Codificador Lempel-Ziv'78 com Predição

Marcelo S. Pinho & Weiler A. Finamore
UNESP PUC-Rio
E-mail: mpinho@feg.unesp.br E-mail: weiler@cetuc.puc-rio.br

Resumo— **Este trabalho apresenta uma forma de utilizar a predição em versões do codificador Lempel-Ziv'78 (LZ78). Utilizando esta nova técnica sobre uma variação do** $LZ78$**, proposta recentemente, denominada** $ppo - LZW$**, este trabalho propõe uma nova versão,** $ppo - LZWp$**. Esta nova versão é testada através do** *conjunto de Canterbury* **e seus resultados são comparados com os resultados de versões anteriores. Uma variação do** $ppo - LZWp$**, que possui complexidade computacional comparável com a complexidade da versão** LZW**, também é proposta. Esta variação é aplicada na compressão do** *conjunto de Canterbury* **e os resultados obtidos são comparados com os resultados de versões anteriores.**

Palavras-chave— **Codificação de fonte, compressão de dados, codificação universal de fonte, codificadores via recorrência de padrões.**

I. Introdução

Seja $x_i^j = x_i, x_{1+i}, \ldots, x_j$ uma seqüência de símbolos pertencentes a um conjunto finito $\mathcal{A}$, com cardinalidade α. Seja $\mathcal{A}^*$ o conjunto de todas as seqüências finitas formadas por símbolos de $\mathcal{A}$; $\mathcal{A}^\infty$ o conjunto de todas as seqüências inifinitas formadas por símbolos de $\mathcal{A}$; e $\mathcal{A}^n$ o conjunto de todas as seqüências formadas por n símbolos de $\mathcal{A}$. Seja ainda p uma medida de probabilidade em $\mathcal{A}^\infty$ e X_1^n uma variável aleatória definida a partir da medida p. A probabilidade de $X_1^n = x_1^n$, medida através de p, é representada por $Pr[X_1^n = x_1^n]$. Uma fonte de informação seqüencial é definida por um par $(\mathcal{A}, p)$, onde $\mathcal{A}$ é denominado alfabeto da fonte.

Um codificador sem perda (de informação) é definido como sendo um mapa biunívoco $C : \mathcal{A}^* \rightarrow \mathcal{B}^*$, onde $\mathcal{B}$ é o alfabeto de saída do codificador. Assim sendo, para cada seqüência $u \in \mathcal{A}^*$, o codificador associa um elemento distinto $C(u) \in \mathcal{B}^*$. Este trabalho assume que $\mathcal{B} = \{0, 1\}$, sem qualquer perda de generalidade.

Se u é uma seqüência de símbolos de algum alfabeto, o comprimento de u, representado por $|u|$, é definido como sendo o número de símbolos de u, i.e., $|x_1^n| = n$. O problema básico da teoria da codificação é minimizar o valor esperado do comprimento da palavra código, i.e., $E[|C(X_1^n)|]$ para uma fonte $(\mathcal{A}, p)$ dada. Um resultado fundamental da teoria da codificação de fonte diz que para todo o codificador sem perda $E[|C(x_1^n)|] \geq H(X_1^n) \geq nH(\mathcal{A}, p)$, onde $H(X_1^n)$ é a entropia de X_1^n definida por

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

e $H(\mathcal{A}, p)$ é a entropia da fonte, definida por

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

Existem vários codificadores, tais como o codificador de Huffman e o codificador aritmético, para os quais $E[|C(x_1^n)|] \approx H(X_1^n)$ e cujas taxas de compressão (em bits por símbolo da fonte) $E[\frac{|C(x_1^n)|}{n}]$ convergem para a entropia da fonte $H(\mathcal{A}, p)$, quando $n \rightarrow \infty$. No entanto, tais codificadores são projetados para uma fonte com uma medida de probabilidade específica. Quando tais codificadores são utilizados para comprimir uma fonte com uma medida de probabilidade $q \neq p$, em geral, os resultados são ruins. Esta é uma forte limitação na prática. De fato, a medida de probabilidade de uma fonte é apenas um modelo matemático e é desconhecida em muitas aplicações. Além disso, existem certas aplicações onde um codificador é utilizado para codificar diferentes tipos de dados. Por exemplo, um modem utiliza o mesmo codificador para comprimir imagens digitalizadas, arquivos de texto e etc. A teoria da codificação universal é uma subárea da teoria da codificação que tem como objetivo desenvolver códigos que sejam bons para todas as fontes pertencentes a uma determinada classe.

A utilização da técnica de recorrência de padrões ("string matching") na compressão de dados teve sua origem em [6]. A partir dos resultados obtidos em [6], foram desenvolvidos diferentes codificadores universais, entre eles o $LZ78$, proposto em [13]. Devido à

Este trabalho foi desenvolvido com o apoio do CNPq. Parte deste trabalho foi desenvolvido no Laboratório de Processamento de Sinais da UFRJ (LPS/UFRJ).

sua baixa complexidade computacional e ao seu bom desempenho, este codificador se tornou extremamente popular. Após a publicação do artigo original [13], surgiram diversas variações deste codificador e algumas destas versões se tornaram padrões em diferentes aplicações (por exemplo o padrão V.42bis da UIT). Uma das versões mais populares do $LZ78$ é o codificador LZW proposto em [11].

Em [13] foi mostrado que para toda fonte ergódica, $\frac{|LZ78(x_1^n)|}{n}$ converge com probabilidade 1 para $H(\mathcal{A}, p)$. É possível mostrar que este resultado também é válido para o LZW (vide [8]). No entanto, o fato da taxa de compressão convergir para a entropia da fonte não garante um bom desempenho na compressão de seqüências finitas (que é o caso real, pois na prática as seqüências são finitas). De fato, se a taxa de compressão convergir lentamente, os resultados serão ruins. Portanto, é fundamental analisar a velocidade de convergência. Recentemente foi mostrado que o $LZ78$ e que o LZW não são ótimos, no sentido de que suas taxas não convergem da forma mais rápida possível [7], [10], nem mesmo para a classe de fontes sem memória. Assim sendo, tais codificadores podem ser melhorados.

Dada uma realização x_1^n da fonte, o codificador $LZ78$ particiona esta seqüência em $m_{LZ78}(x_1^n)$ blocos $s_1, s_2, \ldots, s_{m_{LZ78}(x_1^n)}$, onde cada bloco s_i é a concatenação de um bloco anterior s_j, $j < i$ com um elemento $v \in \mathcal{A}$, i.e., $s_i = s_j, v$. Após particionar a seqüência, o $LZ78$ codifica cada bloco s_i através de um ponteiro para o bloco s_j e do símbolo v, também conhecido como símbolo de inovação.

Um problema existente no $LZ78$, é que este codificador utiliza $\log_2 \alpha$ bits para codificar o símbolo de inovação v. Assim sendo, para cada bloco s_i, existe um símbolo que é codificado como se todos os símbolos de $\mathcal{A}$ fossem equiprováveis. É fácil ver que em geral, esta estratégia é ruim. Para resolver este problema, em [11], foi proposta uma nova versão do $LZ78$ que modifica a forma de particionar a seqüência, de tal forma que a cada passo, o símbolo de inovação v não precisa ser codificado. Esta nova versão, conhecida como LZW, mostrou um desempenho superior em aproximadamente 10%, quando comparada com o $LZ78$.

O desempenho do codificador LZW está diretamente relacionado com o número de blocos que o LZW particiona a seqüência x_1^n, representado por $m_{LZW}(x_1^n)$. De fato, quanto menor o número de blocos, melhor o desempenho do codificador. No entanto, conforme apontado em [2], a partição do LZW não é a melhor possível. Na verdade, em [2], é mostrado que o problema de encontrar a partição ótima para o LZW é NP-completo. Em [2] também é mostrado que o problema da partição ótima para o $LZ78$ é NP-completo. Em [4], é apresentada uma forma ótima de particionar uma seqüência x_1^n em blocos pertencentes a um dicionário $D \subset \mathcal{A}^*$. Além disso, em [4], são sugeridas diversas formas de aplicar este método no $LZ78$. Em [9], o método de [4] é utilizado para desenvolver uma nova versão do LZW. Esta nova versão, denominada $f-mmlz$ em [9], supera o desempenho do lzw em aproximadamente 5%. Neste trabalho, o codificador $f - mmlz$ será representado por $ppo - LZW$, i.e., *partição pseudo-ótima LZW*.

Nas versões $LZ78$, LZW e $ppo-LZW$, a cada passo i, o bloco s_i é codificado através de algum método que não depende do bloco s_{i-1}. Este trabalho mostra que o bloco s_{i-1} pode ser utilizado para melhorar o código do bloco s_i. Isto dá origem a novas versões do $LZ78$. Este trabalho apresenta a versão baseada no $ppo - LZW$, denominada $ppo - LZWp$, i.e., *partição pseudo-ótima LZW com predição entre blocos.*

Este trabalho é dividido da seguinte forma. A seção 2 apresenta um modelo geral para os códigos seqüenciais via recorrência de padrões, e utiliza este modelo para descrever os códigos $LZ78$, LZW e $ppo - LZW$. O codificador proposto por este trabalho, $ppo - LZWp$, é introduzido na seção 3, onde também é apresentada uma variação deste novo codificador. Os resultados obtidos são apresentados na Seção 4. Encerrando o trabalho, a Seção 5 apresenta as conclusões.

II. Codificadores Seqüenciais via Recorrência de Padrões

O princípio de um codificador seqüêncial via recorrência de padrões é utilizar um dicionário de padrões, criado a partir de uma parcela da seqüência que já foi codificada, para codificar a parcela que falta ser codificada. No início do processo, existe um dicionário inicial e a cada parcela da seqüência que é codificada, o dicionário é atualizado.

A Figura 1 ilustra um passo genérico i, de um codificador seqüencial que utiliza a técnica de recorrência de padrões. Nela, a sequência $x_1^{n_i}$ representa a parte da sequência que foi codificada antes do passo i. A sequência $x_{1+n_i}^n$ representa a parte da sequência que ainda falta ser codificada. O bloco "Dicionário" constrói o dicionário que será utilizado no passo i, utilizando a sequência $x_1^{n_i}$. O bloco "Encontra próximo bloco" encontra o bloco de símbolos $s_i = x_{1+n_i}^{n_{1+i}}$ que

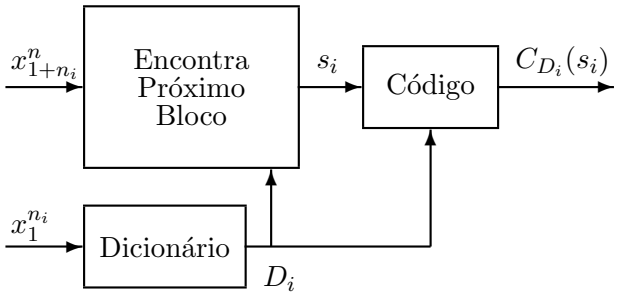

Fig. 1
Codificadores Seqüênciais via Recorrência de Padrões.

será codificado no passo i. O bloco "Código" codifica o bloco s_i, gerando a palavra código $C_{D_i}(s_i)$. Em geral, este código é bem simples, baseado apenas em uma indexação dos blocos pertencentes ao dicionário, e não utiliza diretamente uma medida estatística dos símbolos emitidos pela fonte.

Seguindo o modelo geral, apresentado na Figura 1, é possível descrever qualquer codificador seqüêncial via recorrência de padrões. Para isso, basta estabelecer os seguintes parâmetros.

1. Dicionário inicial;
2. Método para encontrar o próximo bloco, a partir do dicionário D_i;
3. Método para contruir o dicionário;
4. Código para os blocos do dicionário.

O codificador $LZ78$, proposto em [13], é um codificador seqüêncial que utiliza a técnica de recorrência de padrões. Ele se tornou famoso devido ao seu bom desempenho e a sua baixa complexidade computacional. Seguindo a descrição geral, é possível descrever o $LZ78$ da seguinte forma.

$LZ78$

1. Dicionário inicial: $D_1 = \emptyset$.
2. Método para encontrar o próximo bloco - s_i é o menor prefixo de $x_{1+n_i}^n$ que não pertence ao dicionário D_i, i.e., $s_i = dv$, onde $d \in D_i$ e $v \in \mathcal{A}$.
3. O dicionário D_{1+i} é construído a partir do dicionário D_i e do bloco s_i, tal que $D_{1+i} = D_i \cup \{s_i u : u \in A\}$.
4. Código - codifica o bloco $s_i = dv$, através da posição de d no dicionário, e do símbolo v.

Através da descrição do $LZ78$, é possível obsevar que todo bloco s_i é tal que existe s_j, $j < i$, para o qual $s_i = s_j v$, $v \in \mathcal{A}$. Este símbolo v, denominado símbolo de inovação, é codificado através de $\log_2 \alpha$ bits. Portanto, se a fonte emite símbolos com uma medida não uniforme, o símbolo v estará sendo codificado de forma ruim. Para resolver este problema, o codificador LZW foi proposto em [11]. No LZW, o bloco s_i é o maior prefixo que está no dicionário. Portanto não existe um símbolo v que precisa ser codificado em separado. No entanto, retirando o símbolo v, é preciso alterar a atualização do dicionário, pois neste caso $s_i \in D_i$. Portanto, no LZW a atualização do dicionário é realizada, incluindo o bloco s_i seguido do próximo símbolo. Na verdade, o LZW utiliza o primeiro símbolo do próximo bloco como símbolo de inovação. A descrição do LZW, seguindo o modelo geral, é dada a seguir.

LZW

1. Dicionário inicial - A;
2. Método para encontrar o próximo bloco - s_i é o maior prefixo de $x_{1+n_i}^n$ que pertence ao dicionário D_i;
3. O dicionário D_{1+i} é obtido incluindo o menor prefixo de $x_{1+n_i}^n$, que não pertence ao dicionário D_i, i.e. $D_{1+i} = D_i \cup \{x_{1+n_i}^{1+n_{1+i}}\}$;
4. Código - codifica o bloco s_i, através da sua posição no dicionário.

Através da descrição do LZW, é possível observar que a cada passo i, este codificador utiliza $\lceil \log_2 |D_i| \rceil$ bits para codificar o bloco s_i. Portanto, a cada passo, o LZW utiliza um código que mapeia blocos de comprimento variável ($\in D_i$) em blocos de comprimento fixo ($\lceil \log_2 |D_i| \rceil$). O objetivo de um codificador é minimizar a taxa do código (dada em bits por símbolo). Assim sendo, como a taxa do código C_{D_i} é dada por $\frac{\lceil \log_2 |D_i| \rceil}{|s_i|}$, minimizar a taxa do código C_{D_i} é equivalente a maximizar o comprimento do bloco s_i. Por esta razão, o LZW encontra o bloco s_i calculando o maior prefixo de $x_{1+n_i}^n$ em D_i. No entanto, esta maximização (do comprimento de s_i), como é realizada passo a passo, é uma maximização local e não garante um ótimo global. De fato, é possível mostrar que em certos casos, a redução do bloco s_i pode fazer com que o codificador encontre uma nova dupla (s'_i, s'_{1+i}) tal que $|s'_i| + |s'_{1+i}| > |s_i| + |s_{1+i}|$. Portanto a técnica de maximizar $|s_i|$ passo a passo não é a melhor forma de particionar a seqüência.

Seja $m_{LZW}(x_1^n)$ o número de blocos gerados pelo LZW, quando este codificador é aplicado na seqüência x_1^n. Então o comprimento da palavra código

$LZW(x_1^n)$ é dado por

$$|LZW(x_1^n)| = \sum_{i=1}^{m_{LZW}(x_1^n)} \lceil \log_2 |D_i| \rceil.$$

Portanto, a partição ótima, que minimiza $|LZW(x_1^n)|$, é dada pela partição que produz o menor número de blocos. O exemplo dado a seguir mostra que a técnica de partição utilizada pelo LZW não é ótima para qualquer seqüência x_1^n

Exemplo 1: Seja a seqüência $x_1^n = 000100011$, então o LZW particiona x_1^n em 7 blocos 0, 00, 1, 00, 0, 1, 1. No entanto, é possível observar que a seqüência x_1^n pode ser particionada de forma mais eficiente. De fato, x_1^n pode ser particionado em 6 frases, 0, 00, 1, 0, 001, 1.

Em [2] é mostrado que o problema de otimização da partição do LZW é NP-completo. Em [4] é apresentado um método para otimizar a partição de uma seqüência x_1^n em blocos pertencentes a um dicionário fixo D. Em [9] é apresentado um codificador baseado em um método equivalente ao descrito em [4]. Este codificador melhora a partição do LZW, gerando um ganho de aproximadamente 5%. Este codificador é denominada $ppo-LZW$ e sua descrição é dada a seguir.

$ppo - LZW$

1. Dicionário inicial - A.
2. Método para encontrar o próximo bloco - $s'_1 = x_1$. $D_2 = \mathcal{A} \cup \{x_1^2\}$. Seja n'_{1+i} o fim do bloco s'_i, i.e., $s'_i = x_{1+n_i}^{n'_{1+i}}$. Seja s''_{1+i} o maior prefixo de $x_{1+n'_{1+i}}^n$ em D_{1+i}. Então s_i é tal que $s_i s'_{1+i}$ é o maior prefixo de $x_{1+n_i}^n$, onde s_i é um prefixo de s'_i e $s'_{1+i} \in D_{1+i}$.
3. Atualização do dicionário - Seja v_1 o menor prefixo de $x_{1+n_{1+i}}^n$ que não pertence a D_{1+i}, e v_2 o menor prefixo de $x_{1+n'_{1+i}}^n$ que não pertence a D_{1+i}, então $D_{2+i} = D_{1+i} \cup \{v_1, v_2\}$.
4. Código - codifica o bloco s_i através da sua posição em D_i.

O exemplo a seguir ilustra o funcionamento do $ppo-LZW$.

Exemplo 2 ($ppo - LZW$) Considerando o alfabeto binário, i.e., $A = \{0, 1\}$ e a sequência $x_1^n = 010001101100$, então o codificador funciona da seguinte forma. $s'_1 = 0$ e $D_2 = \{0, 1, 01\}$.

1. $D_2 = \{0, 1, 01\}$,
   $s'_1 = 0$, $s''_2 = 1$, $s_1, s'_2 = 0, 1$,
   $C_{D_1}(s_1) = 0$, $v_1 = v_2 = 10$.
2. $D_3 = \{0, 1, 01\} \cup \{10\} = \{0, 1, 01, 10\}$,
   $s'_2 = 1$, $s''_3 = 0$, $s_2 s'_3 = 1, 0$,
   $C_{D_2}(s_2) = 01$, $v_1 = v_2 = 00$.
3. $D_4 = \{0, 1, 01, 10\} \cup \{00\}$
   $= \{0, 1, 01, 10, 00\}$,
   $s'_3 = 0$, $s''_4 = 00$, $s_3 s'_4 = 0, 00$,
   $C_{D_3}(s_3) = 00$, $v_1 = v_2 = 001$.
4. $D_5 = \{0, 1, 01, 10, 00\} \cup \{001\}$
   $= \{0, 1, 01, 10, 00, 001\}$,
   $s'_4 = 00$, $s''_5 = 1$, $s_4 s'_5 = 0, 01$,
   $C_{D_4}(s_4) = 000$, $v_1 = 011$, $v_2 = 11$.
5. $D_6 = \{0, 1, 01, 10, 00, 001\} \cup \{011, 11\}$
   $= \{0, 1, 01, 10, 00, 001, 011, 11\}$,
   $s'_5 = 01$, $s''_6 = 10$, $s_5 s'_6 = 01, 10$,
   $s'_6 = s''_6 = 10$, então $v_1 = v_2 = 101$,
   $C_{D_5}(s_5) = 010$, $v_1 = v_2 = 101$.
6. $D_7 = \{0, 1, 01, 10, 00, 001, 011, 11\} \cup \{101\}$
   $= \{0, 1, 01, 10, 00, 001, 011, 11, 101\}$,
   $s'_6 = 10$, $s''_7 = 11$, $s_6 s'_7 = 1, 011$,
   $C_{D_6}(s_6) = 0001$, $v_1 = 0110$, $v_2 = 110$.
7. $D_8 = \{0, 1, 01, 10, 00, 001, 011, 11, 101\}$
   $\cup \{0110, 110\}$
   $= \{0, 1, 01, 10, 00, 001, 011, 11, 101, 0110, 110\}$,
   $s'_7 = 011$, $s''_8 = 00$, $s_7 s'8 = 011, 00$,
   $C_{D_7}(s_7) = 0110$, $C_{D_8}(s_8) = 0100$.

A palavra código $ppo - LZW(x_1^n)$ é obtida pela concatenação das palavras códigos $C_{D_i}(s_i)$. $\diamond$

III. Aplicando Predição no $ppo - LZW$

O princípio básico da versão $ppo-LZW$ é a redução do comprimento do bloco s'_i, permitindo a maximização da soma $|s_i| + |s'_{1+i}|$. No entanto, é fácil ver que a vantagem deste procedimento está no aumento do comprimento do bloco s'_{1+i}, o que ocorre pelo fato de que neste algoritmo, o início do bloco s'_{1+i} pode ocorrer em qualquer posição do último bloco s'_i. Assim sendo, o codificador tem mais liberdade para procurar padrões mais longos. No entanto, para permitir esta busca mais ampla, não é necessário reduzir o bloco s'_i, basta informar o início do bloco s'_{1+i}. Esta estratégia pode paracer ruim, pois, além de codificar o bloco s'_{1+i}, também precisa codificar a posição do início deste bloco. No entanto, quando houver superposição entre os blocos s'_i e s'_{1+i}, os primeiros símbolos do bloco s'_{1+i} não precisam ser codificados. Isto pode ser feito, criando subconjuntos do dicionário D_i, a partir dos prefixos dos elementos do dicionário. Seja $u \in \mathcal{A}^*$, então $D_i^u = \{d \in D_i : u$ é prefixo próprio de $d\}$. Portanto, para codificar um bloco s'_{1+i}, cujo prefixo é igual ao fim do bloco s'_i, dado por u, basta utilizar um dicionário D_i^u. Como $|D_i^u| < |D_i|$, o número de bits necessários para codificar s_{1+i} é menor que $\log_2 |D_i|$. Partindo desta

análise, este trabalho propõe um novo codificador, denominado *ppo − LZWp, partição pseudo-ótima do LZW com predição*. A descrição deste codificador é dada a seguir.

ppo − LZWp

1. Dicionário inicial - A.
2. Método para encontrar o próximo bloco - $s_1 = s'_1 = x_1$. $D_2 = \mathcal{A} \cup \{x_1^2\}$. Para cada $i > 1$, seja s''_i o maior prefixo de $x_{n_i+1}^n$ em D_i. Seja $\mathcal{P}$ o conjunto formado por todos os prefixos de s_{i-1}. Seja ps'_i o maior prefixo de $x_{1+n_{i-1}}^n$ tal que $p \in \mathcal{P}$ e $s'_i \in D_i$. O bloco s_i é tal que $ps'_i = s_{i-1}s_i$.
3. Atualização do dicionário - Seja v_1 o menor prefixo de $x_{1+n_{1+i}}^n$ que não pertence a D_i, e $v_2 = s'_i x_{1+n_{1+i}}$, então $D_{1+i} = D_{1+i} \cup \{v_1, v_2\}$.
4. Código - Seja u um bloco tal que $pu = s_{i-1}$. Então codifica s_i através do comprimento $|u|$ e da posição de s'_i em D_i^u.

Através da descrição do $ppo-LZWp$ é possível observar, que a cada passo i, é preciso codificar o comprimento do contexto u. Isto pode ser feito através de um código de inteiros [3]. A seguir é apresentado um exemplo para o $ppo-LZWp$, onde o comprimento do contexto u é codificado através de um código que mapeia um inteiro k em k bits iguais a zero seguido de um bit igual a um.

Exemplo 3 ($ppo - LZWp$) Considerando o alfabeto binário, i.e., $A = \{0, 1\}$ e a sequência $x_1^n = 010001101100$, então o codificador funciona da seguinte forma. $s_1 = s'_1 = 0$, $C_{D_1}(s_1) = 0$ (pois obviamente $|u| = 0$ para o primeiro símbolo) e $D_2 = \{0, 1, 01\}$.

2. $s_1 = 0$, $s''_2 = 1$, $p = 0$, $s'_2 = 1$,
   $p = s_1$, $u = \lambda$, $s_2 = s'_2 = 1$,
   $C^u_{D_2}(s_2) = 1, 01$, $v_1 = v_2 = 10$.
3. $D_3 = \{0, 1, 01\} \cup \{10\} = \{0, 1, 01, 10\}$,
   $s_2 = 1$, $s''_3 = 0$, $p = 1$, $s'_3 = 0$,
   $p = s_2$, $u = \lambda$, $s_3 = s'_3 = 0$,
   $C^u_{D_3}(s_3) = 1, 00$, $v_1 = v_2 = 00$.
4. $D_4 = \{0, 1, 01, 10\} \cup \{00\}$
   $= \{0, 1, 01, 10, 00\}$,
   $s_3 = 0$, $s''_4 = 00$, $p = 0$, $s'_4 = 00$,
   $p = s_3$, $u = \lambda$, $s_4 = s'_4 = 00$,
   $C^u_{D_4}(s_4) = 1, 100$, $v_1 = v_2 = 001$.
5. $D_5 = \{0, 1, 01, 10, 00\} \cup \{001\}$
   $= \{0, 1, 01, 10, 00, 001\}$,
   $s_4 = 00$, $s''_5 = 1$, $p = 0$, $s'_5 = 01$,
   $p \neq s_4$, $u = 0$ e $s_5 = 1$,
   $C^u_{D_5}(s_5) = 01, 00$, $v_1 = 011$ e $v_2 = 11$.
6. $D_6 = \{0, 1, 01, 10, 00, 001\} \cup \{011, 11\}$
   $= \{0, 1, 01, 10, 00, 001, 011, 11\}$,
   $s_5 = 1$, $s''_6 = 10$, $p = 1$, $s'_6 = 10$,
   $p = s_5$, $u = \lambda$, $s_6 = s'_6 = 10$,
   $C^u_{D_6}(s_6) = 1, 011$, $v_1 = v_2 = 101$.
7. $D_7 = \{0, 1, 01, 10, 00, 001, 011, 11\} \cup \{101\}$
   $= \{0, 1, 01, 10, 00, 001, 011, 11, 101\}$,
   $s_6 = 10$, $s''_7 = 11$, $p = 1$, $s'_7 = 011$,
   $p \neq s_6$, $u = 0$, $s_7 = 11$,
   $C^u_{D_7}(s_7) = 01, 11$, $v_1 = 0110$, $v_2 = 110$.
8. $D_8 = \{0, 1, 01, 10, 00, 001, 011, 11, 101\}$
   $\cup\{0110, 110\}$
   $= \{0, 1, 01, 10, 00, 001, 011, 11, 101, 0110, 110\}$,
   $s_7 = 11$, $s''_8 = 00$, $p = 11$, $s'_8 = 00$,
   $p = s_7$, $u = \lambda$, $s_8 = s'_8 = 00$,
   $C^u_{D_8}(s_8) = 1, 0100$.

A palavra código $ppo - LZWp(x_1^n)$ é obtida pela concatenação das palavras códigos $C^u_{D_i}(s_i)$. $\diamond$

Ainda com relação ao codificador $ppo - LZWp$ é possível observar que o início do bloco $s'_i = us_i$ pode acontecer em qualquer símbolo do bloco s_{i-1}. A medida que i cresce, o comprimento do bloco s_{i-1} também cresce, e a busca pelo melhor ponto inicial vai se tornando um processo mais demorado. Na verdade, este fato não é crítico pois a complexidade computacional deste algoritmo ainda assim é mais baixa que a complexidade computacional de outros codificadores (p.ex. o $LZ77$ proposto em [12], que também é bastante utilizado na prática). No entanto, é possível modificar ligeiramente este algoritmo para garantir uma complexidade computacional bem próxima à do LZW. Para isto basta limitar a busca do ponto inicial de s'_i. Isto pode ser feito, tomando um comprimento máximo para o contexto u. Assim sendo, é possível definir uma variação do $ppo - LZWp$. Seja o_{max} uma constante. Então o codificador $ppo - LZW/o_{max}$ é uma variação do $ppo - LZWp$ onde $|u| \leq o_{max}$.

É interessante observar que embora esta técnica tenha sido aplicada sobre o $ppo - LZW$, a utilização de subconjuntos D_i^u possibilita o uso da predição entre blocos em qualquer versão do $LZ78$.

Os resulatdos obtidos para estes codificadores são apresentados na Seção 4.

IV. Resultados

Um conjunto de arquivos de teste bastante utilizado para medir o desempenho de codificadores universais foi proposto em [1], e é denominado *conjunto de Canterbury*. Este trabalho aplicou os codificadores $ppo - LZWp$ e $ppo - LZWp/o_{max}$ (para $o_{max} = 2, 3$) no *conjunto de Canterbury*. Os resultados obtidos são apresentados na Tabela 1, que também contém os re-

sultados do LZW e do $ppo - LZW$. Os resultados são dados através da taxa de compressão (em bits por símbolo).

Os resultados da Tabela 1 mostram que na média, o $ppo - LZWp$ produz um ganho de 4.5% sobre o $ppo - LZW$ e de 10% sobre o LZW. Além disso, é possível observar que a perda em limitar o comprimento do contexto u (através de o_{max}) é pequena. De fato, o resultado obtido para o $ppo-LZWp/2$ é apenas 1% pior que o resultado obtido para o $ppo - LZWp$. Aumentando o_{max} para 3, o resultado se torna praticamente igual ao do $ppo - -LZWp$. Portanto, o ganho do $ppo - LZWp$ pode ser obtido através de um algoritmo com complexidade computacional muito próxima à complexidade do LZW.

V. Conclusões

Este trabalho apresentou uma forma de utilizar a predição nas versões do codificador $LZ78$. Com base no codificador $ppo - LZW$, introduzido em [9], este trabalho propôs uma nova versão do $LZ78$. Esta nova versão denominada $ppo - LZWp$ foi utilizada para comprimir os arquivos do *conjunto de Canterbury*. Os resultados obtidos mostraram que na média, o codificador proposto produz um ganho de 10% sobre o LZW (versão mais popular do $LZ78$), e um ganho de 4.5% sobre o $ppo - LZW$. Com o objetivo de manter a complexidade computacional próxima à complexidade do LZW, foi apresentada uma variação do $ppo-LZWp$, que limita a forma de predição. Esta variação, denominada $ppo - LZWp/o_{max}$ apresentou um desempenho praticamente igual ao desempenho do $ppo - LZWp$, mostrando que o ganho obtido pelo $ppo - LZWp$ pode ser conseguido sem aumentar significativamente a complexidade computacional.

References

[1] R. Arnold, & T. Bell, "A corpus for the evaluation of lossless compression algorithms," Proc. of IEEE Data Compress Conference, pp. 201-210, UT, 1997.

[2] S. De Agostino & J. Storer, "On-line versus off-line computation in dynamic text compression," Information Processing Letters, vol. 59, pp. 169-174, 1996.

[3] P. Elias, "Universal codeword sets and representations of the integers," IEEE Trans. Inform. Theory, vol. IT-21, pp. 194-203, 1975.

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

[5] J. C. Kieffer, "A survey of the theory of source coding with a fidelity criterion," IEEE Trans. Inform. Theory, vol. 39, pp. 1473-1490, 1993.

[6] A. Lempel & J. Ziv, "On the complexity of finite sequences," IEEE Trans. Inform. Theory, vol. IT-22, pp. 75-81, 1976.

[7] G. Louchard & W. Szpankowski, "On the average redundancy rate of the Lempel-Ziv code," IEEE Trans. Inform. Theory, vol. 43, pp. 1-8, 1997.

[8] M. S. Pinho & W. A. Finamore, "Convergence of Lempel-Ziv encoders," Revista da Sociedade Brasileira de Telecomunicações, vol 12, pp. 114-122, 1997.

[9] M. S. Pinho, W. A. Finamore & W. A. Pearlman, "Fast multi-match Lempel-Ziv," Proc. IEEE Data Compress Conference, p. 545, UT, 1999.

[10] S. A. Savari, "Redundancy of the Lempel-Ziv incremental parsing rule," IEEE Trans. Inform. Theory, vol. 43, pp. 9-21, 1997.

[11] T. A. Welch, "A technique for high-performance data compression," IEEE Computer, vol. 17, no 6, pp. 8-19, 1984.

[12] J. Ziv & A. Lempel, "A universal algorithm for data compression," IEEE Trans. Inform. Theory, vol. IT-23, pp. 337-343, 1977.

[13] J. Ziv & A. Lempel, "Compression of individual sequences via variable-rate coding," IEEE Trans. Inform. Theory, vol. IT-24, pp. 530-536, 1978.

TABELA 1. Resultados - *Conjunto de Canterbury*.

Arquivos - x_1^n	$\|x_1^n\|$	LZW	$ppo - LZW$	$ppo - LZWp/2$	$ppo - LZWp/3$	$ppo - LZWp$
alice29.txt	152089	3.26	3.09	2.94	2.92	2.91
asyoulik.txt	125179	3.51	3.36	3.21	3.20	3.20
cp.html	24603	3.68	3.39	3.27	3.26	3.24
fields.c	11150	3.56	3.25	3.05	3.02	2.99
grammar.lsp	3721	3.89	3.65	3.47	3.43	3.40
kennedy.xls	1029744	2.36	2.34	2.44	2.46	2.46
lcet10.txt	426754	3.02	2.83	2.68	2.66	2.64
plrabn12.txt	481861	3.27	3.16	3.05	3.03	3.03
ptt5	513216	0.97	0.92	0.97	0.97	0.97
sum	38240	4.20	3.93	3.74	3.73	3.72
xargs.1	4227	4.42	4.21	4.05	4.04	4.02
Média		3.29	3.10	2.99	2.97	2.96