Incompressibility and Lossless Data Compression: An Approach by Pattern Discovery

Incompresibilidad y compresión de datos sin pérdidas: Un acercamiento con descubrimiento de patrones

Oscar Herrera Alcántara and Francisco Javier Zaragoza Martínez
Universidad Autónoma Metropolitana Unidad Azcapotzalco
Departamento de Sistemas
Av. San Pablo No. 180, Col. Reynosa Tamaulipas
Del. Azcapotzalco, 02200, Mexico City, Mexico
Tel. 53 18 95 32, Fax 53 94 45 34
oha@correo.azc.uam.mx, franz@correo.azc.uam.mx

Article received on July 14, 2008; accepted on April 03, 2009

Abstract
We present a novel method for lossless data compression that aims to get a different performance to those proposed in the last decades to tackle the underlying volume of data of the Information and Multimedia Ages. These latter methods are called entropic or classic because they are based on the Classic Information Theory of Claude E. Shannon and include Huffman [8], Arithmetic [14], Lempel-Ziv [15], Burrows Wheeler (BWT) [4], Move To Front (MTF) [3] and Prediction by Partial Matching (PPM) [5] techniques. We review the Incompressibility Theorem and its relation with classic methods and our method based on discovering symbol patterns called metasymbols. Experimental results allow us to propose metasymbolic compression as a tool for multimedia compression, sequence analysis and unsupervised clustering.
Keywords: Incompressibility, Data Compression, Information Theory, Pattern Discovery, Clustering.

Resumen
Presentamos un método novedoso para compresión de datos sin pérdidas que tiene por objetivo principal lograr un desempeño distinto a los propuestos en las últimas décadas para tratar con los volúmenes de datos propios de la Era de la Información y la Era Multimedia. Esos métodos llamados entrópicos o clásicos están basados en la Teoría de la Información Clásica de Claude E. Shannon e incluye los métodos de codificación de Huffman [8], Aritmético [14], Lempel-Ziv [15], Burrows Wheeler (BWT) [4], Move To Front (MTF) [3] y Prediction by Partial Matching (PPM) [5]. Revisamos el Teorema de Incompresibilidad y su relación con los métodos clásicos y con nuestro compresor basado en el descubrimiento de patrones llamados metasímbolos. Los resultados experimentales nos permiten proponer la compresión metasimbólica como una herramienta de compresión de archivos multimedios, útil en el análisis y el agrupamiento no supervisado de secuencias.
Palabras clave: Incompresibilidad, Compresión de Datos, Teoría de la Información, Descubrimiento de Patrones, Agrupamiento.

1 Introduction

In the last decades several methods have been proposed for lossless data compression such as: Huffman [8], Arithmetic [14], Lempel-Ziv (LZ) [15], Burrows Wheeler (BWT) [4], Move To Front (MTF) [3] and Prediction by Partial Matching (PPM) [5]. These methods are based on the Classic Information Theory of Claude E. Shannon where entropy is the measure of average information assigned to each symbol s_i in a sequence with an associated symbol distribution $P_{i=1}^{N} = \{p_i\}$, where p_i is the symbol probability for infinite sequences or symbol probability estimator for finite sequences. Classic methods have a good performance with plain text, computer generated graphics, and other files that: *i)* involve a reduced number of ASCII symbols, *ii)* their symbols follow a non-uniform distribution, *iii)* their symbols follow predefined grammatical rules, or *iv)* have local correlation between neighbor symbols. As classic methods have similar performance for this kind of files they are often compared with corpus such as Calgary [17] and Canterbury [13]. However, those corpora are not representative of our current needs: Internet traffic and storage systems involve other kind of files such as audio, image, video, biological sequences, climatic, and astronomic data that cannot be successfully compressed with classic methods [11].

In Section 2 we review the Incompressibility Theorem (IT) which states the non-existence of any method that compresses all sequences and we comment some consequences on lossless data compression, one of them is that all methods converge to the same performance over a large set of files. As we typically do not compress a large set of files[1] we should have a variety of compressors that represent alternatives to face the challenge of compressing different kinds of them, therefore it is more attractive for our research to create a compressor that aims for a different performance to classic methods, that focuses on files with patterns with no local correlation between neighbor symbols, that do not follow predefined grammatical rules, and that have quasi-uniform symbol distribution (high entropy). Since classic corpora do not deal with these files we prefer to implement an algorithm to create files with these characteristics and apply a statistical methodology (see Section 3.2) to measure the performance of our compressor.

If a comparison with classic corpora were made, a lower performance than classic is expected since our method does not focus on these benchmarks, however according to the IT we must have a better performance by focusing on subsets of files with non-trivial patterns. Moreover, our compressor can be applied after a classic method (post-compression) which does not occur with classic methods such as Huffman, LZ, and PPM methods because of their similarity (see Section 2.1). Other methods can be applied afterwards, as is the case of MTF [3] that essentially encodes a sequence by resorting to a last recent used memory (array) then it uses the index of the array corresponding to the current symbol of the sequence and moves the symbol to the front of the array (index equals zero). This method models the correlation of the immediate context and it applies successfully when a non-uniform distribution of indices is generated, so other entropic method can be applied. The BWT [4] method resorts to MTF by building sequence permutations that are expected to be well suited for MTF and then applies and entropic method.

Sections 2.1.1, 2.1.2, and 2.1.3 pretend to illustrate how Huffman, LZ, and PPM methods exploit the local redundancy and correlations in the symbol neighborhood. A similar analysis can be done for the other classic methods but it is not included here to save space.

In Section 3 we present a novel pattern-based lossless data compressor that aims to be different to the classic methods by encoding groups of non-necessarily neighbor symbols called *metasymbols* instead of encoding single symbols, subsequences, or modeling the local symbol correlation.

A simple example is presented by the sequence xyazbcxydze where we can see the metasymbol α=xy*z repeated twice and a residual metasymbol called *filler* β=**a*bc**d*e; the corresponding mapped sequence looks like αβα and the reconstruction is given by an ordered superposition: α=xy*z, αβ=xyazbc**d*e and finally αβα=xyazbcxyde.

As we pointed out, we do not use a predefined and human selected corpus to test our compressor but a statistical methodology (see Section 3.2) where a set of files with patterns is automatically generated and a probabilistic interval confidence of its performance is provided. Note that this approach does not mean that the method is not general for lossless compression, it can be applied to any finite sequence (file) what is assumed to be generated by a non-ergodic information source. If the sequence has metasymbols, it would be compressed but there is no warranty for this as with the rest of the compressors according to the IT.

In Sections 3.2 and 4 we present experimental results and comparisons with classic methods; the first goal is to show (statistically) that our method has different and better performance than classic methods for files with patterns.

Finally, in Section 5 we present conclusions and comment about applications and future work.

[1] Given a file that stores a sequence of n symbols, a large subset means $\sim 2^n$ different files.

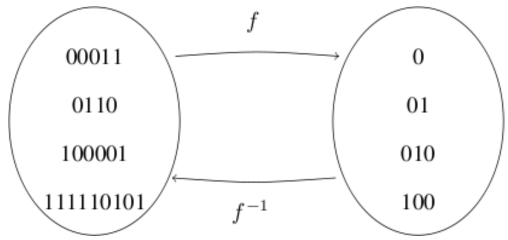

Fig. 1. Mapping function for compression *f* and decompression f^{-1}

2 Incompressibility and compression

An information source generates infinite sequences $s_{i1}s_{i2}s_{i3}...$ with symbols s_i taken from a finite alphabet $\Sigma = \{s_1, s_2, ..., s_N\}$ with an associated probability distribution $P_{i=1}^{N} = \{p_i\}$. In this work we will use string, message, or indexed symbol array to refer to a finite symbol sequence where each probability is estimated with its relative frequency.

Let A_n be any sequence of n bits, $n \geq 1$, and let F_n be the set of all 2^n different sequences A_n. Assume an original sequence A_n is mapped to other A_i that uses i bits. If $i < n$ we say A_n is compressed to A_i, if $i = n$ then we say that there is no compression and if $i > n$ we say A_n was expanded to A_i. The compression rate r (smaller is better) is defined as:

$$r = \frac{i}{n}. \tag{1}$$

A lossless method achieves compression with a mapping function *f* that must be injective in order to warrant the decompression through the inverse function f^{-1} (see Figure 1).

Classic methods focus on files with correlation between neighbor symbols and include plain text, computer generated images and some other files with minimum entropy. The entropy is defined as

$$H(S) = \sum_{i=1}^{N} - p_i I(s_i). \tag{2}$$

where $I(S_i) = \log_2(p_i)$ is the associated information for the i-th symbol defined by Shannon; note that this definition assumes that symbols are independent, consequently

$$I(s_i s_{i+1}) = I(s_i) + I(s_{i+1}) \tag{3}$$

Huffman's compression method achieves compression by building a binary tree and assigning Huffman codes h_i for each s_i of k bits (in the example $k = \lceil \log_2 5 \rceil = 3$), the mapping assigns short codes with $|h_i| < k$ to the most repeated symbols with frequency f_i and large codes for those with fewer repetitions, i.e. $|h_i| > k$. (see Figure 2)

The corresponding h_i is assigned by concatenating a "0" for each left movement and a "1" for each right movement in the tree. Figure 2 shows the Huffman codes for the string "aaaaaaaabbbbccde" and the mapping $a \to h_a = 0$, $b \to h_b = 10$, $c \to h_c = 110$, $d \to h_d = 1110$ and $e \to h_e = 1111$.

Huffman's mapping is optimal in the sense that it minimizes the average length for each symbol given by

$$\overline{l} = \frac{\sum_{i=1}^{N} f_i h_i}{\sum_{i=1}^{N} f_i} \tag{4}$$

which means that there is no other coding that produces a lower average length assuming that symbols are independent.

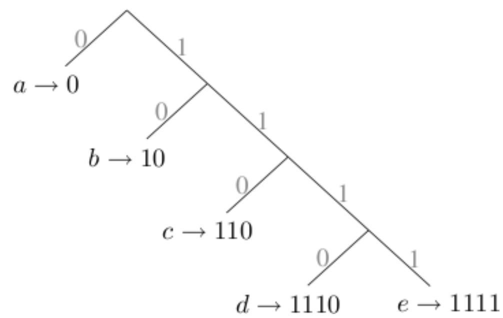

Fig. 2. Huffman Coding and Huffman Tree

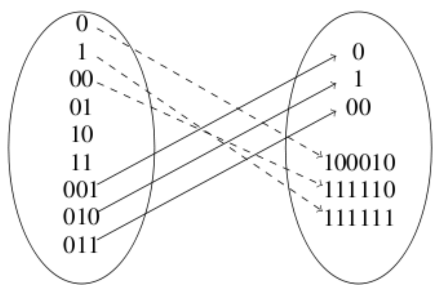

Fig. 3. Mapping function for expansion (dashed arrows) and compression (non-dashed arrows)

The symbol independence assumption restricts the performance of Huffman's technique because symbols are typically correlated and have conditional probabilities. These conditions are exploited by the PPM technique where the mapping considers conditional probabilities for the previous symbols called *context*. As the sequence has conditional probabilities between s_j and the context $s_{j-1}s_{j-2}s_{j-3}$ ..., PPM produces better results than the other classic methods and is considered the state of the art in lossless data compression. We remark that this method applies successfully to sequences with the already mentioned properties of using reduced number of ASCII symbols, follow a non-uniform distribution, follow predefined grammatical rules, or have local correlation between neighbor symbols. But this is not the case of multimedia, astronomical, and biological sequences, so we propose an alternative lossless compressor based on the identification of patterns with non-contiguous context.

At this point we consider important to recall the Incompressibility Theorem: For all *n*, there exists an incompressible string of length *n*. Proof: There are 2^n strings of length *n* and fewer than 2^n descriptions that are shorter than *n*. For $F_1 = \{0,1\}$ it is obvious that each file is incompressible because both of them (0 and 1) have the minimum one bit length and *f* can interchange $0 \rightarrow 1$ and $1 \rightarrow 0$ or keep them unchanged. Now consider the restriction of not to expand any file, the elements of $F_2 = \{00,01,10,11\}$ cannot be compressed because the only way to do this is mapping F_2 to F_1 but they were already used; as the same occurs for $n \geq 2$ we have shown that there is no method that promises to compress at least once and never to expand. The only way to achieve compression is to allow some sequences to be expanded, so elements in F_n can be assigned to elements of F_j for $j = 1,2,...,n-1$ (see Figure 3). Now let expand the families $F_1, F_2, ..., F_{n-1}$ in order to compress F_n, the average compression rate for $2^n - 2$ compressed sequences of F_n is

$$\bar{r} = \frac{\sum_{i=1}^{n-1} 2^i \frac{i}{n}}{2^n - 2} \tag{5}$$

which can be simplified to

$$\bar{r} = \frac{2^n(n-2)+2}{(2^n-2)n} \tag{6}$$

for *n>>2*, $\bar{r} \to 1$ meaning that if we compress $2^n - 2$ different files of F_n the average compression will always tend to one in spite of the *f* we use. Note that in this approach some files of F_n will be mapped to F_1 and achieve the lowest compression rate but other files will be mapped to families which provide undesirable high compression rates.

The success of a compression method relies in the mapping of one sequence to another with minimum length.

But what is the best method? Does this method exist? What about random sequences? If a given method fails then we should try another (different enough) method pretending to fit the data model but this is typically unknown; at this point we pretend to provide an alternative to classic methods.

In the previous discussion we use a unique injective mapping function and we are restricted to assign a compressed sequence for each original sequence, now we explore the possibility of assigning the same compressed sequence to different original sequences through different mapping functions, for this purpose we use *m* bits to identify each function and proceed to compress $2^n - 2$ sequences of *n* bits by assigning 2^m times the same A_i starting with those of minimum length A_1 and ending with A_z . We require that

$$2^n - 2 = \sum_{i=1}^{z} 2^{m+i} \tag{7}$$

Then

$$z = \log_2\left(\frac{2^n-2}{2^m} + 2\right) - 1 ,\ z \geq 1 ,\ m \geq 0 \tag{8}$$

so the average compression rate for $2^n - 2$ sequences is

$$\bar{r} = \frac{\sum_{i=1}^{z} \frac{i+m}{n} 2^{i+m}}{(2^n-2)} \tag{9}$$

The minimum value for $\bar{r}=1$ in (9) is achieved by *m*=0 which suggests to use only one function because unfortunately the use of bits to identify the mapping functions increases the effective compression rate.

Because all compressor functions have the same average compression rate over a large set of files (~2^n) and typically we do not compress all of them, the mapping function must be chosen to fit the data model in order to achieve low compression rates, for example *n*=1024 bits produce $2^{1024} = 16^{256}$ different files (more than a googol) so we will not compress all of them but a subset. Classic methods focus on the same subset of files where high correlation between neighbor symbols and non-uniform distribution are accomplished; in contrast we propose a different model based on the identification of repeated patterns called *metasymbols* which group non-necessarily adjacent symbols. The compressed sequence includes: *i)* the pattern description and *ii)* the sequence description considering these patterns; with both of them it must be possible to express the sequence in the most compact way.

Of course, we assume that sequences have patterns but what if there are no patterns? We say that "they are random" or "incompressible".

2.1 Practical considerations

In the last decades, several implementations of classic compressors have been developed and typically they have been modifications of older methods. Some of them are *gzip* that is a combination of LZ77 and Huffman coding; *compress* that is a UNIX compression program based on LZ techniques; *bzip-bzip2* that uses BWT and Huffman coding, and so many more program implementations.

In practice, the length of the compressor program is assumed to be zero, that is, we transmit a compressed sequence and assume that the receptor has a copy of the decompressor program. The mime type associated with the file extension (.gz for gzip, etc.) represents the identifier of the applied method (mapping function) that was chosen. The compressor program defines a format for the bits in the compressed file that allows the decompressor program to recover the original file.

Table 1. Huffman Coding and Symbol Length Selection

n	Huffman Coding	Compressed sequence	Length
1	0	**01** 0000000000000000	18
2	0	**011** 00000000	11
4	0	**01111** 0000	9
8	0	**011111111** 00	11
16	0	**01111111111111111** 0	18

Consider a compression mapping that represents the sequence 11 as 0 and the sequence 001 as 00, if we transmit 000 the decoder will not be able to decompress it because it can be interpreted as 00,0; 0,00 or 0,0,0 consequently we need to include more bits for the "header" or "format" to indicate the start and end of a compressed code. The format is not unique and in the last example we can use a bit set so 01001 indicates start-0-end start-00-end. Note that the inclusion of header bits will increase the effective compression rate and it should be considered in the encoding process, which does not occur for example in the Huffman encoding.

***2.1.1 Huffman method**:* Now we review a mapping function for the Huffman encoding, as we will see the optimal encoding will be assigned by neglecting the header size and encoding individual symbols without considering the context.

For a given alphabet and a probability distribution, Huffman's coding builds a prefix code where each mapped h_i is not a prefix of any other, for example $h_1 = 1$, $h_2 = 01$, $h_3 = 001, \ldots$ have this property and allow to transmit sequences without ambiguity. As we pointed out, for a given *n* we need to expand some sequences in order to achieve compression and Huffman's technique is not the exception as is shown in Figure 2, where a fixed length encoding requires 3 bits for each symbol whereas Huffman's encoding uses 1, 2, 3 or 4 bits.

In Shannon's Information theory the symbol is an entity with an associated probability but does not define the length of each symbol, the base of logarithms indicates the measure of information, $I(s) = \log_2(p_i)$ refers to *Bits*, $I(s) = \log_e(p_i)$ to *Nats*, and $I(s) = \log_{10}(p_i)$ refers to *Hartleys.*

For example, in the sequence 00000000011011 we can choose two bits as symbols so the file looks like 00 00 00 00 01 10 11 and, although the Huffman symbol mapping may not be unique, the average length will be the same for a fixed size or number of bits for each symbol.

Huffman's technique assigns optimal codes in the sense that there is no better coding that reduces the length of the mapped sequence assuming an ideal header (zero bits to describe the mapping itself) and symbol independence.

The selection for the number of bits *n* for each symbol A_n is selected by the user. Better results can be achieved by exploring different possibilities for *n*. Consider the next simple example *B*=1111111111111111, Table I shows different values for *n*, the corresponding Huffman coding and the compressed sequence with the corresponding length.

From Table I it is clear that *n*=4 is the best choice and the encoded message *B* is given by 0**1111** 0000 where the header is formed by the associated Huffman code "0" and 1111 means the selected symbol, therefore 0000 means 1111111111111111.

Huffman encoding typically predefines a symbol size (8 bits) but in fact it could offer better compression results if we try other symbol sizes, and even more if the encoding considers the header size as we discuss in Section 3.

Of course, it will take a lot of additional time but that leads to the dilemma, how much do we invest in time, how much do we compress.

Table 2. Mapping function for lz technique (x means don't care)

Offset	Matching substring length	Next symbol	Current position	Partially encoded sequence
-	-	-	0	-
000	000	00	1	00
001	001	00	3	00 00 00
001	001	01	5	00 00 00 00 01
000	000	10	6	00 00 00 00 01 10
000	000	11	7	00 00 00 00 01 10 11
111	111	00x	15	00 00 00 00 01 10 11 00 00 00 00 01 10 11

2.1.2 Lempel Ziv-Dictionary Method: Now we review dictionary techniques, as we can see they are adequate to compress text files that have a reduced set of ASCII symbols and a non-uniform symbol distribution. But this is not the case for multimedia and other files characterized by a quasi-uniform distribution.

LZ techniques use adaptive mapping functions that change as the sequence is followed from left to right and the symbols are indexed from positions 0 to $L-1$. Consider an example with *n*=2 for the symbol length and the sequence "00 00 00 00 01 10 11 00 00 00 00 01 10 11", the compression is achieved by mapping to 3-tuples *(Offset, MatchingSubstringLength, NextSymbol)* where the *offset* is the number of symbols of the immediate context that can be reviewed, the *MatchingSubstringLength* represents the length of the largest substring that matches the context with the symbols (window) to the right of the current symbol and the third element *NextSymbol* encodes the symbol that breaks the matching in front of the current position. Suppose we use 3 bits for the *offset* that determines the maximum length for the matching substring, we start to map the first symbol "00" as (0,0,00) because we have no previous context and there is no matching substring, now the current position is 1; the next symbols "00 00" are mapped as (1,1,00) because the previous context matches "00" and encode the next symbol "00", now the current position is 3; in Table II we describe the full encoding process for this sequence.

As we can see in Table II, the same subsequence has different codings depending on the immediate context and the compression rate depends strongly on the possible high context correlation between neighbor symbols. In practice the optimal number of bits for the offset is unknown and it defines the length of the context window, the larger the offset length the larger the context window and the probability of matching increases but the effective compression rate tends to increase; if the context window is reduced the matching probability will decrease.

2.1.3 Prediction by Partial Matching (PPM) Method: PPM uses an adaptive and sophisticated encoding where the main idea is to estimate the conditional probabilities for the current symbol to be mapped C_0 considering contexts of order 1, 2, 3 . . . , *k* expressed as $C_k C_{k-1} C_{k-2} C_{k-3} ... C_1 C_0$, as the sequence is read from left to right the table is updated for each symbol and order context. The length of the encoded sequence for C_0 decreases when the conditional probability increases and the *k*-th order context is preferred to use, meaning a high correlation between neighbor symbols, however it fails when $C_{k-1} C_{k-2} C_{k-3} ... C_1 C_0$ is not in the table and the previous $k-1$ context is used. For this purpose an escape sequence *ESC* must be included and it requires $\log_2(1/p_{ESC})$ bits where p_{ESC} is the probability estimator for the escape sequence calculated for the corresponding order context. If the ($k-1$) order context fails it uses the lower context and the worst case is to consider the 0-order context requiring ($k-1$) escape sequences with no conditional probabilities (independent symbols) as Huffman's coding does, so C_0 is encoded with $\log_2(Q+1)$ bits, where Q is the number of different subsequences in the sequence for a given order context. Because of the penalization on escape sequences, in practice low order contexts are used (typically 1, 2, 3, 4 or 5).

When there exists high correlation between low order contexts, there are enough repeated subsequences in the conditional probabilities table and it follows a non-uniform distribution, PPM achieves better compression rates than the other classic methods and this explains why it is considered the state of the art in lossless data compression.

	0	1	2	3	4	5	6	7
0	A	a	D	b	E	c	F	d
8	e	B	f	g	C	h	i	j
16	k	D	l	E	m	F	n	ñ
24	o	A	p	A	q	r	s	t
32	u	v	B	w	B	C	x	C
40	D	y	E	D	F	E	z	F

Fig. 4. A bi-dimensional representation of a message

We remark the condition of correlation between low order context that defines the kind of files that classic methods are adequate to compress, but this is not satisfied by multimedia, astronomical, biological, and other sequences.

An alternative to compress multimedia sequences has been the lossy compressors such as MP3 [7] and JPEG [12]; MP3 creates a psychoacoustic model of the human ear and eliminates non-relevant information, after this a classic compressor is applied; in the case of JPEG a visual model of the human vision is created to eliminate the non-appreciable visual information of the original sequence and then a classic compressor is applied. In both cases there is a strong dependence on the entropy of the intermediate and modified sequence that depends on the sequence itself and the quality factor of the lossy step. Because PPM is the state of the art we may think that it could be widely used but this is not true because of its relative high computational cost derived from the number of required calculations to update the table of conditional probabilities when it is compared with other classic methods.

Classic methods are based on the principle of symmetry: the same time for compression and decompression. But the web, as a client-server system, is not symmetric. The servers are powerful and have shared resources for the clients, the clients are typically end user systems with low processing capacity and consequently we can compress a single multimedia file at the server side without time priority *(offline)* but focused to lower compression ratios. Fractal image compression [1][2][16] follows this approach and inverts a lot of time to compress an image, the decompression time is lower than the compression time and is used by multiple clients of web and CD/DVD encyclopedias.

In the next section we present a non-symmetric method for lossless data compression that is based on the identification of symbol patterns.

3 Metasymbolic lossless data compression

Metasymbolic lossless data compression identifies symbol patterns called metasymbols that groups non-necessarily neighbor symbols in a sequence *S* and have the next properties:

1) Length $|M_i|$. The number of symbols that the *i*-th metasymbol includes.

2) Frequency $|f_i|$. The number of repetitions of the *i*-th metasymbol in *S*.

3) Gaps $\{g_i\}$. The spaces between the symbols of a metasymbol.

4) Offsets $\{o_i\}$. The indices of the first symbol for all the instances of a metasymbol.

These properties define the metasymbol structure and they require bits to be encoded, so our goal is to minimize the number of bits required to describe the metasymbols plus the number of bits of the encoded sequence using these metasymbols.

Without loss of generality let *n=8* and consider a hypothetical sequence *S* with extended ASCII symbols taken from $F_8 = \{A_8\}$, $S =$ "$A_0a_1D_2b_3E_4c_5F_6d_7e_8B_9f_{10}g_{11}C_{12}h_{13}i_{14}j_{15}k_{16}D_{17}l_{18}E_{19}m_{20}F_{21}n_{22}ñ_{23}o_{24}A_{25}p_{26}A_{27}q_{28}r_{29}s_{30}t_{31}u_{32}v_{33}B_{34}$ $w_{35}B_{36}C_{37}x_{38}C_{39}D_{40}y_{41}E_{42}D_{43}F_{44}E_{45}z_{46}F_{47}$" then $|S| = 48$. To ease visualization we include a two-dimensional matrix for *S* shown in Figure 4.

We can appreciate a first metasymbol ("ABC") at positions 0, 25 and 27 with gaps of size 9 and 3 respectively.

In what follows we will use the representation $M_1 = A_9B_3C : \{0,25,2\}$ which means: There is an "A" at position 0, a "B" 9 positions away from "A" and a "C" 3 positions away from "B"; the metasymbol is repeated at position 25 and 2 away from 25 (25+2=27). A second metasymbol is $M_2 = D_2E_2F : \{2,15,23,3\}$. The symbols that do not form repeated patterns are lumped in a pseudo-metasymbol that we call the filler. One possible efficient encoding depends on the fact that the filler is easily determined as the remaining space once the other metasymbols are known. Hence

$$M_{filler}=\text{abcdefghijklmnñopqrstuvwxyz}$$

Metasymbolic search consists not only in the identification of repeated patterns but also in the selection of those which imply the minimum number of bits to reexpress *M*. In the last example we assume, for simplicity, that metasymbols M_1, M_2 and M_{filler} satisfy this criterion but this problem is NP-hard.

We describe the proposed encoding scheme:

Let *n* be the number of bits of each symbol (in our example *n*=8). Let μ be the number of metasymbols (in our example $\mu = 3$). We use nibble[2] 'a' to indicate the value of *n* in binary[3]. In the example $a = 1000_2$. We use nibble 'b' to indicate the number of bits required to code the value of μ in binary. In the example $b = 0010_2$ and μ is coded as $\mu = 11_2$.

Likewise, the maximum gap for M_1 is 9 and we need $g_1 = \log_2\lceil 9+1 \rceil = 3$ bits to encode this value in binary.

The other gaps for M_1 will be encoded with g_1 bits. We use nibble 'c_1' to indicate the number of bits required by the g_1 value. In the example $c_1 = 0011_2$. The maximum position for M_1 is 25 and we need $p_1 = \log_2\lceil 25+1 \rceil = 5$ bits to encode this value in binary. The positions of the other instances of M_1 will be encoded with p_1 bits. We use nibble 'd_1' to indicate the number of bits required by the p_1 value. In the example $c_1 = 0101_2$ $|M_1| = 3$ and we need *3n bits* to encode its content. The maximum gap for M_2 is 2, and we need $g_2 = \log_2\lceil 2+1 \rceil = 3$ bits to encode this value in binary. The other gaps for M_2 will be encoded with g_2 bits. We use nibble 'c_2' to indicate the number of bits required by the g_2 value. In the example $c_2 = 0011_2$. The maximum position for M_2 is 23 and we need $p_2 = \log_2\lceil 23+1 \rceil = 5$ to encode this value in binary. The positions of the other instances of M_2 will be encoded with p_2 bits. We use nibble 'd_1' to indicate the number of bits required by the p_2 value. In the example $c_2 = 0101_2$. $|M_2| = 3$ and we need 3*n* bits to encode this content. Also $|M_{filler}| = 30$ and is simply an enumeration of values, therefore we need 30*n bits* to encode it. The number of bits for *S* expressed as metasymbols is given by

---

[2] By "nibble" we mean 4 consecutive bits.

[3] We simply write "binary" when we refer to weighted binary.

$$C = a + b + \mu + \sum_{i=1}^{\mu-1}(|M_i - 1|g_i + c_i + d_i + |M_i|n + f_i p_i) + n(|M| - \sum_{i=1}^{\mu-1}|M_i|f_i) \tag{10}$$

We must minimize *C* by selecting the best metasymbols.

3.1 Metasymbolic search

Metasymbol discovery represents an NP-hard problem [9] and the minimization of *C* has the inconvenient that it considers the bits required to encode the sequence whenever metasymbols are known, but to know the metasymbols we need to have minimized *C*, this leads to a cyclical dependence whose way out is described next. We propose an iterative method for metasymbol searching that neglects the first 3 terms *a, b* and μ of (10) since they represent a few bits (in the previous example *a* and *b* use four bits and μ uses two bits) and we proceed with the fourth term that represents the number of bits used by metasymbol descriptions and the bits of the encoded sequence with these metasymbols; the fifth term refers to the bits required by the filler. Therefore it is possible to define a discriminant d_i from the summation in the fourth term that corresponds to the number of bits required by all the $\mu - 1$ non-filler metasymbols, so the fourth term can be expressed as

$$\sum_{i=1}^{\mu-1}(|M_i - 1|g_i + c_i + d_i + |M_i|n + f_i p_i) = \sum_{i=1}^{\mu-1} d_i \tag{11}$$

where the number of bits required to encode the *i*-th metasymbol $(1 \le i \le \mu - 1)$ is

$$d_i = |M_i - 1|g_i + c_i + d_i + |M_i|n + f_i p_i \,. \tag{12}$$

Note that d_i just depends on the *i*-th metasymbol, consequently we can search metasymbols iteratively by comparing their d_i values and selecting the one with minimum value. Our method considers non-overlapped metasymbols to reduce the search space then once a symbol is assigned it can not be used by any other metasymbol but variants of this can be explored (see [9]).

Algorithm 1 called MSIM shows how to search metasymbols under this approach to compress a sequence S with elements $S[i]$ taken from an alphabet $\Sigma = \{s_1, s_2, ..., s_N\}$.

3.2 Experiments

A set of experiments was performed to:

1) Compare the performance of MSIM with classic compressors. In fact we expect to be different because we are not interested on creating yet another lower context correlation compressor.

2) Measure the efficiency of MSIM as a compressor.

3) Set a confidence level for the expected lower bound of compression achieved with MSIM for files with patterns and high entropy.

4) Measure the efficiency of MSIM as a tool to discover the metasymbols that allows us to reexpress a message in the most compact way.

5) Measure the efficiency of MSIM as a tool to discover metasymbols in spite of the content of the messages.

To this effect we use Algorithm 2 to build a set of messages that contain metasymbols. Note that Algorithm 2 simply determines the structure of the metasymbols but does not assign their contents; the contents are determined later in the process.

We define a compressor called REF that represents our theoretical limit of compression because it takes the information stored in a database provided by Algorithm 2 to determine the number of bits *C* of the corresponding compressed message that does not depend on the content of the metasymbols (see ec. (10)). So we can measure the efficiency of MSIM relative to REF, if MSIM reaches a performance similar to the hypothetical one of REF we have a statistical confidence level that MSIM is able to reach the "theoretical" limit of compression (goal 1) and it identifies the hidden metasymbols (goal 2). The experiments include a comparison of MSIM against other classic compressors LZ77, LZW, Arithmetic, and PPM running on the same machine and expecting to get a different performance to show that MSIM focuses on sequences with patterns (goal 3).

Algorithm 1 MSIM for metasymbolic compression.

INPUT(SEQUENCE S) OUTPUT(METASYMBOLS M)
$\mu \leftarrow 1$, $FreeSymbols \leftarrow \mu, di = 0$
Mark all symbols in S as free symbols
while $FreeSymbols > 0$ and $di < 1.0$ **do**
  $LEN = 1, \{s_1, s_2, ..., s_N\}$ is the set of different symbols in S
  for $j = 1$ to N **do**
    $f_{jLEN} \leftarrow$ Highest frequency of all the symbols located at g_{LEN} positions of $\sigma_{j0} = s_j$
    while $f_{jLEN} > 1$ **do**
      Identify the symbol σ_{jLEN} with highest frequency f_{jLEN} nearest to σ_{j0}, located at g_{LEN} positions. Mark all the instances of σ_{jLEN} as non-free symbols.
      LEN = LEN+1
      Evaluate the compression function C, $di \leftarrow C$
      Store the μ-th metasymbol $\{\sigma_{j0}\ \sigma_{j1}\sigma_{j2}...\sigma_{LEN}\}$ with minimal di
    end while
  end for
  Select the μ-th metasymbol $\{\sigma_0\ \sigma_1\ \sigma_2...\sigma_{LEN}\}$ with minimal di
  if $dis < 1.0$ **then**
    Mark symbols of S as $non\_free\_symbols$ for each symbol of the metasymbol instances.
    $FreeSymbols \leftarrow Freesymbols$ in S
    $\mu = \mu + 1$
  end if
end while
$Metasymbol_{filler} \leftarrow$ all free symbols in S
The number of metasymbols is μ = $\mu + 1$

The fourth goal reflects our interest on the search for structural patterns which are independent of the content of the message. Hence, we generate messages with the same metasymbolic structure but different contents. First, we store the properties of the sets of metasymbols (frequency, length, gaps, and offsets) then, for a given set of metasymbols, we explore five variations of the same message by filling the metasymbols with contents determined from experimental probabilistic distribution functions from a) Spanish text file, b) English text file, c) audio compressed file, d) image compressed file and e) uniformly distributed data.

These file variations were compressed with classic methods (PPM, LZW, and Arithmetic) and MSIM, so we get five compression ratios (SPA for Spanish, ENG for English, MP3 for Audio, JPG for image, and UNI for uniform) that represent samples of five unknown probability distributions (call them "A" distributions).

Because the REF compressor is fully based on the structure of the metasymbols and is impervious to its contents it was used to measure the performance of MSIM as a metasymbolic identification tool.

The statistical approach is used to estimate the worst case compression value achieved for each of the compressors and for each of the five "B" distributions, as well as to estimate the mean (μ) and the standard deviation (σ) values of "A" distributions of the compression rates for the messages with metasymbols. We will appeal to the Central Limit

Algorithm 2 Generator of files with metasymbols.

```
i ← 0
LABEL1:
while there are free positions in S and NOT exceeded the maximum number of iterations do
  i = i + 1
  Generate a random value for |M_i| in the interval [1,|S|]
  Propose |M_i|-1 gaps for the symbols of the i-th metasymbol
  Generate a random value for f_i in the interval [1, |S|]
  for j = 1 to f_i do
    Generate a position pos_i for a instance of the current metasymbol
    Try to put the instance of the metasymbol at pos_i whenever it does not overlap with another metasymbol.
    if the number of tries is exceeded then
      f_i ← number of successful outcomes
      go to LABEL1
    end if
  end for
end while
The remaining free positions in the sequence are the filler positions
i = i + 1
μ ← i is the final number of metasymbols
Store in a database the properties for the μ metasymbols.
```

Table 3. Samples by sextil for b-distributions

Compressor	Sequence	Sextil1	Sextil2	Sextil3	Sextil4	Sextil5	Sextil6
REF	ENG	13	10	12	16	23	10
	JPG	13	10	12	16	23	10
	MP3	13	10	12	16	23	10
	SPA	13	10	12	16	23	10
	UNI	13	10	12	16	23	10
MSIM	ENG	12	9	16	19	14	14
	JPG	10	12	11	20	18	13
	MP3	10	11	12	20	18	13
	SPA	12	8	18	18	14	14
	UNI	10	14	12	15	19	14
PPM	ENG	12	12	16	16	15	13
	JPG	11	20	18	14	9	12
	MP3	10	21	18	14	9	12
	SPA	13	9	16	19	14	13
	UNI	11	20	18	14	9	12

Theorem [6] by generating the corresponding distributions where each mean is the average of m=25 samples from "A" distributions (call them "B" distributions). If X represents an object from one distribution A for a given type of file (Spanish, for example) then

$$\overline{X}_i = \frac{\sum_{i=1}^{m} X_i}{m} \tag{13}$$

represents an object of the corresponding "B" distribution.

The mean $\mu_{\bar{x}}$ for any of the B distributions is given by:

$$\mu_{\bar{x}} = \frac{\sum_{i=1}^{Z} \overline{X}}{m} \tag{14}$$

where Z is the number of objects of the "B" distributions.

The standard deviation $\sigma_{\bar{x}}$ for "B" distribution is given by:

$$\sigma_{\bar{x}} = \sqrt{\frac{\sum_{i=1}^{Z} (\overline{X}_i - \mu_{\bar{x}})^2}{Z}} \tag{15}$$

We generated as many messages (objects of "A" distributions) as were necessary until the next two conditions were fulfilled:

1) For each "A" distribution we require that there are at least 16.6% of Z samples in at least one sextil for the associated distribution B (see Table III). This approach allows comparing it with a normal distribution where each sextil has $\frac{100}{16}\%$ of the total area, which is 1.

2) The mean values of both "A" and "B" distributions are similar enough and satisfy (16)

$$\frac{\mu - \sigma_{\bar{x}}}{\mu_{\bar{x}}} \geq 0.95. \tag{16}$$

In our experiments we needed 2,100 messages for each of the different distributions, that is, we analyzed a total of 10,500 messages.

Table 4. Values for μ and σ parameters for "a" distributions and different compressors

	Parameter	**ENG**	**JPG**	**MP3**	**SPA**	**UNI**	**AVG**
Compressor	-	-	-	-	-	-	-
REF	μ	0.52	0.52	0.52	0.52	0.52	0.52
	σ	0.19	0.19	0.19	0.19	0.19	0.19
MSIM	μ	0.63	0.6	0.6	0.63	0.6	0.62
	σ	0.14	0.15	0.15	0.14	0.15	0.13
PPM	μ	0.6	0.83	0.83	0.58	0.83	0.71
	σ	0.33	0.51	0.51	0.31	0.51	0.36
LZW	μ	0.75	0.95	0.95	0.74	0.95	0.86
	σ	0.4	0.68	0.73	0.38	0.68	0.52
ARIT	μ	0.65	0.8	0.8	0.64	0.8	0.73
	σ	0.25	0.22	0.22	0.24	0.22	0.24

Table 5. Values for the worst case compression rates

Compressor	**Parameter**	**ENG**	**JPG**	**MP3**	**SPA**	**UNI**	**AVG**
REF	X_{worst}	0.99	0.99	0.99	0.99	0.99	0.99
MSIM	X_{worst}	0.89	0.88	0.8	0.89	0.88	0.88
PPM	X_{worst}	2.04	4.35	4.35	1.85	4.35	2.94
LZW	X_{worst}	2.38	14.29	10.0	2.27	14.29	5
ARI	X_{worst}	1.33	1.27	1.27	1.27	1.27	1.28

Then we may estimate the values of the mean $\mu = \mu_{\bar{x}}$ and the standard deviation σ for the "A" distributions since:

$$\sigma = \sqrt{n\sigma_{\bar{x}}} \tag{17}$$

4 Results

As we see in Table IV, MSIM provides consistent values of compression (see μ and σ) for the five explored types of file and, as expected, regardless of the contents of the metasymbols.

Once we have estimated the values of the parameters for the five distributions A, we estimate the value of the compression for the worst case X_{worst} of the different compressors appealing to Chebyshev's Theorem: Suppose that X is a random variable (discrete or continuous) having mean $\mu_{\bar{x}}$ and variance σ^2, which are finite. Then if $k\sigma$ is any positive number then

$$P(\mu_x - k\sigma \le X \le \mu_x + k\sigma) \ge 1 - \frac{1}{k^2} \tag{18}$$

Assuming that the distributions A are symmetric we find, from (18) that

$$P(\mu_x - k\sigma \le X) \ge 1 - \frac{1}{2k^2} \tag{19}$$

and then it is possible to estimate the expected compression value X_{worst} for all distributions. It immediately follows that $P(\mu_x - k\sigma \le X) \ge 0.7 \to k = 1.3$. Table V shows the experimental results where it is possible to appreciate a similar performance between MSIM and REF which is not the case for classic compressors.

From Table V we know that in 70% of the cases MSIM achieves better performance than the other compressors and that its behavior is not a function of the contents of the messages.

Table 6. $\chi 2$ values for 35 b- distributions

	ENG	JPG	MP3	SPA	UNI
REF	0	0	0	0	0
MSIM	7.19	4.16	3.78	8.85	4.65
PPM	5.49	22.48	24.96	6.42	22.48

Finally, we analyze the likelihood between the distributions B of REF and the distributions B of MSIM. Remember that REF does not depend on the metasymbol contents so its five B distributions are identical. In Table V and Table IV we can appreciate how MSIM distributions display similar behavior as REF distributions. A direct conclusion is that MSIM also does not depend on the contents. A goodness of fit test was used to test this hypothesis. We can see, from Table VI, that the χ^2 values for MSIM are small enough. In contrast see, for example, the cases of PPM (χ^2 >22) for JPG, MP3 and UNI. They reveal strong dependency between their performance and the content of the messages. Other compressors (LZW, LZ77, Arithmetic, Huffman, etc.) behave similarly (data is available under request).

5 Conclusions

Based on the Incompressibility Theorem is possible to show the non-existence of any method that compresses all files and that the average compression ratio for a large set of files $\sim 2^n$ is the same ($\bar{r} \to 1$) for all the lossless compression methods, then we should have different methods to face the challenge of compressing different kinds of files (subsets). Classic compressors focus on compressing files with local context correlation between neighbor symbols what may not be representative of the files we currently use so we present MSIM, a novel method for lossless data compression based on the identification of patterns called metasymbols that allows identifying high context correlation between symbols in sequences.

MSIM implies the identification of metasymbols and their search is guided by the minimization of a compression function that joins the bits required to encode the metasymbols as well as the encoded message. Since metasymbol's structure is not predefined (frequency, length, gaps and offsets) and symbols are grouped in metasymbols by minimizing the compression function we call this process unsupervised clustering. Correlated symbols belong to the same metasymbol (class) and non-correlated symbols belong to different metasymbols.

For our experiments we propose not to use a human selected benchmark but an algorithmically generated set of files with patterns and a statistical methodology where a goodness of fit test to show that MSIM reaches similar performance to REF, the hypothetical compressor used as reference (goal 1). Statistical results show that MSIM is able to discover the metasymbols embedded in a large enough set of sequences (goal 2), that MSIM has unsimilar performance to classic compressors (goal 3) and that if there are patterns in the sequences we have statistical evidence that MSIM will discover them (goal 4). The last goal is particularly interesting because it allows us to explore sequences where it is not known whether there are patterns.

Considering the above, future work includes multimedia, biological and astronomical sequences analysis as well as to develop parallel algorithms for metasymbol's search and to implement client-server and distributed applications for data mining.

References

1. **Barnsley, M. (1993)** "Fractals Everywhere", Morgan Kaufmann Pub; 2nd. Sub edition.
2. **Barnsley, M.** and **Hurd, L. (1992)** "Fractal Image Compression", AK Petters, Ltd., Wellesley, Ma.
3. **Bentley, J., Sleator, et. al. (1986)** "A locally adaptive data compression algorithm", Communications of the ACM, Vol. 29, No. 4, pp 320-330.
4. **Burrows, M.** and **Wheeler, D. (1994)** "A block-sorting lossless data compression algorithm", Digital Syst. Res. Ctr., Palo Alto, CA, Tech. Rep. SRC 124.
5. **Cleary, J.** and **Witten, I. (1984)** "Data compression using adaptive coding and partial string matching", IEEE Transactions on Communications, Vol. 32, No. 4, pp 396-402.
6. **Feller, W. (1968)** "An Introduction to Probability Theory and Its Applications", John Wiley, 2nd. Edition, pp 233–234.
7. **Hacker, S. (2000)** "MP3: The definitive guide, Sebastopol Calif., O'Reilly.
8. **Huffman, D. (1952)** "A method for the construction of minimum-redundancy codes", Proc. Inst. Radio Eng. 40, 9 (.), pp 1098–1101.
9. **Kuri, A.** and **Galaviz, J. (2004)** "Pattern-based data compression", Lecture Notes in Artificial Intelligence LNAI 2972, pp 1–10.
10. **Nelson, M.** and **Gailly, J. (1995)** The "Data Compression Book", Second Edition, MT Books Redwood City, CA.
11. **Nevill-Manning, C. G.** and **Witten, I. (1999)** "Protein is incompressible", Data Compression Conference (DCC '99). p. 257
12. **Pennebaker, W.B.** and **Mitchell, J. (1993)** "JPEG: Still Image Data Compression Standard", ITP Inc.
13. **Ross, A.** and **Tim, B. (1997)** "A Corpus for the Evaluation of Lossless Compression Algorithms", Proceedings of the Conference on Data Compression, IEEE Computer Society, Washington, DC, USA.
14. **Witten, I., Neal, R.** and **Cleary J. (1987)** "Arithmetic coding for data compression", Communications of the ACM 30(6) pp 520–540.
15. **Ziv, J.,** and **Lempel, A. (1977)** "A Universal Algorithm for Sequential Data Compression", IEEE Trans. on Inf. Theory IT-23, v3, pp 337–343.

16. **Microsoft Encarta, fractal image compression (2008)** *http://encarta.msn.com/encyclopedia_761568021/fractal.html* (accessed July 14, 2008).
17. **Calgary Corpus (2008)** *http://links.uwaterloo.ca/calgary.corpus.html* (accessed July 14, 2008).

Oscar Herrera Alcántara. *He received a Ph. D. and M. S. degree in Computer Science from CIC at Instituto Politécnico Nacional in Mexico. He received, in 2001 from the President of Mexico, the Lázaro Cardenas award and in 1997 the Merit Medal award from the Universidad Autónoma Metropolitana Azcapotzalco as electronics engineer, where he is professor and researcher since 2007. His research focuses on lossless data compression, clustering, optimization, heuristics, genetic algorithms and neural networks applications.*

Francisco Javier Zaragoza Martínez. *He received a Ph. D. in Combinatorics and Optimization from the University of Waterloo, Canada and a M. S. degree from CINVESTAV at Instituto Politécnico Nacional, Mexico. He obtained his bachelor degree in Electronics Engineering from the Universidad Autónoma Metropolitana Azcapotzalco, where he obtained the Merit Medal in 1994. He is currently a full professor at the same institution, where he also is the Undergraduate Coordinator of Computer Engineering. He is a member of the National System of Researchers (SNI 1) and has the PROMEP profile. His research focuses on algorithms for discrete optimization problems.*