International Journal of Signal Processing, Image Processing and Pattern Recognition
Vol.9, No.3 (2016), pp.199-218
http://dx.doi.org/10.14257/ijsip.2016.9.3.18

Leveling: an Efficient VLC for Lossless Data Compression

Javier Joglar Alcubilla

Avionics Department, Barajas Institute, Avda. América 119, 28042, Madrid, Spain
jjoglar@educa.madrid.org

Abstract

Many of the standard compression methods are based, at their lowest level, in coding digital words of variable length or VLC, Huffman type, designed in 1952 or in any of its versions, as canonical. This article presents an VLC with greater efficiency than Huffman encoding, named as "Leveling", which uses two variants, "Leveled Reordering" for low redundancy, e.g. for text and, "Segmented Leveling" for middle and high redundancy, for image processing. Leveling, developed by Javier Joglar in 1995, uses the concepts of "meaning" and "ordering" of the VLC codes generated, to get optimum performance in terms of "compression ratio", higher than any other non-adaptive VLC.

Keywords: *Entropy coding, Huffman encoding, Leveling, Lossless data compression, Meaning, VLC.*

1. Introduction

Suppose a monochromatic scanned image, which in sampling produces *P*Q* pixels (width*height) or symbols and, quantized with an accuracy of *J*-bits. For example, with *J*=8 bits, the brightness levels between 0 and 255 produce a total of *P*Q*J* bits.

Whereas the luminance quantized values are not all equally probable, it is possible a reduction in the number of bits needed to specify the image if, instead of assigning words of constant length *J* for each 2^J possible luminance levels, we assign variable length words.

The way is to associate short length words with those luminance levels that have the highest probability, while the longer length words are assigned to those other levels with lower probability of occurrence: *Variable Length Coding* or *VLC* [4].

If any luminance level ***b*** quantified, with occurrence probability ***P(b)***, is assigned to a binary word of length ***L(b)*** bits, we will have an average word length for the image considered:

$$\bar{L} = \sum_b L(b)P(b) \text{ bits/symbol} \tag{1}$$

We are interested in defining codes, which $\bar{L}$ is as small as possible for. The information theory provides a floor for the average length $\bar{L}$, called *entropy* of the random variable ***B*** (associated with the values *b*), and described as:

$$H(B) = -\sum_b P(b)\log_2 P(b) \text{ bits/symbol, with } H(B) \le \bar{L} \tag{2}$$

Entropy *H(B)* is a measure of the amount of information carried by the random variable *B*. In addition, information redundancy $\bar{R}$ is the difference between the average word length $\bar{L}$ and entropy *H(B)* [6], i.e.:

ISSN: 2005-4254 IJSIP

$$\overline{R} = \overline{L} - H \tag{3}$$

When the quantized luminance is coded in *VLC* and the result of the words coded is concatenated, creating a linear bit stream for storage or transmission, correct decoding requires that all combinations of concatenating code words are uniquely decodable. The necessary and sufficient condition for this is that the code fulfils the "*prefix rule*". According to this, no code word may be the prefix of another different code word.

The non-adaptive *VLC* system most commonly used is *Huffman encoding* or any of its versions. Huffman method is an optimal encoding, that is, with minimum redundancy, between 0 and 1 [8], [6]. Although Huffman encoding minimizes the bits/pixel average, satisfying the prefix rule, leads to certain inconsistency in codes not synchronized with sorting by occurrence of the variable *B* considered, besides lacking of "meaning" in its value, for example, in decimal base. For this reason, discussed in detail below, and although this is one of the few existing lossless entropic encoding, i.e., close to the limit value given by the entropy, Huffman can be improved still more.

Next we include a process of lossless encoding *VLC* type, called "*leveled encoding or leveling*", with higher efficiency in terms of "compression ratio" [11] compared with any other non-adaptive *VLC*. We are going to demonstrate that assertion, making comparisons with some simple examples, regarding the Huffman encoding and its most efficient version, in terms of redundancy, canonical Huffman [15]. Two leveling variants are presented and analyzed [10]: "leveled reordering", specific for low redundancy, for instance when text files are handled; and "segmented leveling", based on leveling combined with segmentation procedures to treat medium and high redundancy, for example, in image processing.

2. Leveling Encoding Process

Let's consider a file ***B*** containing a set of characters (symbols) ***b***, each of which is characterized by a frequency of occurrence, that results in an occurrence probability ***P(b)*** [9].

We have created a table with the symbols *b* sorted by occurrence of highest to lowest, where $\sum$ is the total sum of all occurrences or number of characters in the text proposed; the column $\sum$***P(b)*** indicates the partial sums of *P(b)* made of all previous symbols to *b*, including this one (see Table 1). Using (2), a value for the entropy of the file *H(B)* is obtained, associated with a minimum coding number of bits with no redundancy, given by,

$$Optimum\_bits\_number = \sum_{b} ocurrence(b)P(b) \tag{4}$$

We have incorporated in Table 1 the results of encoding the *B* file using *Huffman*, *canonical Huffman* and *leveling* (see Figure 1). With each of these encodings we offer an analysis of results from the parameters, file size encoded (in bits), $\overline{L}$, $\overline{R}$, S_R and ε. Where, S_R is the size of redundancy, defined as:

$$S_R = File\_size\_encoded - Optimum\_bits\_number \tag{5}$$

ε is the percentage of redundancy, given by:

$$\varepsilon = \frac{\overline{R}}{H} 100 \tag{6}$$

Table 1. Character Encoding Appearing in a Random Example Text

Symb. b	*Ocur.*	*P(b)%*	*ΣP(b) %*	*Huffman code*	*Canonical Huff. code*	*Leveling code*	*Decimal Meaning*
g	14	35.9	35.9	10	10	00	0
h	10	25.6	61.5	11	11	01	1
b	7	18	79.5	010	001	10	2
d	3	7.7	87.2	011	010	110	6
e	3	7.7	94.9	000	011	1110	14
c	1	2.6	97.5	0001	0000	11110	30
a	1	2.6	100	0000	0001	11111	31
	Σ=39	ΣP(b)%= 100		95 bits	95 bits *c(j)*=(2,3,2)	93 bits *c(j)*=(3,1,1,2)	
		91 bits optimum $H = 2.3247$		$\overline{L} = 2.4359$ $\overline{R} = 0.1112$ S_R=(4bits),ε=4.4%	$\overline{L} = 2.4359$ $\overline{R} = 0.1112$ S_R=(4bits),ε=4.4%	$\overline{L} = 2.3846$ $\overline{R} = 0.0599$ S_R=(2bits),ε=2.2%	

Leveling is a type of optimal entropy coding with variable length (*VLC*); that is, it assigns a binary word for each symbol and, like Huffman encoding or its canonical version, it is a *VLC* that, satisfying the prefix rule, minimizes redundancy information [8], such that:

$$H(B) \leq \overline{L} \leq H(B) + 1 \quad \text{bits/symbol} \tag{7}$$

Leveling encoding will deal with occurrence probabilities of characters individually; it does not take into account the correlation that may exist between symbols: for example, more than one character repeated successively with same value several times. In this sense, it works as Huffman encoding or canonical, so that entropy is the same. Furthermore, as the three cases are optimal *VLC*, similar sizes of the total bit coded stream are achieved, generated in different ways, but generally in favor of leveling.

Leveling is a *VLC* with "meaning", that is, symbols ordered by occurrence (high to low frequency), will also be ordered by single binary value *VLC* obtained in coding or value on a different base (for example, in decimal). Any other *VLC* does not present meaning.

On the other hand, it is important to note that, made the file encoding, the receiver must obtain not only the code in the form of binary flow ("message ensemble"), but also the necessary coding table for final decoding ("ensemble code") [8]. The transmitted table, in the case of Huffman encoding at least, consists of three equal length vectors: one containing the symbols (symbol vector), the second will be the code associated with each symbol (code vector) and the third should indicate the length of each code (length vector).

Leveling encoding, like canonical Huffman, will use tables with a reduced_code vector ("codebook") shorter than the symbol vector, taking advantage of the concept of meaning conveyed in the first. So the tables will occupy fewer bytes with the codebook than in the Huffman encoding.

In leveling encoding the reduced_code will be transmitted in a vector, we will name as ***c(j)***, which contains:

- For *j=0*, the word length of the first code, *Ini_Len* (initial lenght). Although, not absolutely necessary its transmission (in practice, will not be used).

- From *j=1*, the number of existing codes at each ***level***. *Level* means a given word length, always fulfilled all *level* is greater than or equal *Ini_Len*. For example, the binary word 1110 belongs to *level*_4.

Relating the word length of the first code *Ini_Len* with the ***maximum number of codes in the first level (MNCF)***, the latter is $(2^{Ini\_Len}-1)$. It is important to know this relationship, because it will allow us to avoid having to transmit *c(j=0)*. See results in Table 2.

Table 2. Relationship between Parameters INI_LEN and MNCF

Ini_Len	1	2	3	4	5	..	n
MNCF	1	3	7	15	31	..	2^n-1

In the text file *B* proposed as an example, the reduced code vector in leveling is *c(j)*=(2,3,1,1,2) with *j=0,1,2,..*, which means that the first code length is 2 and starts at level_2; in this level_2 there are 3 codes, in three there is 1, in four there is 1 and in level_5 there are 2 codes.

Trying to fit the reduced code vector *c(j)* for *j=1,2,3, ..* with the *MNCF* values $(1,3,7,15,..,2^n-1)$, such that all the codes *c(j)* are smaller or equal to the corresponding *MNCF* ones, we see that *c(j)* in this example, should be shifted right into a position. So, for the first level of *c(j)*, it is obtained *Ini_Len=2*. See example result proposed in the following Table 3.

Table 3. Example Search for INI_LEN, Relating *c(j)* and MNCF

Ini_Len	1	**2**	3	4	5	..
c(j)≤MNCF, ∀j	→	3	1	1	2	
MNCF	1	3	7	15	31	..

→ Rightward shift of the values of the vector *c(j)*

The leveled code (message ensemble by leveling) of the example can be obtained graphically, sorting sequentially on a bar each symbol with a length proportional to its occurrence probability (symbol bar). See Figure 1b.

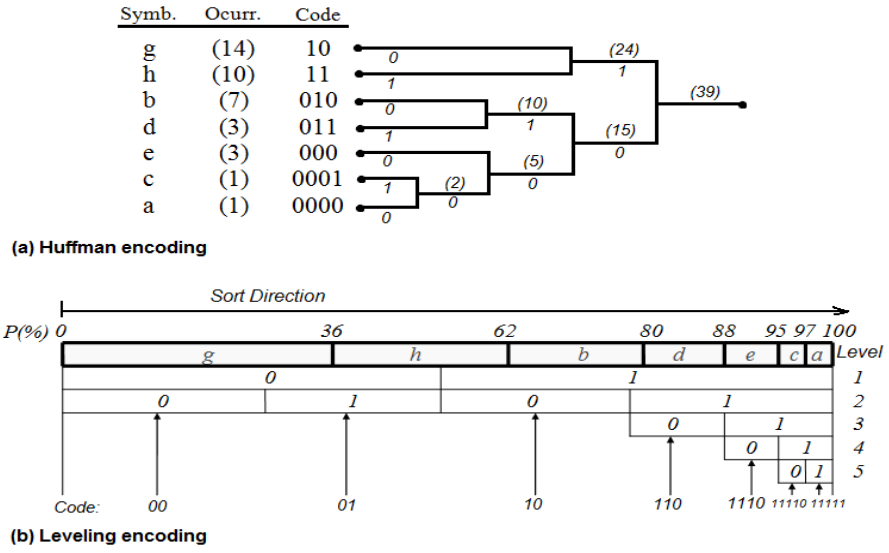

Figure 1. Graphical Representation of Occurrence Probabilities for Example Coding of the file *B* Proposed

Level bars are placed below the symbol bar, segmented into binary form by *2,4,8,.., 2^n*, equal segments, where *n* is the level of the bar. The segments of each level bar are marked with the values 0 and 1, alternatively. Levels are added, until a corresponding of a symbol by segment is taken.

Specifically, "a symbol is considered within the segment of a bar level, if it is at least half of its occurrence".

The value 0 or 1 of the segment corresponding to each symbol is the LSB ("least significant bit") of the leveled code, which is composed reading the segment values of lower levels containing, up to the first level.

For the Figure 1b proposed, it is obtained the first code (00) for the symbol *g* with a *Ini_Len* value 2. In this level_2 we have 3 codes, on the three are 1, .., etc.

The graphical procedure described shows that to get the lowest occurrence of a symbol, the longest is its code, you must meet the following: If codes are ordered by level, for example, from left to right (left shorter codes), sorting of symbols per occurrence must also be left to right (see sort direction in Figure1b). This allows assigning shorter codes with higher occurrence symbols and viceversa.

We are trying to get "*maximum ordering*", i.e. a sort of symbols occurrence as close as possible to the "*optimal entropic binary codes ordering*"; the latter, from now "*optimal entropic ordering*" is the binary codes sorting, which if the symbols to be encoded have an occurrence probability exactly proportional to 2^{-n}, with *n=1,2,..*, i.e., inversely proportional to the power of 2 for each level *n*, $\bar{L} = H$ will be obtained. Furthermore, in optimal entropic ordering, if applied graphic representation by level bars, as in Figure1b, it is observed that the shorter codes are at one end of the bars and longer at counter.

In the example of file *B*, optimal entropic ordering will be obtained with probabilities of occurrence for each symbol value, 25%(*g*), 25%(*h*), 25%(*b*), 12.5% (*d*), 6.25%(*e*), 3.125% (*c*,*a*). However, the most we can hope for is to achieve maximum ordering and, for this, the applied coding should allow achieving a code structure orderly, as optimal entropic ordering; additionally, symbols must have the same "sort direction" that the code structure with optimal entropic ordering, enabling assigning higher occurrence of shorter codes and longer for minor.

The leveling encoding fulfils the premises indicated, so that, maximum ordering is achieved, as shown in Figure 1b. However, it is not so with the Huffman encoding, or with canonical Huffman. In the first case, it is clearly seen that Huffman trees are not optimal entropic ordering structures, as longer codes need not be at one end of the tree.

In the case of canonical encoding, we will use canonical Huffman code associated with the example of the text file *B*, representing it graphically on the same terms as leveling encoding (see Figure 2). We are seeking the degree of sorting it can have.

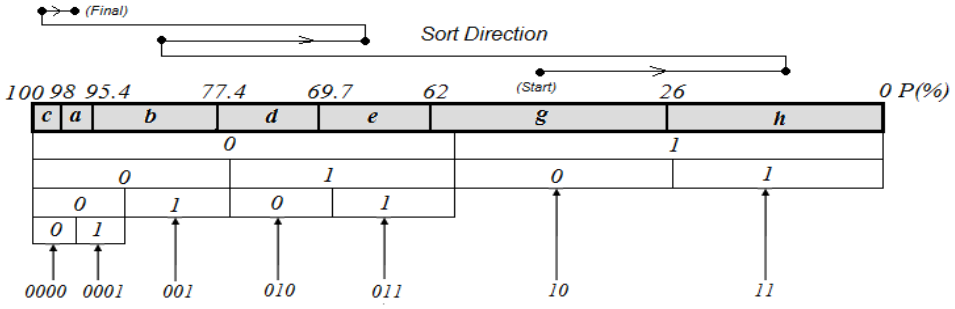

Figure 2. Graph Representation of Occurrence Probabilities by Canonical Huffman. Code Structure Ordered from Right to Left (Right Shorter Codes)

The canonical encoding neither presents maximum ordering, as shown in the example of Figure 2 (Observe the sort direction of symbols in the symbol bar, which does not follow the structure of optimal entropic ordering). Sorting is only within each level and opposite sense to how the codes are sorted in their optimal structure: in the example, can be seen that the longer codes are on the left, but the symbols within each level are ordered to the right. One might think that this does not affect the final result, because with this sort of the symbol bar is achieved codes structure with optimal entropic ordering too. However, seen in this example that if in level_3 symbols (*b*,*d*,*e*) had been sorted on the contrary (*e*,*d*,*b*), then, it would not have assigned a code of length 3 to the symbol *b*, but a code of length 2, thus reducing the overall compression ratio.

2.1. Coding Rules for Leveling

For leveling coding you have to apply the following rules:

a) Use as reference, optimal entropic ordering by level bars, segmented into 2^n elements each one, for each level *n* (*n=1,2,3,..*). So, the codes of the ensemble table to define will be in base two.
b) Order the symbols linearly over the symbol bar segmented by frequency, in descending occurrence.
c) A symbol is considered within a segment of a level bar, if it is at least half of its occurrence.

Considering these encoding rules, a process is achieved in which:

1) The coding will always start with zero value and binary word length *Ini_Len*, corresponding to the first symbol of higher occurrence.
2) Coding is continued at the same level as, at least, half of the considered symbol occurrence and, only that symbol is occupying the position of the code to assign, on the structure with optimal entropic ordering.
3) Levels are added under the symbol bar segmented, until a correspondence of a symbol per level bar segment is achieved.
4) The last symbol of lower occurrence is all "ones" in binary, with maximum word length.

Observe as for the file *B* example proposed (see Figure 3, where conditions are fulfilled for "maximum ordering"), to the first symbol of higher occurrence "*g*" corresponds code 0 with level_2; the value (in decimal base, for example) of each code is unique, not repeated; the ultimate symbol code "*a*" is 11111 (all "ones").

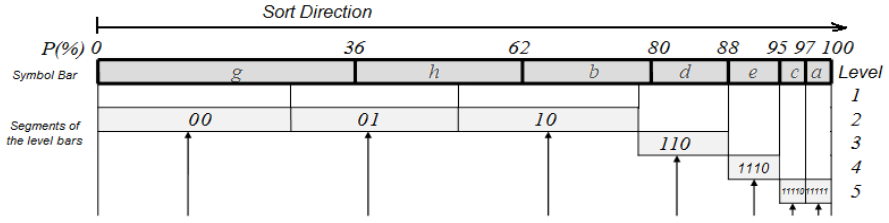

Figure 3. Correspondence between Segments of the Level Bars and the Symbol Bar for Leveling Encoding of the Example File *B* Proposed

2.2. Code Table and Reduced Code Vector *c(j)* Generation

As in Huffman encoding or canonical Huffman, with leveling two readings are needed too to encode the file: with the first one code table is constructed, using the encoding rules illustrated above, which univocally relates symbols and codes; with the second reading, the file is encoded using the code Table and, the reduced table

(symbol vector and reduced code vector) followed by the encrypted file, are transmitted.

Concrete steps in this encoding process are:

1. Parameter calculation of initial length, *Ini_Len:* Symbols must be ordered by occurrence from high to low, on a symbol bar described in terms of occurrence probability $\sum P(b)$.
   We consider $\sum$ represents the total sum of the occurrences, now expressed as an occurrence probability, that is, $\sum=1$:
   - Take into account that a symbol is within $\sum/2$, if at least half of its frequency is in $\sum/2$.
   - *Ini_Len* is defined in general according to those values proposed in Table 4, which indicates the number of codes contained in initial $\sum/2$.
2. Reduced code vector generation *c(j)*: The algorithm to develop, which is obtained directly the vector *c(j)* with, is as follows,
   (a) Initialization: j=1 and d= *Ini_Len*; ***b*** represents the value of the current symbol; ***b+1*** is the next symbol.
   (b) Do, d=(2**d*), *c(j)*=0. Divide $\sum P(b)=1$ into d parts. In each part:

If [$P(b)<1/d$ and ($P(b)+(P(b+1)/2)>1/d$)] or [$P(b)\geq 1/d$] do,
    $c(j)=c(j)+1$;
    $P(b+1)=P(b+1)+P(b)-1/d$;
    Remove current symbol b: $b=b+1$ and $b+1=b+2$;
    Next part;
else,
    $j=j+1$; return to (b)
End if.

Table 4. INI_LEN Determination

Symbol number contained in $\sum/2$	1	2 a 3	4 a 7	8 a 15	16 a 31	..	2^{n-1} a 2^n-1
Ini_Len	1	2	3	4	5	..	*n*

2.3. Reduced Code Vector *c(j)* Decoding

At the receiver, the reduced table consisting of the symbol vector and the reduced code vector *c(j)*, is extracted. The latter becomes into the code vector ***m(i)***, with same size as the symbol vector, where ***i*** is the number of symbols of the encoded file. The code vector contains codes with meaning for decoding the bit stream, from the transmission.

In the transformation of *c(j)* in *m(i)*, *we* must take into account the following rules decoding:

- Always, $m(i=1)=0$.
- To continue at the same level: $m(i)=m(i-1)+1$.
- When jumping to next level: $m(i) = [m(i-1) + 1]*2$

For the example of file *B* proposed, vector *c(j)*=(3,1,1,2) decoding, applying the above rules, gives rise to the code vector *m(i)*=(0,1,2,6,14,30,31).

Remember that in the *VLC* coded data transmission by leveling, the bit stream code is sent preceded by the reduced table, that is, it contains the reduced code vector *c(j)*, not the code vector *m(i)*; if the number of symbols *i* of the file to encode is high, better compression ratio is obtained, compared to Huffman encoding using a coding table comprising three vectors of equal length *i*.

2.4. "Meaning" in Leveling Encoding

When an encoding process fulfils the conditions for maximum ordering, indirectly, generates codes with "meaning". That is, the correspondence by maximum ordering between symbols (symbol bar) and code structure (level bars) causes codes to be also sorted by their unique value, either in binary or in another different base; besides, codes are defined between only "zeros", with minimum length *Ini_Len* (for the first symbol) and all "ones", with maximum length (for the last symbol). The characteristics of *meaning* can be summarized as:

- It is available in a single encoding process. We mean, it is not achieved by applying new rules on a previous coding. This refers to the encoding process of canonical Huffman, where there is some sorting, but got after the Huffman encoding with no sorting, applying new conversion rules. For leveling, meaning is obtained directly with a single set of simple encoding rules.
- Easy construction of reduced code vector *c(j),* in coding process (see section 2.2).
- Simple decoding of code vector *m(i)* from the vector *c(j),* (see section 2.3).
- There is a biunivocal relationship between the meaning and the maximum ordering. The latter one interests, since it generates an average length of optimum encoding better than other static encoding *VLC* also optimum.
- Each code in the code table is unique in value, in any base (binary or otherwise). Thus, the code vector can be fully transmitted by meaning, i.e., such as ASCII decimal values of each binary code; in this manner, no conversion step from *c(j)* to *m(i),* in the receiver decoding, would be necessary.

2.5. Disadvantages of Canonical Coding vs. Leveling

Canonical Huffman is a more efficient *VLC* than Huffman, by introducing certain level sorting of codes (see example in Figure 2) and so operates in transmission, as leveling, a reduced code vector, indicative of the number of codes in each level. For the example indicated in file *B*, it would be *c(j)=(2,3,2)* (see Table 1). However, being the nearest *VLC* approach to leveling, Canonical Huffman has the following shortcomings in connection therewith, resulting in an overall efficiency, at least in terms of compression ratio, slightly below:

- Canonical Huffman is a reordering of codes with respect to those obtained with Huffman encoding. Therefore, it always requires prior Huffman encoding: it is not a *VLC* encoding system itself, but an additional stage to Huffman encoding.
- Sorting is only within each level (see Figure 2) and with opposite sense to the codes as ordered: in the graphic example can be seen that the longer codes are represented on the left, but the symbols are ordered by occurrence (more to less) within each level to the right. This makes the codes obtained not have ordering meaning, as in leveling: in the example, decimal values 2,3,1,2,3,0,1, for codes in canonical Huffman, are obtained.
- It did not use meaning, so the value of the codes at any base (e.g., decimal) is not unique; it may be repeated. Direct transmission of code vector can have more problems, not following a particular scheme of sorting and allowing codes repetition, being more difficult to recover, if one is degraded during transmission.

We are developing next some encoding examples with comparative processes, where we can see the practical differences between them.

3. Segmented Leveling

Segmentation [4] can be applied for any form of static *VLC* code, either Huffman, leveling or otherwise, though not all *VLC* offer the same degree of effectiveness and adaptation to the segmentation process.

The segmentation method consists of using two correlated tables of codes, obtained from the same data set (file *B* composed of symbols *b*) [18], [19]:

1. Occurrence segments table with different intensities (values). Here, different length segments, but equal intensity, are grouped together. From the intensity vector, sorted by occurrences, the *VLC* encoding is applied and it is got the code intensity vector.
2. Occurrence segment lengths table, although representing different intensities. From the length vector, sorted by occurrences, the *VLC* encoding is applied and got the length code vector.

With the code vector of each table, the file to encode is read again and it is generated the encoded bit stream in the *pairs* form of type (*segment_length*, *segment_intensity*).

We are using an image as high redundancy file to apply different types of encoding and compare results. The picture is a frame in black and white, 24*24 pixels sized, with ten possible brightness values, indicated from 0 (the highest) to 9 (the lowest). See Figure 4.

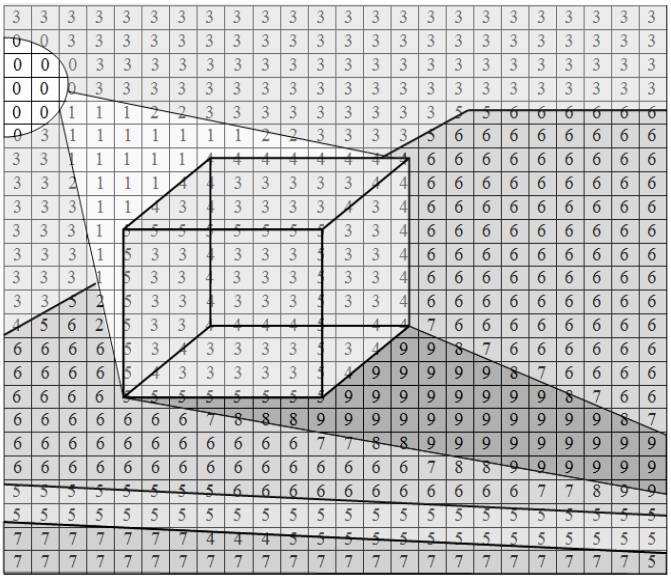

Figure 4. Image Example to Encode with Size 24*24 Pixels and 10 Levels of Brightness (Maximum 0, to Minimum 9)

We are applying the following coding processes for subsequent comparison:

- "Run lenght encoding" o RLE [7]: generation of pairs of runs (*segment_length*, *segment_intensity*) encoded in ASCII.
- Huffman: Bit stream coded plus code table, composed by three vectors.
- Canonical Huffman: bit stream coded plus reduced code table (two vectors).
- Leveling: bit stream coded plus reduced code table (two vectors).
- Huffman, canonical Huffman and leveling with *VLI*: we will apply the technique of "variable length integer" or *VLI* on the table or reduced code table [4].
- Huffman, canonical Huffman and leveling segmented with *VLI*: bit stream coded by segmentation plus table or reduced code table with *VLI*.
- Canonical Huffman and leveling segmented double *VLI*: Bit stream coded by segmentation plus reduced code table with double *VLI*: one for symbol vector and another for reduced code vector.

The different type comparison of suggested coding is regarding the results of transmitting without encoding image, that is:

- No encoding: 576 bytes = 4608 bits, using ASCII.

3.1. Run Length Encoding

Pairs of runs generating (*segment_length*, *segment_intensity*) on the image of Figure 4, with the following result:
24,3,2,0,22,3,3,0,21,3,3,0,15,3,6,5,2,0,3,1,2,2,9,3,2,5,6,6,1,0,1,3,7,1,2,2,4,3,1,5,8,6,2,3,5,1,8,4,9,6,
2,3,1,2,3,1,2,4,5,3,2,4,5,3,2,4,9,6,3,3,1,2,2,1,1,4,1,3,1,4,4,3,1,4,1,3,1,4,9,6,3,3,1,1,8,5,2,3,1,4,9,6,3,
3,1,1,1,5,2,3,1,4,9,6,3,3,1,1,1,5,2,3,1,4,3,3,1,5,2,3,1,4,9,6,2,3,1,5,1,2,1,5,2,3,1,4,3,3,1,5,2,3,1,4,9,6,
1,4,1,5,1,6,1,2,1,5,2,3,4,4,1,5,3,4,1,7,8,6,4,6,1,5,1,3,1,4,4,3,1,5,1,3,1,4,2,9,1,8,1,7,6,6,4,6,1,5,1,4,5,
3,1,5,1,4,5,9,1,8,1,7,4,6,4,6,8,5,8,9,1,8,1,7,2,6,7,6,1,7,3,8,11,9,1,8,1,7,22,6,2,7,2,8,9,9,14,6,1,7,2,8,
6,9,8,5,11,6,2,7,1,8,2,9,24,5,7,7,3,4,14,5,23,7,1,5

These pairs represent ASCII encoded 280 bytes = 2240 bits.

3.2. Huffman, canonical Huffman and leveling encoding

Codes obtained and main results, in order to encode the image of Figure 4 with Huffman, canonical Huffman and leveling, respectively, are summarized in Table 5.

Table 5. Comparing VLCs, Huffman-Canonical Huffman-Leveling for the Example in Figure 4

Symb. b	*Ocurr.*	*P(b)%*	*ΣP(b) %*	*Huffman code*	*Canonical Huff.code*	*Leveling code*	*Decimal Meaning*
3	161	28	28	11	10	00	0
6	154	26.8	54.8	10	11	01	1
5	85	14.8	69.6	011	010	100	4
9	44	7.6	77.2	010	011	101	5
7	41	7.1	84.3	0011	0010	1100	12
4	38	6.6	90.9	0010	0011	1101	13
1	23	4	94.9	00011	00000	1110	14
8	12	2.1	97	00010	00001	11110	30
0	11	1.9	98.9	00001	00010	111110	62
2	7	1.2	100	00000	00011	111111	63
	Σ=576	ΣP(b)%= 100		1598 bits	1598 bits *c(j)*=(2,2,2,4)	1593 bits *c(j)*=(2,2,3,1,2)	
		H = 2.7404 1579bits optimum		$\overline{L}$ = 2.7743 $\overline{R}$ = 0.0339 S_R=(19bits),ε=1.2%	$\overline{L}$ = 2.7743 $\overline{R}$ = 0.0339 S_R=(19bits),ε=1.2%	$\overline{L}$ = 2.7656 $\overline{R}$ = 0.0252 S_R=(14bits),ε=0.9%	

In Figure 5, it is shown the Huffman coding tree for the example, associated to Table 5.

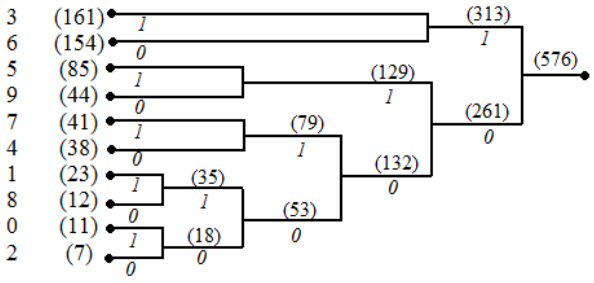

Figure 5. Huffman Tree for the Example Image of Figure 4

In Table 6, it is indicated the algorithm used to convert Huffman codes to canonical Huffman and, its application to the example in Table 5. The parameters "length", "first" and "number" represent, respectively, the word length of each code, the decimal value of the first code with length given and the number of codes with

length indicated. Note that the values of the parameters "length" and "number" are given by Huffman encoding, while the parameter "first" is obtained with the algorithm shown in Table 6.

Table 6. Huffman to Canonical Huffman Code Conversion for Figure 4

First (Length)=[First(Length+1)+Number(Length+1)]/2					
Length	1	2	3	4	5
Number	0	2	2	2	4
First	2	2	2	2	0

In Figure 6, it is represented graphically coding process by leveling for the example considered, associated with Table 6.

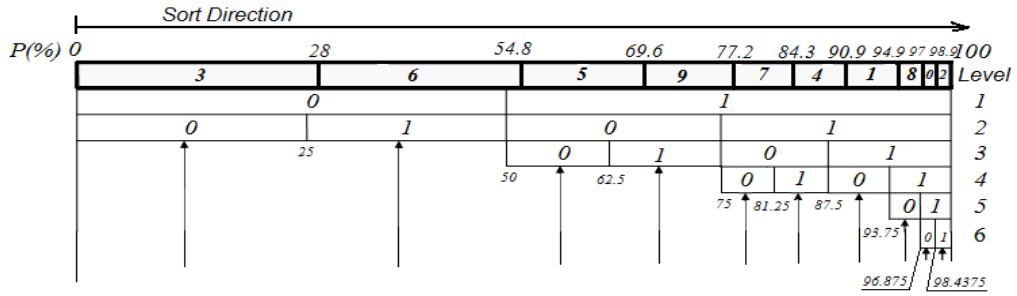

Figure 6. Bar Graph for Leveling Encoding of the Image Example in Figure 4

3.3. Huffman, canonical Huffman and leveling encoding with VLI

The application of the *VLI* technique on the code table or reduced code table in the example proposed, has the following effect on the encodings listed in section 3.2 above:

- Huffman encoding: among the 30 codes of the code table, the highest value is 9, in binary 1001, with length 4; therefore, instead of encoding ASCII length 8, can be used VLI length 4. Thus, a byte is used to indicate the VLI=4 and 15 bytes to encode the 120 bits of the table (30 codes of length 4).
- Canonical Huffman: among the 10+4 codes on the reduced code table, the highest value is 9, therefore, deserves in VLI words with length 4. Thus, a byte is used to indicate the VLI=4 and 7 bytes to encode 56 bits of the reduced table (14 codes of length 4).
- Leveling: among the 10+5 codes on the reduced code table, the highest value is 9, which corresponds VLI with length 4. Thus, a byte is used to indicate the VLI=4 and 8 bytes to encode 60 bits of the reduced table (15 codes of length 4).

3.4. Huffman, canonical Huffman and leveling encoding segmented with VLI

Firstly, we have obtained the codes and main results, in order to code the different intensities on the image of Figure 4 with Huffman, canonical Huffman and leveling, respectively. These data are summarized in Table 7.

Figure 7 shows the Huffman encoding tree for intensities of the Table 7.

Table 8 develops the algorithm used for conversion from Huffman codes to canonical Huffman and their application in the example of Table 7.

In Figure 8, it is plotted the encoding process by leveling for the example considered, associated to intensities in Table 7.

Secondly, we have sought the codes and main results, associated with segment lengths over the image of Figure 4 with Huffman, canonical Huffman and leveling, respectively: see Table 9.

Table 7. Coding of the Intensity (Segmentation) with Huffman, Canonical Huffman and Leveling

Intens. b	*Ocurr.*	*P(b)%*	*ΣP(b) %*	*Huffman code*	*Canonical Huff. code*	*Leveling code*	*Decimal Meaning*
3	34	23.6	23.6	11	11	00	0
5	23	16	39.6	101	010	010	2
4	22	15.3	54.9	100	100	011	3
6	21	14.6	69.5	011	101	100	4
7	11	7.6	77.1	0101	0001	101	5
1	8	5.6	82.7	0100	0010	1100	12
8	8	5.6	88.3	0011	0011	1101	13
9	7	4.9	93.2	0010	0100	1110	14
2	5	3.5	96.7	00001	00000	11110	30
0	5	3.5	100	00000	00001	11111	31
	Σ=144	*ΣP(b)%* =100		452 bits	452 bits *c(j)*=(1,3,4,2)	441 bits *c(j)*=(1,4,3,2)	
		$H=3.049$ 439 bits optimum		$\overline{L}=3.1389$ $\overline{R}=0.0898$ S_R=(13bits),ε=3%	$\overline{L}=3.1389$ $\overline{R}=0.0898$ S_R=(13bits),ε=3%	$\overline{L}=3.0625$ $\overline{R}=0.0134$ S_R=(2bits),ε=0.5%	

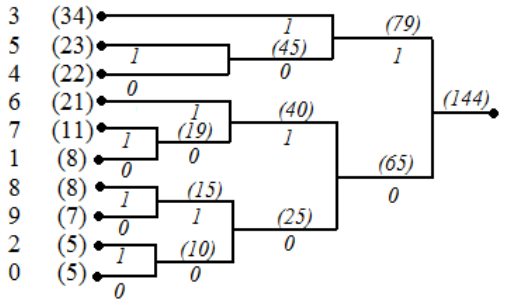

Figure 7. Huffman Tree for Table 7 of Intensities

Table 8. Huffman to Canonical Huffman Code Conversion for Table 7

First (Length)=[First(Length+1)+Number(Length+1)]/2					
Length	1	2	3	4	5
Number	0	1	3	4	2
First	2	3	3	1	0

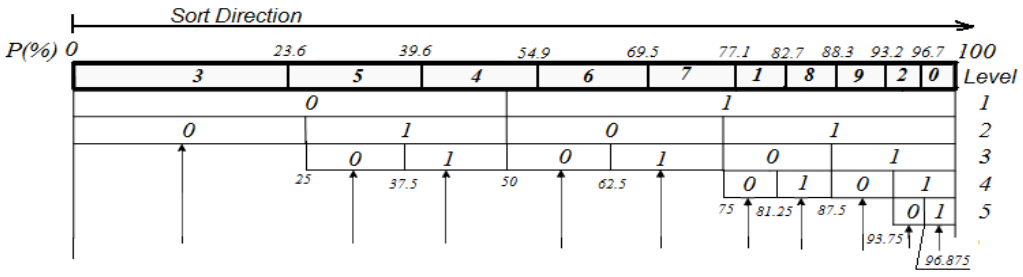

Figure 8. Bar Graph for Leveling Encoding of the Intensities in Table 7

Figure 9 represents the Huffman encoding tree for lengths in Table 9.

Table 10 shows the conversion from Huffman codes to canonical Huffman and its application on the example of Table 9.

Table 9. Coding of the Length (Segmentation) with Huffman, Canonical Huffman and Leveling

Length b	*Ocurr.*	*P(b)%*	*ΣP(b)%*	*Huffman code*	*Canonical Huff. code*	*Leveling code*	*Decimal Meaning*
1	36	34	34	11	10	00	0
2	20	18.9	52.9	10	11	01	1
3	9	8.5	61.3	0111	0100	100	4
4	7	6.6	67.9	0110	0101	1010	10
8	6	5.7	73.6	0101	0110	1011	11
9	6	5.7	79.3	0100	0111	1100	12
6	4	3.8	83.1	00011	00010	11010	26
5	4	3.8	86.9	00110	00011	11011	27
11	3	2.8	89.7	00101	00100	11100	28
7	3	2.8	92.5	00100	00101	11101	29
14	2	1.9	94.4	00011	00110	111100	60
24	2	1.9	96.3	00010	00111	111101	61
22	1	0.9	97.2	000011	000000	1111100	124
21	1	0.9	98.1	000010	000001	1111101	125
15	1	0.9	99	000001	000010	1111110	126
23	1	0.9	100	000000	000011	1111111	127
	Σ=106	*ΣP(b)%* =100		338 bits	338 bits *c(j)*=(2,0,4,6,4)	337 bits *c(j)*=(2,1,3,4,2,4)	
		$H = 3.103$ 329 bits optimum		$\overline{L} = 3.1897$ $\overline{R} = 0.1396$ S_R=(9bits),ε=2.7%	$\overline{L} = 3.1897$ $\overline{R} = 0.1396$ S_R=(9bits),ε=2.7%	$\overline{L} = 3.1793$ $\overline{R} = 0.0754$ S_R=(8bits),ε=2.4%	

Figure 10 shows graphically the bar chart of the coding process by leveling in the example considered, associated to lengths in Table 9.

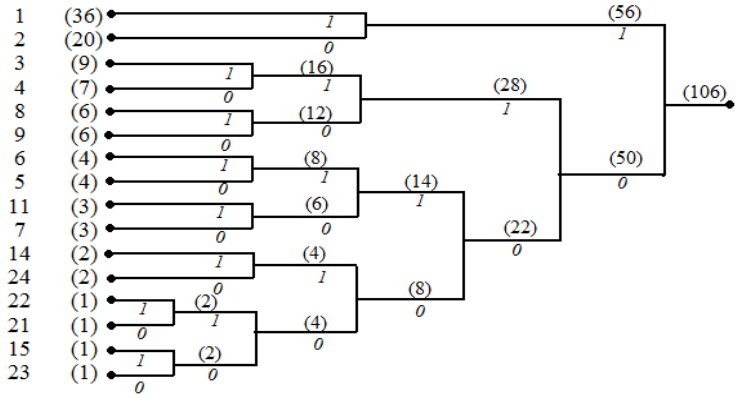

Figure 9. Huffman Tree for Table 9 of Lengths

Table 10. Huffman to Canonical Huffman Code Conversion for Table 9

First (Length)=[First(Length+1)+Number(Length+1)]/2						
Length	1	2	3	4	5	6
Number	0	2	0	4	6	4
First	2	2	4	4	2	0

The transmission of data encoded using segmentation on Figure 4, are summarized as follows:

- Segmented Huffman encoding with VLI: Code tables [(10+16)*3*5]bits + 1byte_VLI(5) + encoded stream (452+338)bits = 1188bits =149bytes
- Segmented Canonical Huffman with VLI: Reduced tables [(10+4+16+5)*5]bits + 1byte_VLI(5) + encoded stream (452+338)bits = 973bits = 122bytes
- Segmented Leveling with VLI: Reduced tables [(10+4+16+6)*5]bits + 1byte_VLI(5) + encoded stream (441+337)bits = 966bits = 121bytes

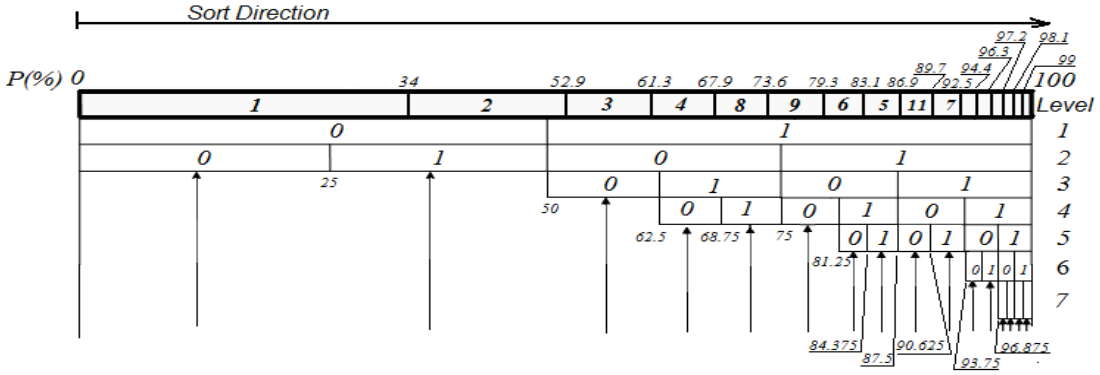

Figure 10. Bar Graph for Leveling Encoding of the Lengths in Table 9

3.5. Canonical Huffman and Leveling Encoding Segmented with Double VLI

Instead of applying the technique VLI in common, over symbol vector and reduced code vector, in the reduced code tables obtained in paragraph 3.4 above, we are going to do it now, on each of them independently; as Huffman encoding uses code tables consisting of three vectors, double VLI is only applicable in canonical Huffman and leveling:

- Canonical Huffman: among the 10+16 codes in symbol vectors, the highest value is 24, therefore, corresponds in VLI words of length 5; among the 4+5 codes in both reduced code vectors, the higher value is 6, therefore, corresponds in VLI words of length 3. Thus, two bytes are used to indicate the VLI=5 and the VLI=3 and 20 bytes to encode 157 bits, in the two reduced tables (26 codes of length 5 and 9 codes of length 3).
- Leveling: among the 10+16 codes in symbol vectors, the highest value is 24, therefore, the VLI will be 5; among the 4+6 codes in reduced code vectors, the highest value is 4, which corresponds in VLI value 3. Thus, two bytes are used to indicate the VLI=5 and the VLI=3 and 20 bytes to encode 160 bits, in the two reduced tables (26 codes of length 5 and 10 codes of length 3).

Table 11. Results of Encoding Image in Figure 4

Procedure	*Trans.Code Stream(bits)*	*VLI (bytes)*	*Table (bytes)*	*Table (bits)*	*Total (bits)*	*Total (bytes)*	*Compression Ratio*
No Compression	4608	-	-	-	4608	576	1:1
RLE	2240	-	-	-	2240	280	1:2.06
Huffman	1598	-	30	240	1838	230	1:2.50
Can. Huffman	1598	-	14	112	1710	214	1:2.69
Leveling	1593	-	15	120	1713	215	1:2.68
Huffman_VLI	1598	1	15	120	1726	216	1:2.67
Can. Huf. VLI	1598	1	7	56	1662	208	1:2.77
Leveling_VLI	1593	1	8	60	1661	208	1:2.77
Segm.Huffman_VLI	790	1	49	390	1188	149	1:3.87

Segm.Can.Huf._VLI	790	1	22	175	973	122	1:4.72
Segm. Leveling_VLI	778	1	12	90	966	121	1:4.76
Segm.Leveling_2VLI	778	2	20	160	954	120	**1:4.80**
Segm.Can.Huf._2VLI	790	2	20	157	963	121	1:4.76

3.6. Summary of Encoding Results

Table 11 indicates results of encoding the image example Figure 4, with the methods proposed in the different parts of the section 2. We have included the compression ratio associated with each type of encoding, defined as the ratio of bytes needed to transmit uncompressed image and those obtained with the coding technique considered.

Observe that for same conditions of coding (pure method, with segmentation, VLI, ..), leveling is the one that offers the best results in terms of compression ratio.

Table 12. Text Encoding "Abstract" in Spanish with Huffman, Canonical Huffman and Leveling, in Section 4

Symbol	*Ocurr.*	*P(Oc)%*	*ΣP(Oc)%*	*Huffman*	*Can.Huff.*	*Leveling*	*Dec.Mean.*
Space	120	15	15	111	111	000	0
a	77	9.7	24.7	1101	1010	001	1
e	70	8.8	33.5	1100	1011	0100	4
i	51	6.4	39.9	1011	1100	0101	5
o	50	6.3	46.2	1010	1101	0110	6
n	50	6.3	52.5	10011	01000	0111	7
d	40	5	57.5	10010	01001	1000	8
r	37	4.7	62.2	10001	01010	1001	9
s	33	4.2	66.4	10000	01011	1010	10
c	32	4	70.4	01111	01100	10110	22
t	27	3.4	73.8	01110	01101	10111	23
l	24	3	76.8	01101	01110	11000	24
u	20	2.5	79.3	01100	01111	11001	25
p	19	2.4	81.7	01011	10000	110100	52
m	17	2.1	83.8	01010	10001	110101	53
,	13	1.6	85.4	01001	10010	110110	54
“	12	1.5	86.9	01000	10011	110111	55
ó	12	1.5	88.4	001111	001000	111000	56
v	11	1.4	89.8	001110	001001	111001	57
g	10	1.3	91.1	001101	001010	1110100	116
f	8	1	92.1	001100	001011	1110101	117
b	7	0.9	93	001011	001100	1110110	118
L	5	0.6	93.6	001010	001101	1110111	119
y	5	0.6	94.2	001001	001110	1111000	120
V	4	0.5	94.7	001000	001111	11110010	242
C	4	0.5	95.2	0001111	0000100	11110011	243
N	4	0.5	95.7	0001110	0000101	11110100	244
á	3	0.4	96.1	0001101	0000110	11110101	245
j	3	0.4	96.5	0001100	0000111	11110110	246
9	3	0.4	96.9	0001011	0001000	11110111	247
5	3	0.4	97.3	0001010	0001001	11111000	248
.	3	0.4	97.7	0001001	0001010	11111001	249
q	3	0.4	98.1	0001000	0001011	11111010	250
h	2	0.3	98.4	0000111	0001100	11111011	251
H	2	0.3	98.7	0000110	0001101	11111100	252
1	2	0.3	99	0000101	0001110	111111100	506
J	2	0.3	99.2	0000100	0001111	111111101	507
M	1	0.13	99.3	00000111	00000000	1111111000	1016
E	1	0.13	99.4	00000110	00000001	1111111001	1017
S	1	0.13	99.5	00000101	00000010	1111111010	1018
ñ	1	0.13	99.6	00000100	00000011	1111111011	1019
í	1	0.13	99.7	00000011	00000100	1111111100	1020
z	1	0.13	99.8	00000010	00000101	1111111101	1021

x	1	0.13	99.9	00000001	00000110	1111111110	1022
R	1	0.13	100	00000000	00000111	1111111111	1023
ASCII 8 bits	Σ=796	H=4.4673		*c(j)*=(1,4,12,8,12,8)		*c(j)*=(2,7,4,6,5,11,2,8)	
	6368 bits	3556bits optimum		3646 bits, $\bar{L}=4.5804$		3567 bits, $\bar{L}=4.4812$	
				$\bar{R}=0.1131$, S_R=(90bits), ε=2.5%		$\bar{R}=0.0139$, S_R=(11bits), ε=0.3%	

4. Leveled Reordering

The application of pure leveling encoding on low redundancy files, where makes no sense to combine it with additional techniques such as segmentation, which operates well with high redundancy, will be appointed as "leveled reordering".

Let's use an example text as low redundancy file, to apply different types of encoding and compare results. In particular we will use the abstract in Spanish paragraph of this paper, included as appendix, consisting of 121 words with 45 different symbols, generating a total of 796 alphanumeric characters.

Table 12 above represents the coding tables of text example indicated with Huffman, canonical Huffman and leveling. Includes reduced code vectors *c(j)* for canonical Huffman and leveling, besides the results parameters for the three encodings: the encoded file size, average word length $\bar{L}$, redundancy $\bar{R}$, redundancy size in bits S_R and redundancy percentage ε.

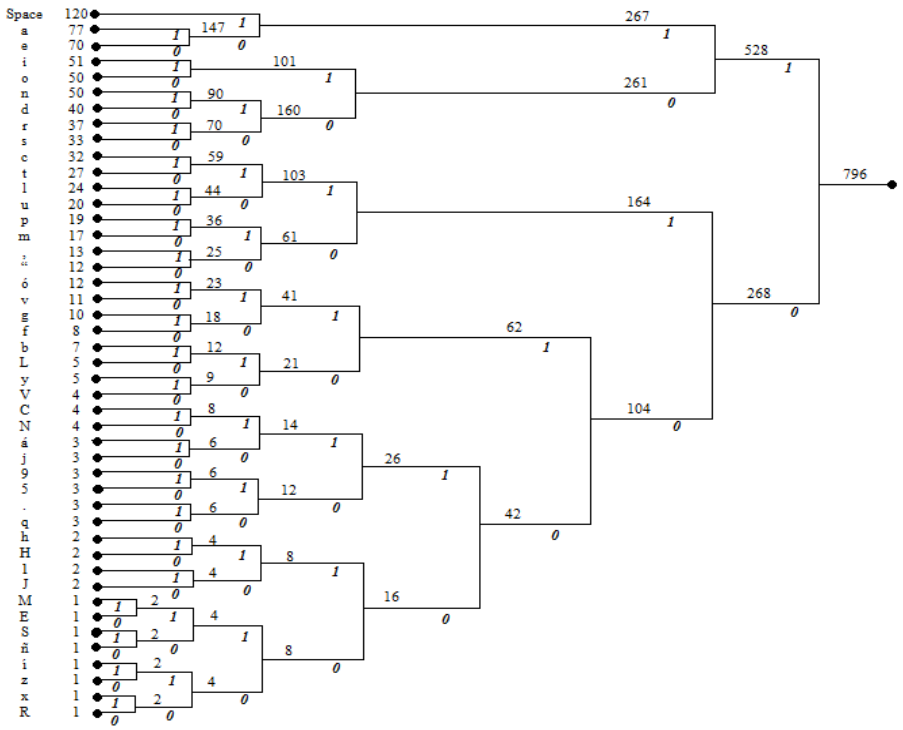

Figure 11. Huffman Coding Tree of the Example Text, "Abstract" in Spanish

Figure 11 is the Huffman coding tree associated with the text example shown.

Figure 12 is a graphical bar representation of coding leveling example text.

Table 13 indicates the conversion process from Huffman codes to canonical Huffman, for the example text used.

Finally, the results summary of the text coding example, where it can be seen that the maximum compression ratio is for leveling, is given in Table 14.

Appendix: Abstract in Spanish

Muchos de los procedimientos estándar de compresión están basados, en su nivel más bajo, en codificación de palabras digitales de longitud variable o VLC del tipo Huffman, diseñado en 1952 o en alguna de sus versiones, como la canónica. En este artículo se presenta un VLC de mayor eficiencia que la codificación Huffman, nombrado como "Nivelación" y que utiliza dos variantes, "Reordenación Nivelada" para baja redundancia, por ejemplo para textos y, "Nivelación Segmentada" para media y alta redundancia, para tratamiento de imagen. La Nivelación, desarrollada por Javier Joglar en 1995, aprovecha los conceptos de "significado" y "ordenación" de los códigos VLC generados, para conseguir un rendimiento óptimo en cuanto a "rate de compresión", superior al de cualquier otro VLC no adaptativo.

5. Conclusions

Lossless compression method presented is based on the reorganization by levels, offering different yields, depending on the characteristics of the information we have to treat. The "compression ratio", function of the type of coding applied, as well as, the characteristics of the information to code, in leveling coding is higher than any other non-adaptive VLC (Huffman encoding or canonical Huffman, *e.g.*,) due to the concepts of "meaning" and "ordering". So, "leveled reordering" provides maximum performance on data with low level of successive states repetition, being optimal for word processing or heterogeneous images with no correlative brightness levels; "Segmented Leveling" presents maximum efficiency applied on data with high level of successive states repetition, for example, in image processing in general, where the different brightness levels are usually repeated successively.

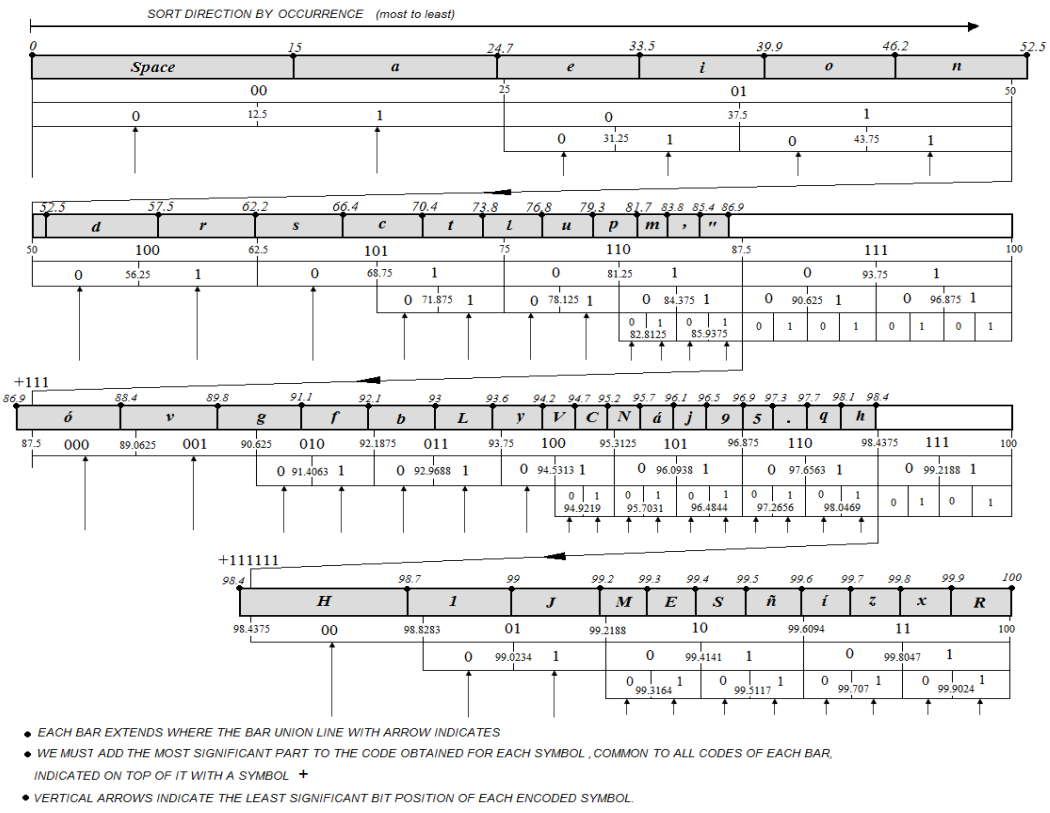

Figure12. Leveling Encoding Bar Graph of the Text, "Abstract" in Spanish

Table 13. Huffman to Canonical Huffman Code Conversion for Example Text

First (Length)=[First(Length+1)+Number(Length+1)]/2								
Length	1	2	3	4	5	6	7	8
Number	0	0	1	4	12	8	12	8
First	2	4	7	10	8	8	4	0

Table 14. Text Encoding Results "Abstract" in Spanish

Procedure	*Transm.Code Stream (bits)*	*VLI (bytes)*	*Table (bytes)*	*Table (bits)*	*Total bits*	*Total bytes*	*Compression ratio*
No Compression	6368	-	-	-	6368	796	1:1
Huffman	3646	-	135	1080	4726	591	1:1.34
Canon. Huffman	3646	-	51	408	4054	507	1:1.57
Leveling	3567	1	53	424	3991	499	1:1.60
Canon. Huf. VLI	3646	1	48	384	4038	505	1:1.58
Leveling VLI	3567	1	49	392	3967	496	**1:1.61**

References

[1] M. Abdul Mannan and M. Kaykobad, "Block Huffman Coding", Computers and Mathematics with Applications, vol. 46, **(2003)**, pp. 1581-1587.
[2] Y. S. Abu-Mostafa and R. J. McEliece, "Maximal Codeword Lengths in Huffman Codes", Computers and Mathematics with Applications, vol. 39, **(2000)**, pp. 129-134.
[3] R. Al-Hashemi and I. Wahbi Kamal, "A New Lossless Image Compression Technique Based on BCH Codes", Int. Journal of Software Engineering and Its Applications, vol. 5, no. 2, **(2011)**, pp. 15-22.
[4] W. E. Carlson, "A Survey of Computer Graphics Image Encoding and Storage Formats", Computer Graphics, vol. 25, no. 2, **(1991)**, pp 67-75.
[5] C. Chan, "Variations on Theme by Huffman: a Literature Review", **(2006)**, May.
[6] R. G. Gallager, "Variations on a Theme by Huffman", IEEE Transactions on Information Theory, vol. 24, no. 6, **(1978)**, pp. 668-674.
[7] S. W. Golomb, "Run-Length Encodings", IEEE Transactions on Information Theory, vol. 12, no.13, **(1966)**, pp. 399-401.
[8] D. A. Huffman, "A Method for the Construction of Minimum Redundancy Codes", Proc. IRE., vol. 40, no.9, **(1952)**, pp. 1098–1101.
[9] J. Joglar Alcubilla, "Comparación en MATLAB de la Eficacia de un Nuevo Procedimiento para la Codificación de Señales de Vídeo sin Pérdida", Congreso de Usuarios de Matlab, UNED, Spain, **(1996),** [Online Available]: https://www.dropbox.com/s/yveueageabf8u5e/video.pdf?dl=0.
[10] J. Joglar Alcubilla, "Video Compresión", [Online Available]: http://www.lulu.com/spotlight/inercia, **(2009)**, Nov.
[11] S. R. Kodituwakku and U. S. Amarasinghe, "Comparison of Lossless Data Compression Algorithms for Text Data", Indian Journal of Computer Science and Engineering, vol. 1, no.4, **(2010)**, pp. 416-425.
[12] N. Krishnan and D. Baron, "A Universal Parallel Two-Pass MDL Context Tree Compression Algorithm", arXiv:1407.1514v4 [cs.IT], **(2015)**, 21 Mar.
[13] M. Prantl, "Image Compression Overview", arXiv:1410.2259v1 [cs.GR], **(2014)**,14 Sep.
[14] R. F. Rice and R. Plavat, "Adaptive Variable-Length Coding for Efficient Compression of Spacecraft Television Data", IEEE Transactions on Communications, vol. 16, no. 9, **(1971)**, pp. 889-897.
[15] A. San Emeterio Campos, "Canonical Huffman", [Online Available]: http://www.arturocampos.com/ac_canonical_huffman.html, **(1999).**
[16] D. Venkatasekhar and P. Aruna, "A Fast Fractal Image Compression Using Huffman Coding", Asian Journal of Computer Science and Information Technology, vol. 2, no. 9, **(2012)**, pp. 272-275.
[17] D. Vohl, C. J. fluke and G. Vernardos, "Data Compression in the Petascale Astronomy Era: a GERLUMPH Case Study", arXiv:1505.05617v2 [astr-ph.IM], **(2015)**,22 May.
[18] Y. Ma, H. Derksen and W. Hong, "Segmentation of Multivariable Mixed Data Via Lossy Data Coding and Compression", IEEE Trans Pattern Anal Mach Intell, vol. 29, no. 9, **(2007)**, pp. 1546-1562.
[19] Q. Min and R.J.T. Sadleir, "A Segmentation Based Lossless Compression Scheme for Volumetric Medical Image Data", Machine vision and image processing conference, pp. 101-102, **(2011)**,September.

Author

Javier Joglar-Alcubilla, He works as a professor in the avionics department of the Barajas Institute in Madrid, Spain. He received his B.Sc in Aeronautics Engineering from Polytechnic University at Madrid, Spain, in 1990 and his M.Sc degree in Electronics and Automation Physics from UNED, Spain, in 1994. He received his Ph.D. degree in Informatics Engineering from ETSII at UNED, Spain, in 2015. He is the author of nine technical books.