazu's cataracts (308,158 bytes).
   f) An .MP3 piano recording of Manzanero's (2,819,971 bytes).
3. We filled the patterns obtained in step 1 with bytes having the same probability distributions as in each of the data samples. That is, we obtained 5 patterned data files whose symbols' probabilities were distributed as a uniform (equi-probable) distribution plus the distributions of the sampled data files.
4. We subjected each of the 5 patterned files to the following algorithms:
   a) $MLDC_0$
   b) LZ77
   c) LZW
   d) Huffman
   e) PPM
5. We calculated the compression ratio (ρ) as $\rho = L/C$.

Where L is the size of the original message and C is the size of the transformed message (in bits). These ratios for the five algorithms are shown in Table 1.

Several points are to be made:

1. First and foremost, in all cases $MLDC_0$ outperformed the *traditional* compressors.
2. The next best compressor was PPMZ2 (a variation of PPM). This is to be expected given that PPM explores larger correlation orders than the rest.

Distribution	Compressor	Compressed File's Size	Compression Ratio
Uniform	**$MLDC_0$**	**281**	**1.8221**
	LZ77	571	0.8967
	LZW	512	1.0000
	HUFFMAN	648	0.7901
	PPMZ2	437	1.1716
Spanish	**$MLDC_0$**	**281**	**1.8221**
	LZ77	539	0.9499
	LZW	392	1.3061
	HUFFMAN	345	1.4841
	PPMZ2	298	1.7181
English	**$MLDC_0$**	**281**	**1.8221**
	LZ77	536	0.9552
	LZW	390	1.3128
	HUFFMAN	341	1.5015
	PPMZ2	297	1.7239
Image	**$MLDC_0$**	**281**	**1.8221**
	LZ77	572	0.8951
	LZW	522	0.9808
	HUFFMAN	657	0.7793
	PPMZ2	447	1.1454
Audio	**$MLDC_0$**	**281**	**1.8221**
	LZ77	572	0.8951
	LZW	524	0.9771
	HUFFMAN	668	0.7665
	PPMZ2	446	1.1480

Table 1. Compressor's Performances

3. In general, traditional methods' performance was relatively better for text files. This may be explained by noting that short "patterns" and correlations are frequent in natural languages such as English and Spanish
4. .JPG and .MP3 data has already undergone a compression (albeit lossy) process. Therefore, with the notable exception of PPM, all traditional compressors yield compression rations *below unity* or, in other words, the "compressed" file is *larger* than the original, uncompressed one.
5. Uniformly distributed data turns out to be almost as impervious to traditional compressors as pre-compressed data. This is explained by noting that the short range and simply patterned search that traditional compressors perform is unsuccessful when range is not short nor patterns are simple; a fact that outstands when data is uniformly distributed.
6. Traditional compressors perform differently for different distributions whereas MLDC, by finding complex patterns, is indifferent to basic, simple probability distributions.

5 Theoretical Bounds

To determine when $MLDC_0$ is effective, we perform the following analysis.

Definitions.

1. Let L denote the size of the original uncompressed message in bits.
2. Let H denote the size of the header in bits. By convention H=16.
3. Let M denote the number of different metasymbols. In the example above M=5.
4. Let N_i denote the number of instances of metasymbol i. For instance, N_1=8 and N_3=30 in the example.
5. Let $|m_i|$ denote the number of symbols of metasymbol i.
6. Let $\lambda, \mu, \omega, \gamma$ be defined as in algorithm MT.

Then C (the length of the $MLDC_0$ transformed message in bits) is expressed by equation (1).

$$C < H + \sum_{i=1}^{M} \mu N_i + \sum_{i=1}^{M} \omega N_i + \sum_{i=1}^{M} \gamma |m_i| + \sum_{i=1}^{M} \lambda |m_i| + \left[L - \sum_{i=1}^{M} |m_i| N_i \right]$$

$$C < H + L + (\mu + \omega) \sum_{i=1}^{M} N_i + (\gamma + \lambda) \sum_{i=1}^{M} |m_i| - \sum_{i=1}^{M} |m_i| N_i \quad (1)$$

To achieve data compression from $MLDC_0$ it is only needed that $L/C>1$. Evidently, if $C<L$ the original data is expanded rather than compressed. This is possible because, in practice, it is needed to store additional information other than the simple data in order to achieve decompression. For instance, in Huffman encoding, we need to include the coding tree; in LZW we need to supply the dictionary, etc. When

compression is low, the advantages derived from it are offset by the need to supply the decoding information. In $MLDC_0$ most of the actual size of the compressed data consists of decoding information. In $MLDC_2$, for example, the size of the compressed data is given by (2) where the term corresponding to metasymbolic structure has been left out because it is no longer needed.

$$C < H + L + (\mu + \omega) \sum_{i=1}^{M} N_i + \lambda \sum_{i=1}^{M} |m_i| - \sum_{i=1}^{M} |m_i| N_i \qquad (2)$$

6 Conclusions

We have illustrated the fact that $MLDC_0$ may yield better potential compression ratios than any of the traditional compression methods. This is, of course, true only if: a) There are patterns to be found in the message, b) C>L in equation (1) and c) We are able to detect the embedded patterns. In [6] we have shown that the problem of finding the metasymbols is of NP complexity. Moreover, it is NP-complete. Therefore, except for the most simple cases which carry no practical interest, there is no deterministic algorithm which guarantees that condition (c) is fulfilled. However, we have been able to approach the theoretical bounds in some cases by applying evolutionary optimization. In particular, as reported in [7] a genetic algorithm with special operators has approached such limits.

There is ample room for further research in this area. For instance, we should show that data sets whose symbols come from different sources (for example, a mixture of image and audio files) when transformed by $MLDC_0$ retain the compression properties. We should also establish the most adequate method for post-processing transformed data to achieve $MLDC_1$. We should determine also the different sets of interesting data and proceed with their cataloguing and indexing to achieve $MLDC_2$. We ought to compare the efficiency of our method (in terms of processing requirements) with traditional ones. Finally, we have to benchmark our method versus traditional ones on some accepted data corpus.

At any rate, archival storage/retrieval may benefit from optimized compression even at greater encoding expenses in view of the fact that decompression times for MLDC are comparable to traditional methods'.

References:

[1] Gailly, Jean-loup, "Data Compression FAQs", ftp://rtfm.mit.edu/pub/usenet/news.answers/compression-faq/

[2] Zhao, Ben, "Algorithms in the "Real World" , http://www-2.cs.cmu.edu/afs/cs/project/pscico-guyb/realworld/www/index.html#notes

[3] Nelson, Mark, http://dogma.net/markn/articles/bwt/bwt.htm

[4] Hamming, R.W., Coding and Information Theory, Prentice-Hall, 1980, p. 80-89.

[5] Mahoney, Matthew, "Fast text compression with neural networks", Proc. FLAIRS, Orlando, 2000. cs.fit.edu/~mmahoney/compression/nn_paper.html

[6] Kuri, A., Galaviz, J., Pattern-Based Data Compression, *III Mexican International Congress on Artificial Intelligence*, to be published.

[7] Kuri, A., Herrera, O., Metrics for Symbol Clustering from a Pseudoergodic Information Source, Proceedings of ENC03, Tlaxcala, IEEE Press, 2003.