CONDITIONAL LEMPEL-ZIV ENCODERS

Marcelo S. Pinho and Weiler A. Finamore

Abstract - In this work, the concept of conditional Lempel-Ziv encoder is presented. This concept is used to establish a variant of the Lempel-Ziv-Welch encoder (LZW). The practical performance of the new version is measured by using the *Canterbury Corpus* and it is compared to the performance of the LZW. The new version has shown an improvement around 10% over the LZW, when compressing the *Canterbury Corpus*.

Resumo - Este trabalho apresenta o conceito de codificador de Lempel-Ziv condicional. Este conceito é utilizado para estabelecer uma variante do codificador Lempel-Ziv-Welch (LZW). O desempenho desta nova versão é avaliado, aplicando este compressor no *Canterbury Corpus* e comparado com o desempenho do LZW. Esta nova versão apresenta um ganho de aproximadamente 10% sobre o LZW, quando aplicado no *Canterbury Corpus*.

Keywords: source coding, universal source coding, lossless data compression, string matching.

1. INTRODUCTION

In 1976 Lempel and Ziv proposed a complexity measure for finite sequences based on the string matching technique [1]. The results presented in that work showed that the string matching technique could be very useful to design universal source encoders. Using those concepts, Ziv and Lempel introduced two encoders, which are known as $LZ77$ and $LZ78$ [2, 3]. The practical importance of these encoders is due mainly to the relationship of their low complexity and good performance. Following those works, many variants were proposed, some of them widely used in practice [4].

Let $\mathcal{A}$ be a set of cardinality α and let $\mathcal{A}^\infty$ denote the set of all one-sided infinite sequences. Let p be a probability measure in $\mathcal{A}^\infty$. An information source is defined as a pair $(\mathcal{A}, p)$, where $\mathcal{A}$ is called the source alphabet. The semi-infinite sequence of random variables output by the source, drawn according to p, is denoted by $X_1^\infty = X_1 X_2 \ldots$.

Let $\mathcal{A}^*$ be the set of all finite sequences of symbol drawn from $\mathcal{A}$. A lossless encoder C is a reversible transformation which maps any finite sequence $u \in \mathcal{A}^*$ (also called word) into a finite sequence of bits $C(u)$ (also known as codeword).

Let $X_1^n = X_1 \ldots X_n$ be a finite random sequence obtained from X_1^∞ and let x_1^n denote a realization of X_1^n (with the convention that for $n = 0$, x_1^n denotes the empty sequence). Let $|u|$ denote the length of the sequence u, that is, the number of symbols in u. Let $\mathcal{A}^n$ be the set of all length n sequences.

Source coding theory aims at minimizing $E[|C(X_1^n)|]$, where E denotes expectation. A fundamental result of the source coding theory establishes that for any lossless encoder C, it is true that $E[|C(X_1^n)|] \geq H(X_1^n) \geq nH(\mathcal{A}, p)$, where $H(X_1^n)$, the entropy of the random sequence X_1^n, is defined as

$$H(X_1^n) = \sum_{x_1^n \in \mathcal{A}^n} -Pr[X_1^n = x_1^n] \log_2 Pr[X_1^n = x_1^n], \quad (1)$$

and $H(\mathcal{A}, p)$, the source entropy, is defined as

$$H(\mathcal{A}, p) = \lim_{n \to \infty} \frac{H(X_1^n)}{n}. \quad (2)$$

Many lossless encoders, Huffman Encoder and Arithmetic Encoder for instance, are such that $E[|C(X_1^n)|] \approx H(X_1^n)$ and the respective compression rate, $E[\frac{|C(X_1^n)|}{n}]$ (in bits per source symbol), converges to the source entropy, when n goes to infinity [5]. These are encoders designed for sources with an specific probability measure p which, whenever used to compress a source, with a different probability measure q, achieve perfromance that can be very poor. This mismatch poses strong limitations in practice. There is also the fact, that the source probability measure is just a model and it is unknown in many applications. Furthermore, there are many cases where an encoder is used to compress different kinds of data, drawn from different sources. A modem, for instance, uses the same encoder to compress images, text, programs, etc. The universal source coding theory is a branch of the source coding theory which addresses problems such as this. Its aim is to search for encoders which achieve good results for any source belonging to a given class of sources.

To find the best compression rate which can be attained by a lossless encoder, for a given class of sources, is a central question in universal source coding theory. An interesting result from this theory shows that it is possible to design lossless encoders that are optimal in the sense that their compression rates converge to the source entropy for many classes of sources. This result might lead us to the wrong assumption that there is no penalty in selecting a lossless universal encoder as opposed to using a lossless encoder whose design relies on the source probability measure. The problem with this assumption is that the optimality criterion which guarantees the convergence of the compression rate disregard the speed of convergence. The compression rate can converge to the source entropy, but if the convergency is too slow, the encoder will be useless, since in practice we deal with finite sequences. Therefore, a better criterion for an universal encoder is the asymptotic behavior of its compression rate with respect to a class of sources (instead of the limit of convergence – $H(\mathcal{A}, p)$). Using this criterion, it can be shown that

This work was supported by CNPq - Brazil. This work was presented in part at the Simpósio Brasileiro de Telecomunicações, Gramado-RS, Brazil, 2000.

Marcelo S. Pinho is with the Department of Electrical Engineering, UNESP, Guaratinguetá, SP, Brazil. Weiler A. Finamore is with the Center for Telecommunications Studies, PUC-Rio, RJ, Brazil.
E-mails: mpinho@feg.unesp.br, weiler@cetuc.puc-rio.br

there is no universal encoder which achieves the best performance ($\approx H(X_1^n)$) for any source in a given class of sources.

In [3] it was shown that for any ergodic source $(\mathcal{A}, p)$, the $LZ78$ compression rate, $\frac{|LZ78(X_1^n)|}{n}$, converges to $H(\mathcal{A}, p)$, almost surely. In [6] it was shown that for any stationary source $(\mathcal{A}, p)$, the $LZ77$ compression rate, $\frac{|LZ77(X_1^n)|}{n}$, converges to $H(\mathcal{A}, p)$, in probability. However, recently it was shown that the compression rates of these encoders do not converge as fast as it could, that is, there are universal encoders whose compression rates converge faster [7]. The disadvantage of these encoders, for which the compression rates converge faster, is their high computational complexity. Therefore, the $LZ78$ and the $LZ77$ are yet used in many applications.

The present work proposes a procedure to improve the performance of string match encoders. The rationale behind the proposition is briefly discussed next. Let C be a sequential encoder via string matching (e.g. the $LZ77$ or the $LZ78$). Given a realization x_1^n, C parses the sequence in $m = m_C(x_1^n)$ phrases $s_1 = x_1^{n_2}, s_2 = x_{1+n_2}^{n_3}, \ldots, s_m = x_{1+n_m}^{n_{1+m}}$, where each phrase $s_i = x_i^j$, $i < 1 + n_i$, that is, each phrase is a simple reproduction of a past string in the sequence. After that, the encoder C encodes each phrase s_i, using a dictionary of strings, based on the past. In general, the well known string matching encoders, such as $LZ77$ and $LZ78$, do not use the knowledge of the phrase s_{i-1} to encode the phrase s_i. In [8] it was developed an algorithm, based on the $LZ77$ which uses the phrase s_{i-1} to encode the string s_i. This encoder was called LZp and preliminary results have shown a good performance. In this work, it is shown a general description of the string matching encoders which use the knowledge of the phrase s_{i-1} to encode the phrase s_i, which is called here *Conditional Lempel-Ziv Encoders*. This technique is used to improve the performance of the LZW encoder - the most used version of the $LZ78$.

Section 2 presents the general model of a sequential string matching encoder and gives three examples, $LZ77$, $LZ78$, LZW. The general description of the Conditional Lempel-Ziv is given in Section 3. Section 4 is devoted to practical implementations and their results. Section 5 presents the conclusion.

2. SEQUENTIAL STRING MATCHING ENCODER

A sequential string matching encoder uses a dictionary of strings, derived from the part of sequence which was already encoded to encode the part of the sequence which remains to be coded. At the first step, the encoder establishes an initial dictionary which is used to compute the first phrase. The dictionary is then updated, at each step.

Figure 1 shows a generic step i of a sequential string matching encoder. The sequence $x_1^{n_i}$ represents the part of the sequence x_1^n which was encoded before step i. The part of sequence which remains to be coded is denoted by $x_{1+n_i}^n$. The box called "Dictionary" builds, using the sequence $x_1^{n_i}$, the updated dictionary D_i which is used in step i. The box "Next Phrase" establishes the phrase $s_i = x_{1+n_i}^{n_{1+i}}$ which is

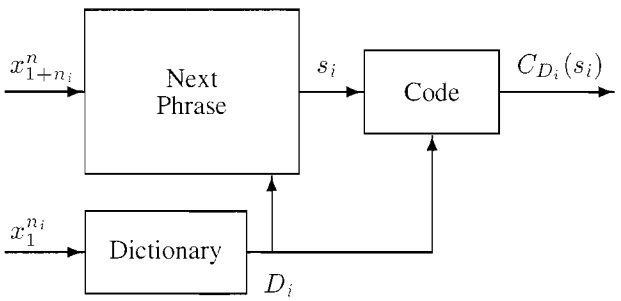

Figure 1. Sequential string matching encoders.

parsed off the sequence $x_{1+n_i}^n$ in step i. The box "Code" encode the phrase s_i into its codeword $C_{D_i}(s_i)$, using a map which is dependent of the dictionary D_i. In general, the code C_{D_i} is very simple and does not relies on any type of statistic measure. In fact, this is the simplicity that guarantees the low complexity of the encoder.

Following the general model, it is possible to define different sequential string matching encoders. An encoder is characterized when the four procedures below are specified.

1. Procedure to build the initial dictionary;
2. Procedure to build the dictionary D_i, which can either be an update procedure built upon D_{i-1} and $x_1^{n_{1+i}}$, or a simple procedure, that uses just the sequence $x_1^{n_{1+i}}$.
3. Procedure to find the next phrase s_i, using the dictionary D_i;
4. Procedure C_{D_i} to encode s_i.

The $LZ77$ and the $LZ78$ are examples of sequential string matching encoders. Therefore one can define these encoders just settling those four procedures. Several Lempel-Ziv variations are described next. In all cases the parameter n_1 (to be defined) is set equal to zero.

$LZ77$

1. Initial Dictionary: $D_1 = \mathcal{A}$.
2. Dictionary D_i: $D_i = \{uv : u = x_j^k, v \in \mathcal{A}, j \leq n_i, k \geq n_i - 1\}$.
3. Next Phrase: s_i is the longest prefix of $x_{1+n_i}^n$ which belongs to D_i.
4. Code C_{D_i}: Since $s_i \in D_i$, there is an integer $j \leq n_i$ such that $s_i = x_j^k v$, $v \in \mathcal{A}$. The phrase s_i is encoded by an integer code of $k - j + 1$, a simple code of j plus a simple code (fixed length) of v.

It is important to point out that, since $u = \emptyset$ when k assume the value $j - 1$ the alphabet $\mathcal{A}$ is a subset of the dictionary $D_i = \{uv : u = x_j^k, v \in \mathcal{A},\ j \leq n_i,\ k \geq n_i - 1\}$. Since the range of $k - j$ is $\{-1, 0, 1, \ldots\}$, any non negative integer code can be used to encode the integer $k-j+1$ (e.g. the Elias Code [9]). The range of j is $\{1, \ldots, i - 1\}$. Therefore j can be encoded using $\log_2(i - 1)$ bits. The symbol $v \in \mathcal{A}$ needs $\log_2 \alpha$ bits to be encoded. It is easy to see that in the first step,

the integers $k-j+1$ and j do not need to be encoded. In fact, it is easy to see that, since $D_1 = \mathcal{A}$, $s_1 = v$. Furthermore, if $k - j = -1$, then the integer j does not need to be encoded, since $u = \emptyset$, the empty sequence, and therefore $s_i = v$.

LZ78

1. Initial Dictionary: $D_1 = \mathcal{A}$.
2. Dictionary D_i: $D_i = D_{i-1} \cup \{s_{i-1}v : v \in \mathcal{A}\}$.
3. Next Phrase: s_i is the longest prefix of $x_{1+n_i}^n$ which belongs to D_i.
4. Code C_{D_i}: Since $s_i \in D_i$, there is an older phrase $s_j, j < i$ such that $s_i = s_j v$, $v \in \mathcal{A}$. The phrase s_i is encoded by a simple code of j plus a simple code (fixed length) of v.

The range of j is $\{0, \ldots, i-1\}$, therefore the integer j can be encoded using $\log_2 i$ bits. The symbol $v \in \mathcal{A}$ needs $\log_2 \alpha$ bits to be encoded. It is easy to see that in the first step, the integer j does not need to be encoded. In fact, since $D_1 = \mathcal{A}$, one has $s_1 = v$.

There are many variants of $LZ77$ and $LZ78$. One of the most famous is the variant of the $LZ78$ which was proposed in [10] and is called LZW. The LZW was designed to circumvent a weakness of the $LZ78$. To understand this point more explicitly, consider the description of the $LZ78$. It is easy to see that, at step i, there are many phrases in the dictionary D_i which never appeared in the past sequence $x_1^{n_i}$. Some of them, as a matter of fact, will rather likely, never appear again in the source sequence. In a text file, for example, when the letter q appear, the $LZ78$ includes α new phrases on D_i. Many of them, like qw for instance, will obviously not be part of an ordinary text. The LZW get rid of this problem, avoiding the insertion of this type of phrases. Although this is problem that can be neglected in the asymptotic analysis, in practice it can not. In fact, simulation results using the LZW have shown an improvement around 10% over the simulation results obtained with the $LZ78$. The description of the LZW is given bellow.

LZW

1. Initial Dictionary: $D_1 = \mathcal{A}$.
2. Dictionary D_i:
   $D_i = D_{i-1} \cup \{s_{i-1}v : v \text{ is the first symbol of } s_i\}$.
3. Next Phrase: s_i is the longest prefix of $x_{1+n_i}^n$ which belongs to D_i.
4. Code C_{D_i}: Each element of D_i has an index as follows. The elements of $\mathcal{A}$ is ordered from 0 to $\alpha - 1$. The index of $s_1 v$ is α, the index of $s_2 v$ is $\alpha + 1$ and so on. The phrase s_i is encoded by just encoding its index i with a simple integer code.

The range of the index of s_i in D_i is $\{0, \ldots, \alpha + i - 1\}$. Therefore the integer j can be encoded using $\log_2(\alpha + i)$ bits.

The following examples illustrates the work of the encoders. The source sequence to be compressed is considered to be $x_1^n = 010001101100$.

Example 1 *LZ77*

1. $D_1 = \{0, 1\}$,
   $s_1 = 0$, $k - j = -1$, $j = 0$, $v = 0$
   $C_{D_1}(s_1) = 0$.
2. $D_2 = D_1 \cup \{0v, 01v, \ldots, 010001101100v\}$
   $s_2 = 1$ $k - j = -1$, $j = 0$, $v = 1$
   $C_{D_2}(s_2) = 110, 1$.
3. $D_3 = D_2 \cup \{1v, 10v, \ldots, 10001101100v\}$
   $s_3 = 00$, $k - j = 0$, $j = 1$, $v = 0$
   $C_{D_3}(s_3) = 111, 0, 0$.
4. $D_4 = D_3 \cup \{00v, 000v, \ldots, 0001101100v\} \cup$
   $\cup\{00v, 001v, \ldots, 001101100v\}$
   $s_4 = 011$, $k - j = 1$, $j = 1$, $v = 1$
   $C_{D_4}(s_4) = 100110, 11, 1$.
5. $D_5 = D_4 \cup \{011v, \ldots. 01101100v\} \cup$
   $\cup\{11v, \ldots, 1101100v\} \cup$
   $\cup\{1v, \ldots, 101100v\}$,
   $s_5 = 01100$, $k - j = 3$, $j = 5$, $v = 0$
   $C_{D_5}(s_5) = 1011100, 100, 0$.

□

Example 2 *LZ78*

1. $D_1 = \{0, 1\}$,
   $s_1 = 0$,
   $C_{D_1}(s_1) = 0$.
2. $D_2 = D_1 \cup \{00, 01\}$
   $s_2 = 1$,
   $C_{D_2}(s_2) = 0, 1$.
3. $D_3 = D_2 \cup \{10, 11\}$
   $s_3 = 00$,
   $C_{D_3}(s_3) = 01, 0$.
4. $D_4 = D_3 \cup \{000, 001\}$
   $s_4 = 01$,
   $C_{D_4}(s_4) = 01, 1$.
5. $D_5 = D_4 \cup \{010, 011\}$
   $s_5 = 10$,
   $C_{D_5}(s_5) = 010, 0$.
6. $D_6 = D_5 \cup \{100, 101\}$
   $s_6 = 11$,
   $C_{D_6}(s_6) = 010, 1$.
7. $D_7 = D_6 \cup \{110, 111\}$
   $s_7 = 00$,
   $C_{D_7}(s_7) = 001, 0$.

□

Example 3 *LZW*

1. $D_1 = \{0, 1\}$,
   $s_1 = 0$,
   $C_{D_1}(s_1) = 0$.
2. $D_2 = D_1 \cup \{01\}$
   $s_2 = 1$,
   $C_{D_2}(s_2) = 01$.
3. $D_3 = D_2 \cup \{10\}$
   $s_3 = 0$,
   $C_{D_3}(s_3) = 00$.

4. $D_4 = D_3 \cup \{00\}$
   $s_4 = 00$,
   $C_{D_4}(s_4) = 100$.
5. $D_5 = D_4 \cup \{001\}$
   $s_5 = 1$,
   $C_{D_5}(s_5) = 001$.
6. $D_6 = D_5 \cup \{11\}$
   $s_6 = 10$,
   $C_{D_6}(s_6) = 011$.
7. $D_7 = D_6 \cup \{101\}$
   $s_7 = 11$,
   $C_{D_7}(s_7) = 110$.
8. $D_8 = D_7 \cup \{110\}$
   $s_8 = 00$,
   $C_{D_8}(s_8) = 0100$.

□

3. CONDITIONAL LEMPEL-ZIV

It becomes evident from the encoders description presented that the code C_{D_i} disregard the correlation tha may exist between the phrase s_{i-1} immediately preceding s_i and the phrase s_i. The phrase s_i is encoded as if it were independent of s_{i-1}, such as in a memoryless model. It is true that the dictionary structure somehow build a context model for each symbol of s_i, as pointed out by Langdon [11]. A close examination of this structure reveals however that the initial portion of s_i is modelled with a short context, which can be poor for the source model. An example of this poor choice is the context of the first symbol of s_i, x_{1+n_i} — the context is the empty sequence. Since the length of s_i grows indefinitely, the influence of this bad effect vanishes in the limit when n goes to infinity. When dealing with finite sequences this problem can be relevant yielding to significant performance degradation.

In [8] it was shown a new variant of the $LZ77$, called LZp, which uses the phrase s_{i-1} to build the code C_{D_i}. The LZp fix an order k (related to the context length) and the phrase s_i is such that $\mathcal{L}_k(s_{i-1})s_i$ is the longest prefix of $\mathcal{L}_k(s_{i-1})x_{1+n_i}^n$ which belongs to D_i, where $\mathcal{L}_k(s_{i-1})$ are the k last symbols of s_{i-1}. The code of s_i is based on a subset $D_i(\mathcal{L}_k(s_{i-1}))$ of D_i, which is the set of all sequences in D_i which starts with $\mathcal{L}_k(s_{i-1})$. Practical results have shown a good improvement. In fact, the result of the LZp for the Canterbury Corpus is one of the best in the literature, as reported in [8].

Although the idea of conditional Lempel-Ziv has been introduced in [8] as a variant of the $LZ77$, the straightforward application of this idea to the $LZ78$ will not be efficient [12]. The problem resides with the dictionary structure. Notice that the phrase s_i produced by LZp tends to be shorter than the equivalent phrase produced by the original $LZ77$. The phrase s_i is encoded using just a subset of D_i, and for this reason the corresponding codeword will also be smaller. Furthermore, the dictionary update procedure will not be affected by this modification.

The analysis of a straightforward conditional-$LZ78$ leads to a different picture. As with the LZp, a straightforward conditional-$LZ78$ will tend to reduce the phrase s_i and its corresponding codeword. However, in this case, the update of D_i will be affected. This statement can be understood by noticing the fact, that the $LZ78$ dictionary is updated by appending the phrases $s_i v$, $v \in \mathcal{A}$ and that if s_i is reduced, the dictionary will be updated with reduced phrases. Even though the conditional technique yield an improvement in the performance at step i, the modification in the update procedure will tend to reduce the performance of the next steps. Therefore, a straightforward conditional-$LZ78$ will not be effective.

To avoid this difficulty, by setting the value of the order k to a value that maximizes the length of s_i which turns out to be the same procedure used in a version of $LZ78$, [13] called multi-match Lempel-Ziv, dubbed $mmLZ$.

The version $mmLZ$ was proposed to improve the parsing procedure of LZW. In fact, from the description of the $LZ78$ (and also of the LZW) it is easy to see that the codeword length $C(x_1^n)$ increases with the number m of parsed phrases. Therefore, if exist a procedure that can reduces m, then the performance of the encoder will be improved. In [14] it was shown that the procedure which finds the optimal parsing (with lowest number of phrases) for the $LZ78$ is NP-complete, a result which also holds for the LZW. In [15] it is presented a procedure to compute the optimal parsing, when the dictionary of phrases is fixed — which is not the case of the $LZ78$ and LZW encoders. The $mmLZ$ is briefly described next.

At each step i, the $mmLZ$ searches the pair (s_i, s_{i+1}) in $D_i \times D_{1+i}$ which is the longest prefix of $x_{1+n_i}^n$. Notice that s_i is not anymore, necessarily, the longest prefix of $x_{1+n_i}^n$ which belongs to D_i. In fact, sometimes it is interesting to reduce the phrase s_i to find a longer s_{1+i} such that the pair $s_i s_{1+i}$ is better than the pair established by the original encoder. However, it is obvious that the gain of $mmLZ$ is not obtained by reducing s_i, but by enlarging s_{1+i}. This enlargement is achieved by starting the phrase s_{1+i} in a symbol x_j different from $x_{1+n_{i+1}}$, $j \leq 1 + n_{i+1}$. The idea of conditional Lempel-Ziv can therefore, be used to improve the $mmLZ$. Let s_i be kept as in the original encoder and the let s_{1+i} be such that $s_i s_{1+i}$ can be written as a concatenation of a pair $(\mathcal{F}_j(s_i), \mathcal{L}_k(s_i)s_{1+i}) \in D_i \times D_i$, where $\mathcal{F}_j(s_i)$ is the first symbols of s_i such that $\mathcal{F}_j(s_i)\mathcal{L}_k(s_i) = s_i$. Thus the phrase s_{1+i} can be encoded using a subset of D_{1+i} which is composed by all phrases which starts with $\mathcal{L}_k(s_i)$.

It is thus possible to establish new versions of the $LZ78$ by using the concept of conditional Lempel-Ziv following the same ideas outlined in the above discussion. The version called $cLZW$, built upon the LZW, with an illustrative example are presented next.

$cLZW$

1. Initial Dictionary $D_1 = \mathcal{A}$.
2. Next Phrase: $s_1 = x_1$. For $i > 1$, s_i is such that $s_{i-1}s_i$ is the longest prefix of $x_{1+n_{i-1}}^n$ which can be written as a concatenation of two phrases in D_i, that is, $s_{i-1}s_i = s'_{i-1}s'_i$, where $(s'_{i-1}, s'_i) \in D_i \times D_i$.
3. Dictionary D_i:
   $D_{1+i} = D_i \cup \{s'_i v : v \text{ is the first symbol of } s_{i+1}\}$.

4. Code C_{D_i}: From the NEXT Pharse procedure it can be seen that s_i is a sufix of s'_i. Let u be the prefix of s'_i such that $s'_i = us_i$ (u can be the empty sequence). The phrase s_i is encoded by the length of u, followed by the index of s'_i in a subset $D_i(u)$ of D_i such that $D_i(u) = \{y \in D_i : u \text{ is a strict prefix of } y\}$.

In the description of the $cLZW^*$, the Next Phrase procedure is placed before the Dictionary Update procedure to make the exposition easier. The range of the length $|u|$ of u is less than $|s'_{i-1}|$. Therefore it can be encoded using $\lceil \log_2 |s'_{i-1}| \rceil$ bits. Each subset $D_i(u)$ of D_i, for any $u \in D_i$ must be ordered and the index of s'_i can be encoded using $\lceil \log_2 |D_i(u)| \rceil$ bits. To simplify the implementation of the encoder, the ordering of $D_i(u)$ can be established by using the ordering of D_i.

Example 4 $cLZW^*$

Let $x_1^n = 010001101100$. *Therefore the* $cLZW^*$ *encode* x_1^n *as follows.*

1. $D_1 = \{0, 1\}$,
   $s_1 = 0$,
   $C_{D_1}(s_1) = 0$.
2. $D_2 = D_1 \cup \{01\}$
   $s'_1 = 0$, $s'_2 = 1$ *and* $s_2 = 1$,
   $C_{D_2}(s_2) = 01$.
3. $D_3 = D_2 \cup \{10\}$
   $s'_2 = 1$, $s'_3 = 0$ *and* $s_3 = 0$,
   $C_{D_3}(s_3) = 00$.
4. $D_4 = D_3 \cup \{00\}$
   $s'_3 = 0$, $s'_4 = 00$ *and* $s_4 = 00$,
   $C_{D_4}(s_4) = 100$.
5. $D_5 = D_4 \cup \{001\}$
   $s'_4 = 0$, $s'_5 = 01$ *and* $s_5 = 1$,
   $D_5(0) = \{01.00, 001\}$
   $C_{D_5}(s_5) = 1, 00$.
6. $D_6 = D_5 \cup \{011\}$
   $s'_5 = 1$, $s'_6 = 10$ *and* $s_6 = 10$
   $C_{D_6}(s_6) = 011$.
7. $D_7 = D_6 \cup \{101\}$
   $s'_6 = 1$, $s'_7 = 011$ *and* $s_7 = 11$,
   $D_7(0) = \{01, 00, 001, 011\}$
   $C_{D_7}(s_7) = 1, 11$.
8. $D_8 = D_7 \cup \{0110\}$
   $s'_7 = 11$, $s'_8 = 00$ *and* $s_8 = 00$,
   $C_{D_8}(s_8) = 0, 0100$.

□

It can be seen from the description of the encoders that the length of the phrase s_i tends to grow with i. Therefore, when the length n of the input sequence x_1^n is large, the length of the phrases s_i can be large. The length of s_i is an important parameter of the encoder because the $cLZW^*$ computational complexity depends of it. In fact, the procedure to find the best pair (s'_{i-1}, s'_i) is done by finding the longest prefix s_{i-1} of $x_{1+n_{i-1}}^n$ followed by the next longest prefix s''_i. Once this task is completed, the phrase s_{i-1} is reduced, by suppressing from it the last symbol. A search for a new longest prefix is then initiated. This procedure is repeated until the length of the phrase s_{i-1} is equal to one. Then, the encoder compares the length of all pairs of phrases and chooses the longest pair. Therefore, the complexity of the step i of $cLZW^*$ grows with the length of s_{i-1}. This complexity is not of much concern since the LZW^* complexity is very low, yet a small modification can be introduced in this encoder to bring the $cLZW$ complexity even closer to the complexity of the LZW.

As pointed out in the $cLZW$ description, the s'_i can be written as $s'_i = us_i$. It can be see that the phrase u is a sufix of s_{i-1} and it is used as a context to encode the phrase s_i. As discussed before, the complexity of $cLZW$ depends of the length of s_{i-1}, since the length of u must be the best one among $\{0, 1, \ldots, |s_{i-1}| - 1\}$ and the search is exhaustively done. Therefore, if the length of u is bounded, the complexity of the encoder will be reduced. This work denotes by $cLZW/o$ the encoder that bounds the length of u such that $|u| \leq o$. The description of this encoder is similar to the $cLZW^*$ and will be omitted.

4. RESULTS

In [16] the problem of establishing a good test set for a lossless encoder is discussed and a new test set is introduced. This new test set, called Canterbury Corpus, became popular and it has been used to measure the practical performance of many lossless encoders. The $cLZW$ was implemented and its practical performance was measured, using the Canterbury Corpus. Table 1 presents the compression rate (in bits/symbol) obtained by the LZW^* and by the $cLZW$ when compressing the *Canterbury Corpus*.

Files - x_1^n	$\lvert x_1^n \rvert$	$\frac{\lvert LZW^*(x_1^n) \rvert}{n}$	$\frac{\lvert cLZW^*(x_1^n) \rvert}{n}$
alice29.txt	152089	3.26	2.91
asyoulik.txt	125179	3.51	3.20
cp.html	24603	3.68	3.24
fields.c	11150	3.56	2.99
grammar.lsp	3721	3.89	3.40
kennedy.xls	1029744	2.36	2.46
lcet10.txt	426754	3.02	2.64
plrabn12.txt	481861	3.27	3.03
ptt5	513216	0.97	0.97
sum	38240	4.20	3.72
xargs.1	4227	4.42	4.02
Average		3.29	2.96

Table 1. $cLZW^*$ Results

As pointed out in Section 3, the complexity of the $cLZW$ can be reduced, by bounding the length of the context. The encoder which bounds the context length is called $cLZW/o$ where o is the maximum length of the context. This encoder was implemented and was also used to compress the *Canterbury Corpus*. The results obtained are displayed in Table 2.

The results presented in Table 1 show that in the average, the $cLZW^*$ performance is 10% better than the LZW performance. Furthermore, from Table 2 it can be seen that the $cLZW^*/o$ achieves a performance very close to the $cLZW$ performance. Therefore, the penalty due to introduction of a bound for the context length can be neglected. This point is very interesting because it shows that the improvement of 10%, obtained by the $cLZW^*$, can be achieved using a version

Arquivos - x_1^n	$\lvert x_1^n \rvert$	$\frac{\lvert cLZW/o(x_1^n) \rvert}{n}$	
		$o=2$	$o=3$
alice29.txt	152089	2.94	2.92
asyoulik.txt	125179	3.21	3.20
cp.html	24603	3.27	3.26
fields.c	11150	3.05	3.02
grammar.lsp	3721	3.47	3.43
kennedy.xls	1029744	2.44	2.46
lcet10.txt	426754	2.68	2.66
plrabn12.txt	481861	3.05	3.03
ptt5	513216	0.97	0.97
sum	38240	3.74	3.73
xargs.1	4227	4.05	4.04
Average		2.99	2.97

Table 2. $cLZW/o$ Results

whose complexity is very close to the LZW.

5. CONCLUSION

This work presented the concept of conditional Lempel-Ziv encoders. It showed how this technique can be used to improve the performance of the encoders based on string matching. A new version of the $LZ78$, called $cLZW$, was presented and its practical performance was measured by testing the encoder with *Canterbury Corpus*. Practical results have shown an improvement around 10% over the well known LZW. Since the complexity of the $cLZW$ is greater than the LZW complexity, it was modified to keep the complexity near to the LZW complexity. The modified version, in short $cLZW/o$, keeps a maximum context length of o. It was shown that the performance of the $cLZW/o$ approximates the performance of the $cLZW$ for $o = 2$. Therefore the improvement of 10% can be obtained by using a variant of the $LZ78$ whose complexity is near to the LZW complexity.

REFERENCES

[1] A. Lempel and J. Ziv, "On the complexity of finite sequences," *IEEE Transactions on Information Theory*, vol. 22, pp. 75–81, 1976.

[2] J. Ziv and A. Lempel, "A universal algorithm for data compression," *IEEE Transactions on Information Theory*, vol. 23, pp. 337–343, 1977.

[3] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Transactions on Information Theory*, vol. 24, pp. 530–536, 1978.

[4] D. Hankerson, G. A. Harris, and P. D. J. Jr., *Introduction to Information Theory and Data compression*. Florida: CRC Press, 1998.

[5] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. New York: Wiley, 1991.

[6] A. D. Wyner and J. Ziv, "The sliding-window lempel-ziv algorithm is asymptotically optimal," *Proc. IEEE*, vol. 82, pp. 872–877, 1994.

[7] F. M. J. Willems, Y. M. Shtarkov, and T. J. Tjalkens, "The context-tree weighting method: basic properties," *IEEE Transactions on Information Theory*, vol. 41, pp. 653–664, 1995.

[8] C. Bloom, "Lzp: a new data compression algorithm," *Proc. of IEEE Data Compress Conference*, 1996.

[9] P. Elias, "Universal codeword sets and representations of the integers," *IEEE Transactions on Information Theory*, vol. 21, pp. 194–203, 1975.

[10] T. A. Welch, "A technique for high-performance data compression," *IEEE Computer*, vol. 17, pp. 8–19, 1984.

[11] G. G. Langdon, "A note on the ziv-lempel model for compressing individual sequences," *IEEE Transactions on Information Theory*, vol. 29, pp. 284–287, 1983.

[12] M. S. Pinho, *Codificadores Universais via Recorrência de Padrões para Fontes com Número de Estados Finitos*. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Elétrica, 1996.

[13] M. S. Pinho, W. A. Finamore, and W. A. Pearlman, "Fast multi-match lempel-ziv," *Proc. IEEE Data Compress Conference*, p. 545, 1999.

[14] S. D. Agostino and J. Storer, "On-line versus off-line computation in dynamic text compression," *Information Processing Letters*, vol. 59, pp. 169–174, 1996.

[15] A. Hartman and M. Rodeh, "Optimal parsing of strings," *Combinatorial Algorithms on Words, Springer-Verlag, A. Apostolico and Z. Galil, editors,*, pp. 155–167, 1985.

[16] R. Arnold and T. Bell, "A corpus for the evaluation of lossless compression algorithms," *Proc. of IEEE Data Compress Conference*, pp. 201–210, 1997.

Marcelo S. Pinho received the B.S., M.S. and the D.S. degrees in Electrical Engineering from Pontifícia Universidade Católica do Rio de Janeiro, in 1994, 1996 and 2000, respectively. In 1998, he was a Visiting Scholar in the Image Processing Lab of the Rensselaer *Polytechnic Institute, NY - USA. He is currently a faculty member* in the Department of Electrical Engineering of Universidade Estadual Paulista at Guaratinguetá. His research interests are on information theory, digital signal processing, and digital transmission.

Weiler A. Finamore was born in Cisneiros, Minas Gerais, on October 16, 1944. He graduated in Electrical Engineering at Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) in 1969, and got his MS and PhD degrees in Electrical Engineering from the University of Winsconsin, USA, in 1974 and 1978, respectively. He was a faculty member in the Department of Electrical Engineering of the Universidade Federal do Pará from 1970 to 1973 and during the 1978-79 academic year. Since 1979 he has been a faculty member in the Department of Electrical Engineering of PUC-Rio, working at its Center for Telecommunications Studies (CETUC). His research interests are on coding and information theory, digital image processing, and digital transmission.