Transmissão de Informação Embutida em Arquivos Comprimidos com o Algoritmo LZW

Mariana Olivieri Caixeta Altoé e Marcelo da Silva Pinho

***Resumo*— Este artigo apresenta um algoritmo para embutir dados adicionais em um arquivo comprimido através do codificador *LZW*. O método proposto utiliza o fato do codificador não remover completamente a redundância dos arquivos originais, para adicionar bits extras, sem qualquer perda de desempenho na compressão e sem criar qualquer distorção nos dados originais. O desempenho do método é medido através da sua aplicação em três grupos diferentes de arquivos: o conjunto de *Calgary*, o conjunto de *Canterbury* e um grupo de imagens naturais em tons de cinza. Os resultados de simulação mostram que o algoritmo proposto consegue adicionar 0,1 bit a cada símbolo do arquivo original, na média.**

***Palavras-Chave*— informação embutida, informação camuflada, esteganografia, compressão de dados**

***Abstract*— This paper presents an algorithm to embed additional data in a *LZW* compressed file. The proposed method makes use the fact that *LZW* does not completely remove all redundancy to add extra bits in the codeword, with no penalty in compression rate and no distortion in original data. The performance of the method is measured using three different groups of files: *Calgary Corpus*, *Canterbury Corpus*, and a group of natural graylevel images. Simulation results have shown that the algorithm can add 0.1 bit per symbol of the original data, in the mean.**

***Keywords*— information embedding, information hiding, steganography, data compression**

I. Introdução

Nos últimos anos, a transmissão de informação adicional embutida em arquivos de dados, imagens, vídeo e/ou áudio tem despertado muito interesse do meio acadêmico e da indústria [1]. O crescimento da quantidade de informação que trafega pelas redes de comunicação gera uma série de questões relativas a segurança e a confiabilidade da informação. Por esta razão, assuntos como marca d'água digital, impressão digital e transmissão camuflada (que são casos particulares do problema de se embutir informação em sinais) são temas freqüentes em projetos de pesquisa e desenvolvimento.

Os dados transmitidos através de sistemas de comunicação eficientes, em geral, trafegam em forma comprimida. Sendo assim, os estudos para embutir informação nestes dados devem levar em consideração os sistemas de compressão. Um dos métodos mais populares para a compressão de dados é o algoritmo *LZW*, que foi introduzido em [2] e que é baseado no famoso codificador *LZ78*, introduzido por Ziv e Lempel em [3]. Devido à sua baixa complexidade computacional e ao seu bom desempenho, o *LZW* é utilizado em diferentes aplicações. Por exemplo, o padrão *TIFF* utiliza o *LZW* para comprimir imagens sem perda de informação, e o programa *compress* do sistema operacional *UNIX* é baseado neste algoritmo.

Em geral, a seqüência de dados a ser comprimida por um codificador de fonte, possui determinados padrões que se repetem ao longo desta seqüência. Sendo assim, uma forma de se comprimir dados é através da utilização de padrões recorrentes. Para atingir a compressão, basta identificar padrões repetidos e codificar cada padrão através de um ponteiro para sua ocorrência anterior. Esta técnica é conhecida como recorrência de padrões e é utilizada em vários algoritmos de compressão, dentre os quais é possível citar o *LZW*. Basicamente, o algoritmo em questão particiona a seqüência de dados em padrões recorrentes e codifica-os através de ponteiros. Neste caso, o desempenho é função direta da quantidade de partições encontradas.

Uma forma de se inserir informação adicional nos dados comprimidos por um codificador de fonte (ou compressor) é utilizar a redundância remanescente do código. Esta estratégia tem sido utilizada com freqüência para embutir códigos de controle de erro em códigos de fonte, produzindo assim sistemas de codificação conjunta fonte-canal [4], [5]. Por exemplo, em [5] é proposto um método para embutir um código Reed-Solomon, explorando a redundância remanescente, apontada em [6], existente no código *LZ77*, proposto por Ziv e Lempel em [7].

Em [8] foi mostrado que dadas as regras de partição do *LZW*, o algoritmo de partição utilizado não gera uma partição ótima, i.e., uma partição com o menor número possível de padrões. Embora tenha sido provado em [9] que o problema de se encontrar uma partição ótima para o *LZW* seja NP-completo, em [10] foi apresentado um método capaz de melhorar o desempenho do algoritmo de partição na prática. O fato da partição do *LZW* não ser ótima mostra que existe uma redundância remanescente no código. Além disso, como existem métodos práticos para alterar de forma eficiente esta partição, é possível utilizar estes métodos para embutir informação adicional no código *LZW*.

Este trabalho apresenta um método para embutir informação no código gerado pelo algoritmo *LZW*. É importante ressaltar que ao contrário da maioria dos sistemas propostos para sinais de vídeo e áudio, o sistema proposto neste artigo não introduz qualquer distorção nos dados originais, pois a informação adicional é inserida no código e não no sinal. Isto é possível devido à existência de redundância no algoritmo *LZW*. Este artigo está dividido da seguinte forma. A segunda seção apresenta o codificador *LZW*, mostrando que sua partição não é ótima. O algoritmo proposto por este trabalho é introduzido na seção III. Os resultados obtidos são analisados na seção IV. Fechando o trabalho, a seção V apresenta a conclusão do

Mariana Olivieri Caixeta Altoé e Marcelo da Silva Pinho, Divisão de Engenharia Eletrônica, Instituto Tecnológico de Aeronáutica, São José dos Campos, Brasil, E-mails: mariana@ele.ita.br, mpinho@ieee.org. Este trabalho foi financiado pela FAPESP (03/09625-1).

artigo.

II. O ALGORITMO *LZW*

O algoritmo *LZW* foi proposto em [2] com o objetivo de melhorar na prática o desempenho do famoso codificador *LZ78* [3]. Este último foi introduzido por Ziv e Lempel como uma possível solução para o problema da codificação universal para fontes ergódicas. Seja $\mathcal{S}$ uma fonte de informação, cuja saída é uma seqüência semi-infinita de variáveis aleatórias, $X_1^\infty = X_1, \ldots, X_n, \ldots$, que podem assumir valores em um alfabeto finito $\mathcal{A}$, segundo uma medida de probabilidade p. Neste caso, a teoria da informação mostra que a melhor taxa de bits que um codificador de fonte pode atingir é dada pela entropia da fonte, $H(\mathcal{S})$. Além disso, usando a medida de probabilidade p, é possível projetar facilmente um código cuja média da taxa de bits converge para $H(S)$. De fato, existem diferentes métodos capazes de gerar um código com esta característica, como por exemplo o método de Huffman ou o método do codificador aritmético [11]. Infelizmente, na maior parte das aplicações, uma medida de probabilidade que modele a fonte de forma adequada não é conhecida a priori. Além disso, em algumas aplicações, um codificador pode ter que ser utilizado para dados com estatísticas diferentes. É neste contexto que surge a teoria da codificação universal, cujo objetivo é projetar códigos que sejam eficientes para qualquer fonte pertencente a uma determinada classe. Em [3], foi mostrado que quando o codificador *LZ78* é aplicado na saída de uma fonte ergódica arbitrária, sua taxa de bits converge para entropia desta fonte, com probabilidade 1. Este resultado pode ser estendido para o LZW [12].

O algoritmo *LZW* utiliza a técnica da recorrência de padrões para comprimir a saída da fonte. Seja x_1^n uma realização de X_1^n. O algoritmo particiona a seqüência x_1^n em m subseqüências, $(\mathbf{s}_1, \ldots, \mathbf{s}_m)$, tal que $\mathbf{s}_i = x_{j_i}^{j_{i+1}-1}$, onde $j_1 = 1$ e $j_{m+1} - 1 = n$. A subseqüência $\mathbf{s}_i$ é o maior prefixo de $x_{j_i}^n$ que pertence a um dicionário D_i cuja lei de formação é dada a seguir.

$$D_i = \begin{cases} \mathcal{A}, & \text{se } i = 1; \\ D_{i-1} \cup \{\mathbf{s}_{i-1} x_{j_i}\}, & \text{se } i \neq 1. \end{cases} \quad (1)$$

Em (1), $\mathbf{s}_{i-1} x_{j_i}$ representa a concatenação de $\mathbf{s}_{i-1}$ com x_{j_i}. Os elementos de D_i são ordenados e cada subseqüência $\mathbf{s}_i$ é codificada pelo *LZW* através da sua posição no dicionário, utilizando $\lceil \log_2 |D_i| \rceil$ bits, onde $|D_i|$ representa a cardinalidade de D_i e $\lceil y \rceil$ representa o menor inteiro maior ou igual a y. De (1) pode ser observado que $|D_1| = |\mathcal{A}|$ e que a cada passo uma nova seqüência é incluída no dicionário. Portanto, $|D_i| = |\mathcal{A}| + i - 1$ e conseqüentemente o comprimento da palavra código é dado por

$$|\mathit{LZW}(x_1^n)| = \sum_{i=1}^{m} \log_2(|\mathcal{A}| + i - 1), \quad (2)$$

ou seja, fixado o alfabeto da fonte, $|\mathit{LZW}(x_1^n)|$ só depende de m, o número de subseqüências existentes na partição. O seguinte exemplo ilustra o funcionamento do algoritmo.

Exemplo 1: Seja $\mathcal{S}$ uma fonte binária e

$$x_1^n = 0001001010010. \quad (3)$$

Sendo assim, o *LZW* particiona a seqüência em

$$(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4, \mathbf{s}_5, \mathbf{s}_6, \mathbf{s}_7, \mathbf{s}_8) = (0, 00, 1, 001, 0, 10, 01, 0), \quad (4)$$

gerando o código $0, 2, 1, 3, 0, 4, 6, 0$ que transformado em código binário (com o número de bits adequado em cada passo) gera a palavra código

$$\mathit{LZW}(x_1^n) = 0\ 10\ 01\ 011\ 000\ 100\ 110\ 0000 \quad (5)$$

□

No caso em que o dicionário é fixo, em [8], foi mostrado que existem seqüências onde o fato de escolher $\mathbf{s}_i$ como o maior prefixo de $x_{j_i}^n$ pertencente ao dicionário (critério de maior prefixo), não produz necessariamente uma partição com o menor número de subseqüências possíveis (partição ótima). No entanto, se o dicionário é fixo, é possível estabelecer um algoritmo relativamente simples para se encontrar esta partição ótima, vide [8], [10]. Quando o dicionário é variável, também pode ocorrer este problema, i.e., o critério de maior prefixo não produz uma partição ótima. No entanto, neste caso a solução do problema é bem mais complexa. De fato, conforme demonstrado em [9], encontrar uma partição ótima para o *LZW* é um problema NP-completo.

O problema que existe no *LZW* que faz com que o critério de maior prefixo não gere uma partição ótima, é devido ao fato de que em alguns casos um encurtamento da subseqüência $\mathbf{s}_i$, pode permitir uma nova subseqüência $\mathbf{t}_{i+1}$ mais longa que $\mathbf{s}_{i+1}$. Neste caso, se a soma dos comprimentos das duas novas subseqüências ($\mathbf{s}_i$ encurtada e $\mathbf{t}_{i+1}$) for maior que a soma dos comprimentos das seqüências $\mathbf{s}_i$ e $\mathbf{s}_{i+1}$, é melhor escolher a i-ésima subseqüência encurtada. O exemplo a seguir ilustra este fato.

Exemplo 2: Seja x_1^n a seqüência utilizada no Exemplo 1. Se $\mathbf{s}_6$ for encurtada de 10 para 1, a seqüência pode ser particionada em

$$(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4, \mathbf{s}_5, \mathbf{s}'_6, \mathbf{t}_7) = (0, 00, 1, 001, 0, 1, 0010), \quad (6)$$

gerando o código $0, 2, 1, 3, 0, 1, 5$ que transformado em código binário gera a palavra código

$$\mathit{LZW}_m(x_1^n) = 0\ 10\ 01\ 011\ 000\ 001\ 101 \quad (7)$$

□

Comparando (4) com (6) (ou (5) com (7)), é possível notar que a partição do *LZW* pode não ser a mais eficiente possível. Em [10] é proposto um algoritmo capaz de melhorar o desempenho do *LZW* em aproximadamente 5%, quando aplicado nos arquivos do *Corpo de Canterbury* [13].

Esta ineficiência do algoritmo *LZW* pode ser vista como uma redundância remanescente do código. Como este algoritmo é um padrão adotado em diferentes sistemas e devido ao crescente interesse em métodos para embutir informação adicional em dados, esta redundância pode ser utilizada para adicionar novos dados ao código, conforme mostra a seção III.

III. EMBUTINDO INFORMAÇÃO NO *LZW*

Uma análise mais detalhada do exemplo utilizado para ilustrar o funcionamento do *LZW* permite observar que para a seqüência x_1^n, caso o método de escolha do maior prefixo seja relaxado, é possível encontrar outras partições para x_1^n, com a mesma quantidade de subseqüências, m. Dependendo de x_1^n e a medida que n cresce, a quantidade de partições diferentes com o mesmo número de subseqüências, em geral, também cresce. Supondo que para uma seqüência arbitrária, existam N partições diferentes com m subseqüências, então, nesse caso, um sistema de transmissão de dados poderia escolher qualquer uma das N partições diferentes para enviar os dados comprimidos, sem qualquer perda de eficiência em relação ao *LZW*. Sendo assim, a informação adicional seria embutida na escolha da partição a ser transmitida. Desta forma, a quantidade de informação adicional é dada por $\log_2 N$ bits.

É importante observar que a lei de formação do dicionário D_i, definida em (1), mostra que a cada passo do algoritmo, uma nova subseqüência, $\mathbf{s}_{i-1}x_{j_i}$, é inserida em D_i. No entanto, quando $\mathbf{s}_{i-1}$ é encurtado, a nova subseqüência $\mathbf{s}'_{i-1}x_{j'_i}$ irá pertencer ao dicionário, vide a extensão da subseqüência $\mathbf{s}'_6$ no Exemplo 2. Se neste caso, o dicionário não for atualizado, a taxa de crescimento do dicionário diminui. Uma forma de evitar esta alteração é adicionar ao dicionário o padrão $\mathbf{s}'_{i-1}x_{j'_i}\ldots x_{j_i}$.

Encontrar todas as possíveis partições com m subseqüências para uma dada seqüência x_1^n, não parece ser um problema de fácil solução. Devido à semelhança deste problema com o problema de se encontrar a partição ótima, talvez não exista nem mesmo um algoritmo seqüencial capaz de solucionar esta questão. No entanto, uma solução alternativa, que é capaz de encontrar um subconjunto do conjunto de todas as soluções possíveis, pode ser apresentada, utilizando os conceitos aplicados no algoritmo proposto em [10] para melhorar o desempenho da partição. Esta solução, que é a maior contribuição deste trabalho, é apresentada a seguir.

Seja $(\mathbf{s}_1,\ldots,\mathbf{s}_m)$ a partição gerada pelo algoritmo *LZW* para uma seqüência x_1^n arbitrária. Neste caso, o par $(\mathbf{s}_i,\mathbf{s}_{i+1})$ representa a subseqüência $x_{j_i}^{j_{i+2}-1}$. O método proposto, encurtando o padrão $\mathbf{s}_i$, calcula o número, N_i, de pares de padrões contidos em D_i que concatenados sejam iguais a $x_{j_i}^{j_{i+2}-1}$. Este procedimento é repetido para todo i. Sendo assim, é possível afirmar que existe um subconjunto das partições com m subseqüências com um número de elementos igual a

$$N_s = \prod_{i=1}^{m} N_i. \tag{8}$$

Além disso, este algoritmo que calcula os valores $N_1,\ldots,N_m$ é capaz de codificar a seqüência x_1^n, utilizando qualquer uma das partições.

É importante notar que como a lei de formação do dicionário não altera a taxa de crescimento do dicionário, todas as possíveis partições irão gerar palavras código com o mesmo comprimento. Além disso, o método não produz qualquer distorção nos dados comprimidos. Para decodificar a palavra código, o algoritmo de decodificação do *LZW* precisa ser ligeiramente modificado, devido ao fato de que neste caso, $\mathbf{s}'_{i-1}x_{j'_i}$ pode pertencer ao dicionário. Para resolver este problema o algoritmo deve procurar o maior prefixo de $\mathbf{s}'_{i-1}\mathbf{s}'_i$ que não pertence ao dicionário e incluí-lo.

IV. RESULTADOS

O codificador *LZW* foi implementado em *software* e o método para calcular N_s, definido em (8), foi embutido no algoritmo. Para medir a quantidade de informação adicional que se consegue inserir em dados reais, foram utilizados diferentes grupos de dados, que são utilizados freqüentemente para avaliar o desempenho de compressores. Os grupos utilizados foram os seguintes: (a) Corpo de Calgary [14]; (b) Corpo de Canterbury [13]; e (c) Grupo de Imagens Naturais. Neste último grupo, foram utilizadas imagens disponíveis na página na Internet do *Center for Image Processing Research* do *Rensselaer Polytechnic Institute* - NY (http://www.cipr.rpi.edu). As imagens que compõem o grupo são as seguintes: *airplane, baboon, barbara, lena512, peppers, sailboat*, e *tiffany*. Todas elas são em tons de cinza, com 8 bits por pixel, 512 linhas e 512 colunas.

A Tabela I apresenta os resultados para os arquivos do corpo de Calgary. Nela, as colunas representam o nome do arquivo; o tamanho do arquivo em bytes (n); a taxa do código em bits por bytes ($\rho(x_1^n)$); o número de bits que podem ser adicionados ao código ($\lfloor \log_2 N_s \rfloor$); e a razão entre o número de bits adicionados e o número de bytes do arquivo ($\frac{\lfloor \log_2 N_s \rfloor}{n}$). A partir dos resultados é possível observar que o sistema proposto permite, em média, incluir $0,13$ bit no código do LZW, a cada byte dos arquivos do corpo de Calgary.

TABELA I
RESULTADOS OBTIDOS PARA O CORPO DE CALGARY.

Arquivo (x_1^n)	n	$\rho(x_1^n)$ (bits/bytes)	$\lfloor \log_2 N_s \rfloor$ (bits)	$\frac{\lfloor \log_2 N_s \rfloor}{n}$
bib	111.261	3,35	13.964	0,13
book1	768.771	3,27	79.095	0,10
book2	610.856	3,15	67.780	0,11
geo	102.400	6,08	15.772	0,15
news	377.109	3,76	44.215	0,12
obj1	21.504	5,23	2.286	0,11
obj2	246.814	4,17	33.262	0,13
paper1	53.161	3,77	6.964	0,13
paper2	82.199	3,52	10.071	0,12
paper3	46.526	3,81	5.728	0,12
paper4	13.286	4,03	1.709	0,13
paper5	11.954	4,40	1.598	0,13
paper6	38.105	3,92	5.109	0,13
pic	513.216	0,97	20.536	0,04
progc	39.611	3,87	5.275	0,13
progl	71.646	3,03	9.133	0,13
progp	49.379	3,11	6.352	0,13
trans	93.695	3,26	12.513	0,13
Média		**3,71**	**18.965**	**0,13**

Até onde vai o conhecimento dos autores deste trabalho, não existem outros métodos para se embutir dados em palavras código do LZW. Conforme apontado na seção I, em [5] foi apresentado um método para embutir um código Reed-Solomon em palavras código geradas pelo *LZ77*. Apenas como referência, o método proposto em [5] permite incluir em média

$0,14$ bits por bytes quando aplicado no conjunto de Calgary. No entanto, é importante ressaltar que tal número não deve ser comparado diretamente com o apresentado na Tabela I, pois os métodos operam em codificadores distintos, utilizando técnicas totalmente diferentes (p.ex., a técnica utilizada em [5] não pode ser estendida para o *LZW*).

Os resultados obtidos para o corpo de Canterbury são apresentados na Tabela II. Neste caso, a média do número de bits que podem ser incluídos no código é de $0,11$, o que não difere muito dos resultados apresentados para o corpo de Calgary. Dos resultados das Tabelas I e II, é interessante observar que, em geral, a razão $\frac{\lfloor \log_2 N_s \rfloor}{n}$ não varia muito para os diferentes tipos de arquivo. Exceção feita aos casos onde a taxa de bits é muito baixa, por exemplo, o arquivo *pic* no corpo de Calgary e o arquivo *ptt5* no corpo de Canterbury. Este fato já era esperado, pois nestes arquivos, um subconjunto muito grande das subseqüências $s_1, \ldots, s_m$ são palavras que possuem comprimento igual ao da maior subseqüência em D_i. Este fato, faz com que a taxa de bits seja bem reduzida, mas também faz com que a redundância explorada pelo método proposto seja pequena. É importante ressaltar que isto não significa que a redundância do código LZW seja pequena para estes arquivos, e sim que o método proposto é menos eficiente para tais arquivos.

TABELA II
RESULTADOS OBTIDOS PARA O CORPO DE CANTERBURY.

Arquivo (x_1^n)	n	$\rho(x_1^n)$ (bits/bytes)	$\log_2 N_s$ (bits)	$\frac{\log_2 N_s}{n\rho(x_1^n)}$
alice29	152.089	3,27	17.306	0,11
asyoulik	125.179	3,51	14.300	0,11
cp	24.603	3,68	3.102	0,13
fields	11.150	3,56	1.477	0,13
grammar	3.721	3,89	492	0,13
kennedy	1.029.744	2,49	93.930	0,09
lcet10	426.754	3,06	46.705	0,11
plrabn12	481.861	3,29	51.373	0,11
ptt5	513.216	0,97	20.536	0,04
sum	38.240	4,20	5.159	0,13
xargs	4.227	4,42	544	0,13
Média		**3,30**	**23.175**	**0,11**

Por fim, os resultados para o conjunto de imagens são apresentados na Tabela III. É possível notar que para imagens, o número de bits que podem ser incluídos através do método proposto, que na média é de $0,08$ bit por bytes, é ligeiramente menor que os resultados obtidos para os demais conjuntos de teste. No entanto, é interessante observar que o número médio de bits que podem ser incluídos neste grupo de imagens é de 21.356 bits. (ou 2.669 bytes). Esta quantidade de bits seria o suficiente para incluir um código para verificar a autenticidade da informação, ou para incluir em uma imagem médica comprimida, informações sobre o paciente, o local onde exame foi realizado e o equipamento utilizado.

Com relação ao método proposto, ainda é importante notar que a informação incluída não está no sinal, e sim no código LZW. Sendo assim, caso a codificação seja retirada, a informação adicional é perdida, o que poderia ser útil para um sistema de marca d'água frágil, por exemplo. Além disso, como a informação está no código, não é introduzida qualquer distorção no sinal original. Por esta razão, os resultados obtidos para imagens não podem ser comparados com os métodos mais populares que introduzem distorções no sinal original, como por exemplo o método proposto em [15].

TABELA III
RESULTADOS OBTIDOS PARA O CONJUNTO DE IMAGENS.

Arquivo (x_1^n)	n	$\rho(x_1^n)$ (bits/bytes)	$\lfloor \log_2 N_s \rfloor$ (bits)	$\frac{\lfloor \log_2 N_s \rfloor}{n}$
airplane	262.144	5,66	21.766	0,08
baboon	262.144	7,82	17.640	0,07
barbara	262.144	7,84	21.236	0,08
lena512	262.144	6,93	22.959	0,09
peppers	262.144	6,75	22.933	0,09
sailboat	262.144	7,01	21.490	0,09
tiffany	262.144	5,86	21.467	0,08
Média		**6,93**	**21.356**	**0,08**

V. CONCLUSÕES

Este artigo apresentou um método para embutir bits extras na palavra código gerada pelo codificador *LZW*. O algoritmo proposto explora o fato de existirem diferentes formas de particionar uma mesma seqüência de entrada em padrões recorrentes, gerando uma mesma taxa de compressão. Escolhendo a partição de acordo com a seqüência de bits extras que se pretende adicionar à palavra código, o algoritmo atinge o objetivo proposto sem reduzir o desempenho do codificador e sem introduzir qualquer distorção nos dados originais. Este trabalho testou o desempenho do algoritmo quando aplicado no conjunto de *Calgary*, no conjunto de *Canterbury* e em um grupo de imagens naturais em tons de cinza. Os resultados obtidos nas simulações mostraram que a proposta permite incluir aproximadamente $0,1$ bit a cada símbolo do arquivo original. Embora este número pareça pequeno, é importante observar que ele é o suficiente para algumas aplicações interessantes, tais como a inclusão de uma assinatura digital em um documento ou a adição de informações úteis em imagens médicas, como por exemplo, nome do paciente, equipamento utilizado e data do exame.

REFERÊNCIAS

[1] F. A. P. Petitcolas, R. J. Anderson, e M. G. Kuhn, "Information hiding," *Proceedings of IEEE*, vol. 87, no. 7, pp. 1062-1078, 1999.

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

[3] J. Ziv, e A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Transactionson Information Theory*, vol. 24, no. 5, pp. 530-536, 1978.

[4] B. D. Pettijohn, M. W. Hoffman, e K. Sayood, "Joint Source/Channel Coding Using Arithmetic Codes," *IEEE Transactions on Communications*, vol. 49, no. 5, pp. 826-836, 2001.

[5] S. Lonardi, e J. Szpankowski, "Joint source-channel LZ77 coding," *Proceedings of IEEE Data Compress Conference*, 2003.

[6] A. D. Wyner, e A. J. Wyner, "Improved redundancy of a version of the Lempel-Ziv algorithm," *IEEE Transactions on Information Theory*, vol. 41, no. 3, pp. 723-732, 1995.

[7] J. Ziv, e A. Lempel, "A universal algorithm for data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337-343, 1977.

[8] A. Hartman e M. Rodeh, "Optimal parsing of strings," *Combinatorial Algorithms on Words*, Springer-Verlag, A. Apostolico e Z. Galil, editors, 1985.

[9] S. De Agostino e J. Storer, "On-line versus off-line computation in dynamic text compression," *Information Processing Letters*, vol. 59, no. 3, pp. 169-174, 1996.

[10] M. S. Pinho, W. A. Finamore, e W. A. Pearlman, "Fast multi-match Lempel-Ziv," *Proceedings of IEEE Data Compress Conference*, 1999.

[11] T. M. Cover, e J. A. Thomas, *Elements of Information Theory*, Wiley, NY, 1991.

[12] M. S. Pinho, e W. A. Finamore, "Convergence of Lempel-Ziv encoders," *Revista da Sociedade Brasileira de Telecomunicações*, v. 12, no. 2, pp. 114-122, 1997.

[13] R. Arnold, e T. Bell, "A corpus for the evaluation of lossless compression algorithms," *Proceedings of IEEE Data Compresssion Conference*, pp. 201-210, 1997.

[14] T. C. Bell, J. G. Cleary, e I. H. Witten, *Text Compression*, Prentice-Hall, NJ, 1990.

[15] I. J. Cox, J. Kilian, F. T. Leighton, e T. Shamoon, "Secure spread spectrum watermarking for images, audio and video," *IEEE Transactions on Image Processing*, vol. 6, no. 12, pp. 1673-1687, 1997.