Lossless Scheme for Data Compression using Metasymbols

Ángel F. Kuri Morales[1], Oscar Herrera Alcántara[2]

[1]Instituto Tecnológico Autónomo de México
Río Hondo No. 1 México D.F.
akuri@itam.mx

[2]Centro de Investigación en Computación
Av. Juan de Dios Batiz S/N
Unidad Profesional "Adolfo López Mateos" – Zacatenco
heoscar@sagitario.cic.ipn.mx

ABSTRACT

Huffman Coding provides a mechanism to minimize the average length of codes assigned to symbols from an information source. The average length reaches the minimum theoretical limit bounded by the entropy when symbols have a probability distribution which is an exact integer power of ½. Shannon showed that an n-th extension of a code allows to reach the entropy as n tends to infinite. However, in a finite message that is not possible. In this work we present a scheme of lossless data compression which allows us to find digrams, trigrams,...,γ -grams (for unbounded γ) as Shannon suggested in order to get maximum compression.

Keywords. Entropy, Extension of a code, Huffman Coding, Information Source.

1. INTRODUCTION

In his original work about Information Theory, Shannon [1] defined the amount of information assigned to a symbol generated by a source as:

$$I(s_i) = -\log(p_i) = \log\left(\frac{1}{p_i}\right) \quad (1)$$

where s_i denotes the i-th symbol and p_i denotes its apparition probability.

The coding process of the symbols s_i consists in establishing a correspondence between each of them (named symbols of the source alphabet) and a sequence of symbols of a target alphabet. The target alphabet is called a code alphabet and each of the symbol sequences of the target alphabet that corresponds with a symbol s_i is called a codeword. Coding is desired for different purposes such as encryption, transmission and compression. This last is the subject of this work.

How to achieve compression? As we see, we can assign a codeword of length l_k to each s_i. For example, suppose we have five symbols s_1='a', s_2='b', s_3='c', s_4='d', s_5='e' which are iid and that the target alphabet is {0,1}. Table 1 shows their probabilities as well as some arbitrary coding and their respective lengths. Considering that the target alphabet is {0,1} we can measure the length l_k in bits.

TABLE 1.
FIVE SYMBOLS, ITS PROBABILITY AND SOME ARBITRARY CODING

Column1 s_i	Column2 Probability	Column3 Arbitrary Coding	Column4 l_k(bits)
s_1 = a	p_1=1/3	$s_1 \rightarrow 0$	1
s_2 = b	p_2=4/15	$s_2 \rightarrow 10$	2
s3= c	p_1=1/5	$s_1 \rightarrow 110$	3
s4 =d	p_1=2/15	$s_1 \rightarrow 1110$	4
s5 =e	p_1=1/15	$s_1 \rightarrow 1111$	4

The average length of a code is defined as

$$\bar{L} = \sum_{k=1}^{M} p_k l_k \quad (2)$$

where M is the number of different symbols in a message and $\bar{L}$ represents the average number of bits needed by each symbol s_i in the encoding process. In the example of table 1, M=5 and $\bar{L} = 2.26$

The average information of the source (called its entropy) is

$$H(x) = \sum_{k=1}^{M} p_k \log\left(\frac{1}{pk}\right) \quad (3)$$

$$H(x) = 2.14 \quad (4)$$

which represents the theoretical limit of the average length of all codes assigned to the associated information source. As we can see $H(x) = 2.14 < \bar{L} = 2.26$. The proof of the general case may be found in the Shannon's First Theorem where the minimum average length L_{min} that we can assign to the symbols complies with

$$L_{min} \geq H(x) \tag{5}$$

Shannon states this result but does not discuss how to assign the codes. Later, Huffman [2] established a technique to promote compression by assigning short codewords to symbols which appear more often and large codewords to symbols which occurs less frequently. Therefore, the average length is minimized.

2. CODING AND COMPRESSION

In the previous example, the apparition probabilities of table 1 can be obtained from a message such as Msg1= "aaaaabbbbcccdde". Each s_i corresponds to a ASCII character (8 bits) so, the message has a length of 120 bits. With the coding of table 1 we can codify the message concatenating codewords of each symbol and we get the coded message CodedMsg1="000001010101011011011011101110 11111". The length of the Coded Message CodedMsg1 is 35 bits

To decode Msg1 we need to remember that 0 represents the ASCII character 'a', 10 represents 'b', 110 represents 'c', 1110 represents 'd' and 11111 represents 'e', i.e., besides the coded message we need columns 1 and 3 of the table 1 (which is known as its "dictionary").

For simplicity we represent the ASCII value of a character like 'character'. Now the fully coded message formed with the dictionary and the coded message looks like:
0'a'10'b'110'c'1110'd'11111'e'00000101010101101 101101110111011111
The length of the fully coded message L_{full} is calculated with the bits of the dictionary plus the bits of the coded message

$$L_{full} = \sum_{k=1}^{M} l_k + 8 * M + L * \bar{L} \tag{6}$$

where:
L is the length of the message, i.e., the number of symbols which conforms the message.

f_k is the frequency of a symbol in the message coded, in fact, p_k is estimated by

$$p_k = \frac{f_k}{L} \text{ and } L = \sum_{k=1}^{M} f_k \tag{7}$$

$$L_{full} = 14 + 40 + 15 * 2.2 = 87 \text{ bits} \tag{8}$$

which is less than 120 bits, the number of bits of Msg1 coded with ASCII values. Therefore, we can think that compression is a consequence of a coding chosen *ad hoc* [3].

2.1 Huffman Coding

In table 1 we choose an arbitrary coding which is not optimal for compression because it is possible to find at least another one which yields better compression. That is the case of Huffman codes which assigns the codewords efficiently.

In table 2 we show a Huffman Coding for the symbols and probabilities of table 1.

TABLE 2. HUFFMAN CODING GIVEN A SET OF PROBABILITIES

s_i	Probability	Huffman Coding	l_k (bits)
s_1	p_1=1/3	$s_1 \rightarrow$11	2
s_2	p_2=4/15	$s_2 \rightarrow$ 10	2
s_3	p_1=1/5	$s_1 \rightarrow$ 01	2
s_4	p_1=2/15	$s_1 \rightarrow$001	3
s_5	p_1=1/15	$s_1 \rightarrow$ 000	3

Using table 2 the coded message with Huffman codes including the dictionary is

HuffMsg1=11'a'10'b'01'c'001'd'000'e'11111111111 0101010010101001001000

with a length of 85 bits, which is lower than the two cases previously studied. That is the advantage of Huffman codes over other encoding schemes.

The average length of Huffman codes of table 2 are $\bar{L}_{huff} = 2.2$ as we can appreciate

$$\bar{L}huff = 2.2 \geq H(x) = 2.14 \tag{9}$$

The best case of Huffman Coding happens when symbols have a probability distribution with exact integer power of ½ then $\bar{L} = H(x)$ and we get the best possible encoding.
The worst case for Huffman Coding happens when probabilities are not exact power of ½, because

tropy always is less than the average length. To ninish the difference is recommendable to use a h extension of the source alphabet as Shannon howed in his First Theorem. The k-th extension of source is formed by grouping the symbols of the riginal source in blocks of size k. When k=2 we all them digrams, with k=3 trigrams and so on.

the second extension of a code the probabilities are given by the product $p_i{}^*p_j$ considering that hey are independent symbols and besides we need code M^k digrams (k=2).

The average length for the 2nd extension $\bar{L}_{2nd}=2.86$. To compare this length with the value the first extension we need to divide $\bar{L}_{2nd}$ by 2 to get $\bar{L}_{2nd/2}=1.43$ bits. As long as k tends to infinity the average length tends to the theoretical limit (entropy). In this case

$$H(x)=1.41 < \bar{L}_{2nd/2}=1.43 < \bar{L}=1.47 \qquad (10)$$

METASYMBOLS

Shannon suggested the use of words instead of individual symbols. In Huffman Coding the k-th extension of a coding allows to encode digrams, trigrams, etc., but assuming that symbols are independent of each other whereas in techniques of dictionary[4] it is natural to use them but they have the restriction of using adjacent symbols. In this work we intend to identify groups of not necessarily adjacent symbols (metasymbols) that, when optimally encoded offer a new approach to data compression. A metasymbol contains information about a set of symbols within a message. For example, given a message msg = "abcdefghijklm", its decomposition in metasymbols would be: M={abm, ef, d, ghijk, lc}.

Take the message "xyazxybzxyczxydz" of length L=16 characters. A possible Huffman Coding is shown in table 4.

TABLE 4.
HUFFMAN CODING FOR THE MESSAGE
"xyazxybzxyczxydz"

s_i	Probability	Huffman Coding	l_k (bits)
x	4/16	00	2
y	4/16	01	2
z	4/16	10	2
a	1/16	1100	4
b	1/16	1101	4
c	1/16	1110	4
d	1/16	1111	4

The Huffman Coded Message is 00'x'01'y'10'z'1100 'a'1101'b'1110'c'1111'd' 00 01 100 10 00 01 1101 10 00 01 1110 10 00 01 1111 10 which has a length of 118 bits.

With metasymbols we need to assign positions to the symbols as if they were an array of characters. Then the message looks like

"x y a z x y b z x y c z x y d z"
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

where it is possible to identify the pattern 'xyz' formed with non adjacent symbols at positions 0-1-3, 4-5-7, 8-9-11 and 12-13-15. Another metasymbol is 'abcd' with symbols at positions 2-6-10-14 as we show in figure 1.

```
"x y a z x y b z x y c z  x  y  d  z"
 0 1   3 4 5   7 8 9   11 12 13    15
     2       6      10          14
```

Fig.1 Symbols and its positions in the message "xyazxybzxyczxydz"

With these two metasymbols we create a "metadictionary" which codification could be

1: x1y2z
2: a4b4c4d

We read this as follows: Write 'x' at position 0 then advance 1 position and put 'y' then advance 2 positions and put 'z'.
Write 'a' then advance 4 positions and put 'b' then advance 4 position and put 'c' then advance 4 positions and put 'd'.

The coded message (without dictionary) can be written as the sequence of indices of the dictionary 12111

The decoding process is as follows:

1. Reading the coded message from left to right and using the dictionary we know that 1 represents X1y2z then we decode it as:

```
x y   z
0 1 2 3
```

Using the dictionary we know that 2 represents a4b4c4d then we decode it as

```
a         b         c          d
0 1 2 3 4 5 6 7 8 9 10 11 12
```

2. Read the first 1 from the encoded message to form the partially decoded message

```
x y   z
0 1 2 3
```

3. Search for the leftmost empty position (LEP) in the partially decoded message, LEP=2

4. Read the next value of 2 from the encoded message and at position LEP =2 we insert the symbol 'a' and in a kind of OR operation the other symbols 'b', 'c' and 'd' to get

```
x y a z   b       c         d
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Repeat steps 2 and 4 until we finish reading the encoded message. Here we show how the decoded message is reconstructed:

```
1. Read 1
2. LEP=4
3. x y a z x y b z       c         d
   0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1. Read 1
2. LEP= 8
3. x y a z x y b z x y c z         d
   0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1. Read 1
2. LEP= 12
3. x y a z x y b z x y c z x y d z
   0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

4.1 Compression with metasymbols

In the last example we encoded the message concatenating the indices of the metadictionary. However, it is possible to encode assigning Huffman Codes to each metasymbol. Therefore, if we have just 2 metasymbols the Huffman Codes are 0 and 1 and the Encoded Metadictionary looks as shown in table 5.

TABLE 5.
METADICTIONARY USING METASYMBOLS AND HUFFMAN CODING

Huffman Code	What represents?
0	'x'1'y'2'z'
1	'a'4'b'4'c'4'd'

and the encoded message is 01000. The number of bits used to code the message including the metadictionary can be calculated in comparison with Ec. (6) for First Order Huffman Coding. Now M is the number of different Metasymbols in the message to encode. Note that we need to include the offset between each symbol which constitutes the metasymbol. In the case of the first row of table 5 we need to encode 1 and 2 and for the second row we need to encode 4, 4 and 4. Hence, in the worst case we need $\log_2(4)$ bits and in general $\log_2(\mathrm{Max(Offset)})$.

Then, to encode the message with metasymbols we need L_{metas} bits, where

$$Lmetas = \sum_{k=1}^{M} l_k + (8 + \log_2(\mathrm{Max(Offset)})) * \sum_{k=1}^{M} lm_k + L * \bar{L} \qquad (11)$$

In our example, L_{metas}= 88 bits which is less than the 118 bits used in the classic Huffman Codification, i.e. in some cases is possible to get better compression ratios.

5. EXPERIMENTS

In table 6 we show some results we have gotten. Column 3 indicates the number of bits used by First Order Huffman Coding and Column 4 is the number of bits used with metasymbols.

TABLES 6.
RESULT OF CODING WITH METASYMBOLS

Table 6.1

Message "GGGGGAEDCBAEDCBAEDCBGGGGGGGGG AEDCBAEDCBAEDCBGGGGGGGGGAEDCBAE DCBAEDCBGGGG"			
Metasymbols M=3	No. Bits ASCII	No. Bits (Huffman Coding)	No. Bits (Metasy mbols)
G D1C1B A1E	72	250	175

Table 6.2

Message "AAAAABBBBBCCCCCDDDDDEEEEEFFFFF"			
Metasymbols M=2	No. Bits ASCII	No. Bits (Huffman Coding)	No. Bits (Metasy mbols)
C5D5E5F A5B	240	142	115

In Huffman encoding generally the symbols correspond to bytes (8 bits) but we don't know if this is optimum, with metasymbols we can define that the basic symbols are the bits 0 and 1. For instance, when we have M=256 metasymbols using 8 bits each one we get the special case of Huffman coding for bytes.

Another special case of encoding with metasymbols is the one where our metasymbols are formed for just adjacent individual symbols. There we have a scheme similar to the LZ[4] technique.

Consider now a bit stream [5] 0000000100000001 where we want to get compression using Huffman Coding at one bit level. Then the symbols are 0 and 1 and their probabilities are 14 and 2 respectively. The corresponding Huffman Codes are synthesized in table 7.

TABLE 7
HUFFMAN CODING AT LEVEL OF BIT FOR A BIT STREAM MESSAGE

Column1 Symbol	Column2 Probability	Column 3 Codeword	Column 4 l_k
S1=0	14	0	1
S1=1	2	1	1

As we have just 2 different symbols (M=2) then the Huffman codes are 0 and 1 which are the same as the source symbol and the encoded message is 0000000100000001 plus the dictionary, i.e. we have negative compression.

In [6] we show that the selection of groups is not an easy work because it is an NP problem so we propose to use Ec. (11) as a metric for symbol clustering where we use a Vasconcelos Genetic Algorithm [7] to search the patterns in a message.

6. CONCLUSIONS

Shannon information theory gives fundaments which allow us to build probabilistic compression techniques but its most important restriction is that it considers that symbols are independent. Then the theoretical limit of compression applies to messages with this characteristic. In practice the symbols of a message usually are not independent. For example in a Spanish text is clear that after a 'q' it is highly probable that a 'u' follows i.e, $p(u \cap q) \neq p(u)p(q)$.

Huffman Codes use a probabilistic model that is restricted to use symbols with more that one bit each (unless we use a k-th extension where k must tend to infinity). This is impossible given a finite length message. Besides, average Huffman code length tends to the theoretical limit which bounded by the entropy of the source but without considering the size of its associated dictionary.

Adaptive Dictionary Techniques provide mechanism to determine dynamically the size of "word" in a message through a search of the best string matching in a dictionary but it is restricted adjacent symbols.

When compressing with metasymbols our goal is identify a set of groups of symbols with underlying assumptions and the groups selected in order to get maximum compression including the size of the metadictionary.

Given the above we want to select the groups optimally in order to get maximum compression; this effect we propose to use Ec.(11) as a fitness function of a Genetic Algorithm as reported in [6].

7. REFERENCES

[1] Shannon, C.E., *A Mathematical Theory of Communication*, Bell Sys. Tech. *J. 27* (1948), 379-423, 623-656.

[2] Huffman, D. A., *A method for the construction of minimum-redundancy codes. Proc. Inst. Radio Eng. 40*, 9 (.), 1098–1101, Sept 1952.

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

[4] Ziv, J., and Lempel, A., *A Universal Algorithm for Sequential Data Compression. IEEE Trans. on Inf. Theory IT-23*,3 (May 1977), 337-343.

[5] Steven Prestwich, *A Hybrid Local Search Algorithm for Low Autocorrelation Binary Sequences*, Technical Report, Department of Computer Science, National University of Ireland at Cork.

[6] Kuri, A. Herrera O., *Metrics for Symbol Clustering from Pseudoergodic Information Source*, IEEE Proceedings ENC2003.

[7] Kuri, A., *A Comprehensive Approach to Genetic Algorithms in Optimization and Learning*, Editorial Politécnico, 1999.