Proteção Desigual de Erros usando Códigos BCH para Dados Compactados com LZSS

Zaqueu Cabral Pereira, Richard Demo Souza e Marcelo Eduardo Pellenz

***Resumo*— Este trabalho apresenta um esquema de proteção desigual de erros para dados compactados. O novo método divide o arquivo compactado com LZSS em blocos curtos, permitindo operação próxima de tempo real. A proteção de erros é obtida através da aplicação de códigos BCH e de entrelaçadores de bloco. Comparado com outros métodos encontrados na literatura, o esquema proposto é mais eficiente em termos de taxa de compressão ou em termos do atraso de decodificação.**

***Palavras-Chave*— Compressão de Dados, UEP, LZSS.**

***Abstract*— An unequal error protection scheme for LZSS compressed data is proposed. The novel method divides the compressed data into short blocks, what allows for near real time operation. The error protection is obtained through the application of BCH codes and block interleavers. Compared to other methods found in the literature, the proposed scheme is either more efficiency in terms of compression ratio or in terms of decoding delay.**

***Keywords*— Data Compression, UEP, LZSS.**

I. Introdução

Com o crescente aumento da quantidade de dados que são armazenados e que trafegam em redes de comunicação, tornam-se necessárias estratégias de compressão que melhor aproveitem o espaço de armazenamento ou a banda disponível para transmissão. Quando os dados são do tipo texto [1], banco de dados [2] ou alguns formatos de imagens [3], os esquemas de compressão sem perdas do tipo Lempel-Ziv apresentam muita eficiência [4]–[7]. Com o uso de técnicas de compressão, a forma de representação dos dados é alterada para uma forma mais eficiente. Em contrapartida, apenas poucos erros em algumas partes do arquivo compactado podem corromper todo o arquivo reconstruído.

Aplicações de armazenamento de dados, como dispositivos semicondutores, magnéticos e óticos, são suscetíveis a erros no processo de leitura e gravação. Estes podem ocorrer em surtos devido a regiões defeituosas [8], [9]. Erros em surtos também podem ocorrer devido a desvanecimentos profundos em canais de comunicação sem fio [10], [11]. Como os dados compactados são muito suscetíveis à propagação de erros, torna-se imprescindível a utilização de técnicas de correção de erros [12]. Com a proteção dos códigos corretores o efeito da propagação de erros durante a reconstrução será muito reduzido. Entretanto, um melhor aproveitamento do canal de transmissão ou do dispositivo de armazenamento será comprometido, pois a utilização de técnicas de proteção uniforme (EEP) prejudica em muito a eficiência da compressão. Desse modo, é essencial o desenvolvimento de técnicas de proteção dos dados compactados que sejam mais eficazes.

As técnicas do tipo Lempel-Ziv [4], [5] separam o dado compactado em diferentes classes, algumas dessas com um impacto maior do que outras no processo de reconstrução. Nesse casos os esquemas de proteção desigual (UEP) podem ser usados com sucesso [13]–[15]. Existem também alguns esquemas que implementam mecanismos de resistência a erros em dados compactados do tipo Lempel-Ziv [16], [17]. Tais métodos mantém o arquivo compactado sem nenhum aumento de tamanho, pois aproveitam os espaços remanescentes do processo de compressão. Porém, métodos deste tipo não podem garantir uma dada proteção, já que a quantidade de espaços remanescentes é função do tipo do arquivo.

Alguns dos esquemas UEP citados [14], [15], foram propostos para dados compactados utilizando o LZSS [5]. Esse algoritmo é muito eficiente para compressão de texto [1], [5], e é utilizado em várias aplicações comerciais. Por esta razão adotamos o algoritmo LZSS para realizar o estudo da nova estratégia de UEP que propomos.

Neste trabalho implementamos uma nova solução para proteção desigual de arquivos de texto compactados com o algoritmo LZSS. Essa solução baseia-se em um trabalho anterior onde propomos uma estrutura UEP com códigos Reed-Solomon para os dados compactados com LZSS [15]. Tal estrutura reduz o impacto da propagação de erros nos arquivos ao custo de uma pequena redundância adicional, porém a necessidade da implementação de um *buffer* torna o esquema proibitivo para aplicações em tempo real. Para resolver essa restrição, nosso novo esquema procura dividir as classes de símbolos do arquivo compactado em blocos, que são codificados separadamente utilizando códigos BCH [12] entrelaçados. O desempenho do sistema é comparado com outros esquemas UEP. São apresentadas simulações utilizando arquivos da base de dados Calgary Corpus [18].

A solução proposta neste trabalho pode ser aplicada em casos onde a entrega de dados com poucos erros seja possível. Nesses casos alguns erros locais seriam aceitáveis, enquanto erros globais, como os causados devido à propagação no processo de descompressão, seriam inaceitáveis. Por exemplo, o método poderia ser utilizado em compressão de grandes volumes de dados [19] e transmissão de texto em tempo-real em canais sem fio [20]. Exemplos deste último caso são a tradução simultânea para aulas transmitidas ao vivo, sistemas de legendas para portadores de deficiência auditiva, transmissão de dados informativos para a televisão digital como previsão do tempo e situação do trânsito.

Z. C. Pereira e R. D. Souza, CPGEI, UTFPR, Curitiba, Paraná, Brasil, e-mails: richard,zaqueu@cpgei.cefetpr.br

M. E. Pellenz, PPGIA, PUCPR, Curitiba, Paraná, Brasil, e-mail: marcelo@ppgia.pucpr.br

Este trabalho foi parcialmente financiado pelo CNPq (472977/2007-5).

O restante deste artigo está organizado como segue. Na Seção II apresentamos o algoritmo LZSS. Na Seção III investigamos, através de simulações computacionais, o impacto do surto de erros na descompressão de arquivos compactados com LZSS. Na Seção IV apresentamos os métodos UEP para arquivos compactados com LZSS. Um novo método UEP é desenvolvido buscando um melhor desempenho em sistemas de tempo real na Seção V, além de comparações com outros métodos. Por fim, na Seção VI são apresentados os comentários finais.

II. O ALGORITMO LZSS

O algoritmo LZSS [5] é um tipo de método de dicionário adaptativo variante do LZ77 [4]. No método LZSS, o dicionário é uma parte da seqüência previamente codificada. O codificador examina a seqüência de entrada através de uma janela deslizante como visto na Figura 1. A janela consiste em duas partes, um *buffer* de busca, ou dicionário, que contém uma parte da seqüência recentemente codificada, e um *look-ahead buffer* que contém a próxima parte da seqüência a ser codificada. Na Figura 1, o dicionário contém oito símbolos, enquanto o *look-ahead buffer* contém sete símbolos.

Fig. 1. Exemplo de uma janela deslizante do codificador LZSS.

A janela utilizada no LZSS é basicamente a mesma do LZ77. A codificação do LZ77 gera na saída uma tripla (o, m, c). Nesta, (o) é chamado *offset* e é a distância do ponteiro até o *look-ahead buffer*. O *match* (m) é o número de símbolos consecutivos no dicionário que são idênticos aos símbolos consecutivos no *look-ahead buffer*, iniciando com o primeiro símbolo referenciado no ponteiro, ou seja, o *match* especifica o comprimento da seqüência. O LZ77 usa um terceiro elemento que é o primeiro símbolo do *look-ahead buffer* subseqüente à frase, o caractere (c). Mesmo quando o algoritmo não encontra uma seqüência de símbolos no dicionário, ele envia a tripla com os elementos *offset* e *match* zerados, e um caractere sem codificação, tornando o LZ77 ineficiente nesta situação.

O LZSS também utiliza as mesmas classes do LZ77. Porém, quando o algoritmo não encontra seqüências de símbolos repetidos no dicionário, não é mais gerada a tripla na saída com os dois primeiros elementos zerados. O LZSS usa um bit como prefixo, chamado de *flag* (f), para definir que tipo de palavra-código será gerada. O *flag* assume o valor 0 se não existe nenhuma seqüência de símbolos repetidos no dicionário, ou seja, gera uma saída (f, c) para um caractere não codificado. Quando existe uma seqüência a codificar, o *flag* assume o valor 1, que vem acompanhado do *offset* e do *match* (f, o, m). A equação (1) define a composição da palavra-código C_w no método LZSS:

$$C_w = \begin{cases} (f, o, m) & \text{para } f = 1 \\ (f, c) & \text{para } f = 0 \end{cases} . \qquad (1)$$

O arquivo compactado utilizando LZSS pode ser armazenado ou transmitido com a estrutura vista na Figura 2.

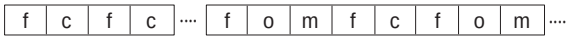

Fig. 2. Estrutura dos dados compactados pelo algoritmo LZSS.

III. IMPACTO DOS SURTOS DE ERROS NO PROCESSO DE DESCOMPRESSÃO

Se erros são introduzidos durante o armazenamento ou transmissão do arquivo compactado, seus efeitos na descompressão podem ser muito grandes, dependendo do tipo de informação que será afetada.

Para avaliar o impacto de um surto de erros ao longo do arquivo compactado com o algoritmo LZSS, foi utilizado um arquivo de texto ('paper1.txt') de uma base de dados de testes de compressão de arquivos de texto [18]. Arquivos da mesma base de dados foram utilizados em [14], [15] para analisar esquemas UEP para variantes Lempel-Ziv. Foi utilizado um surto de erros de 20 bits, em diferentes posições do arquivo compactado. O surto de erros é aplicado para intervalos consecutivos de 20 bits, semelhante a regiões deficientes em um dispositivo de armazenamento de dados [12], ou a um canal sem fio correlacionado [10], [11]. Em seguida avaliamos a taxa de erro de símbolos[1] (SER). A SER média depois da descompressão, considerando que o surto de erros é inserido em todas as diferentes posições, é de 42.68%.

A estrutura LZSS espalha as diferentes classes em todo o arquivo compactado. Contudo, foi considerada uma nova estrutura que permita uma análise melhor de cada uma das classes dos dados compactados com LZSS, a qual foi proposta em [15]. Nessa nova estrutura os bits são organizados em quatro classes diferentes, como mostra a Figura 3, onde **F**, **O**, **M** e **C** correspondem aos bits de *flag*, de *offset*, de *match* e aos de caractere, respectivamente.

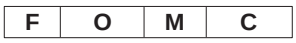

Fig. 3. Estrutura em classes para o arquivo compactado com LZSS.

Após simulações com os dados compactados e organizados como na estrutura da Figura 3, conclui-se que essa estrutura é consideravelmente menos sensível a surtos de erros que a estrutura original do LZSS, mostrada na Figura 2. Considerando essa nova estrutura para o arquivo compactado, a SER média depois da descompressão é de 10.71%.

Utilizando a estrutura proposta na Figura 3, pode-se analisar o efeito de um surto de erros para cada classe separadamente. A Figura 4 mostra o efeito do surto de erros no processo de descompressão, quando o surto de erros de 20 bits é aplicado em cada uma das quatro classes. Analisando a figura, pode-se ver que as classes mais sensíveis são os *flags* e os *matches*.

[1]A SER é definida como a distância de Levenshtein pelo número de símbolos na mensagem transmitida. A distância de Levenshtein é o número mínimo de inserções, eliminações e substituições de símbolos requerido para transformar a mensagem decodificada na mensagem original transmitida.

Se um bit de *flag* é corrompido, então um caractere não-codificado é substituído por uma palavra-código, ou vice-versa. Como resultado, os *offsets* apontarão para uma posição inicial errada, e os *matches* indicarão um comprimento da seqüência errado para ser usado no dicionário. Assim, erros nos *flags* irão corromper o tamanho do arquivo, gerando um arquivo descompactado com um tamanho diferente do arquivo original. Um erro nos *matches* também gera um comprimento da seqüência errado no dicionário e causa erros graves no arquivo descompactado. Em contra partida, os erros nos *offsets* e nos caracteres causam um impacto muito pequeno na descompressão.

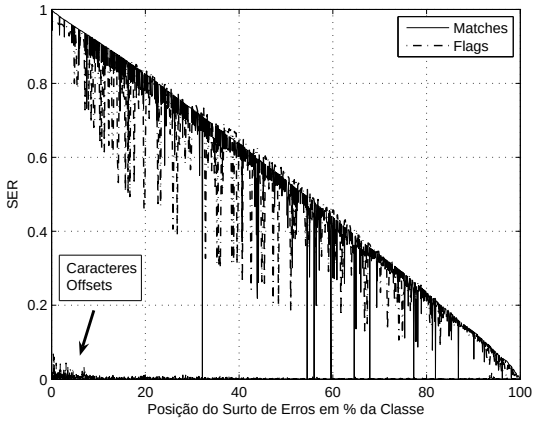

Fig. 4. Efeito do surto de erros de 20 bits no processo de descompressão, em termos da SER para cada classe em função da posição do surto de erros para o arquivo 'paper1.txt'.

IV. Métodos UEP para Arquivos Compactados

Nessa seção são apresentados dois métodos UEP encontrados na literatura [14], [15], os quais serão comparados com o método proposto nesse trabalho.

A. Esquema de Kitakami e Kawasaki

Em [14], é proposto um método de proteção desigual de erros baseado na repetição das partes mais sensíveis dos dados compactados, os *flags* e os *matches*. O método proposto divide os dados, já compactados com LZSS, em vários blocos, cada um contendo B palavras-código. O método agrupa os *flags* **F** e os *matches* **M** no início do bloco. Os *matches* são codificados por codificação unária e uma seqüência de sincronização **SS** é alocada depois da parte que contém os *flags* e os *matches*. Uma cópia dessa parte, mais os *offsets* **O** e os caracteres **C** são alocados depois da seqüência de sincronização. A Figura 5 mostra como as classes de bits são organizadas na estrutura proposta em [14].

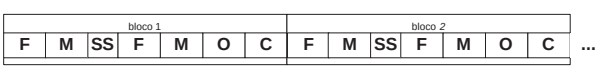

Fig. 5. Estrutura utilizada no método proposto em [14] para o arquivo compactado com LZSS.

Em [14] o esquema foi avaliado por simulações computacionais utilizando arquivos da base Calgary Corpus [18] e com os seguintes parâmetros: *i*) Tamanho do bloco com $B = 200$ palavras-código; *ii*) Tamanho da seqüência de sincronização com 13 bits. Foram feitas algumas simulações para o método em questão. A Figura 6 mostra o impacto de um surto de erros de 20 bits no processo de descompressão, em termos da SER e da posição do surto de erros para o arquivo 'paper1.txt'. Nota-se que a média de erros nesse caso é de apenas 0.16%. A redundância adicional inserida no arquivo compactado foi de 12.57% do arquivo original.

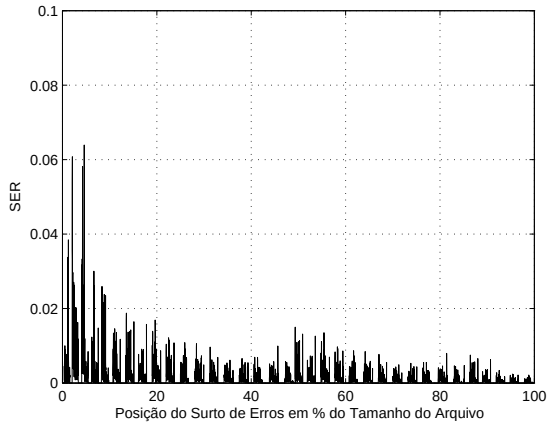

Fig. 6. SER depois da descompressão, em função da posição do surto de erros de 20 bits, considerando o esquema em [14], e o arquivo de texto 'paper1.txt'.

Este método, apesar de garantir uma proteção eficaz contra surtos de erros, insere a redundância como uma repetição dos bits de *flag* e dos bits de *match*, correspondendo a um código de repetição de taxa $\frac{1}{2}$, o que degrada muito a taxa de compressão do LZSS. Além disso, utiliza codificação unária dos *matches*, que é mais um fator para degradação da taxa de compressão.

B. Esquema UEP utilizando Códigos Reed-Solomon

Em [15], foi proposto um esquema UEP, que faz uso da estrutura modificada vista na Figura 3. Os *flags*, *offsets*, *matches* e caracteres são codificados separadamente, tal que possam ter diferentes níveis de proteção em cada uma das quatro classes. Sejam t_f, t_o, t_m, e t_c os níveis do correção de erros do código de canal usados para os *flags*, *offsets*, *matches* e caracteres, respectivamente. Então, de acordo com a análise do impacto de surtos de erros em cada uma das classes, um esquema de proteção desigual de erros eficiente apresenta t_f e t_m muito maiores que t_c e t_o:

$$(t_f; t_m) >> (t_o; t_c).$$

Na estratégia UEP proposta em [15] é usado um código corretor de erros Reed-Solomon com alta taxa. A Figura 7 mostra a estrutura final do dado compactado com a inserção da redundância. Há um decréscimo na taxa de compressão devido à inserção das redundâncias $\widetilde{\mathbf{F}}$, $\widetilde{\mathbf{O}}$, $\widetilde{\mathbf{M}}$, e $\widetilde{\mathbf{C}}$.

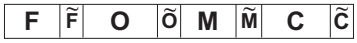

Fig. 7. Estrutura do arquivo compactado com a aplicação da UEP com Reed-Solomon.

Uma desvantagem deste esquema UEP é, devido a sua estrutura mostrada na Figura 7, requerer que os dados sejam colocados em um *buffer* antes do armazenamento ou transmissão. Em comparação com o esquema proposto em [14] este esquema mostra-se mais eficiente em termos de compressão e garantia de proteção. Porém, o esquema UEP Reed-Solomon é deficiente quando utilizado para transmissão em tempo real, apresentando a necessidade de grandes *buffers*, o que não ocorre em [14].

O desempenho do método em [15] é visto na Figura 8 para o caso de um surto de 20 bits e o arquivo 'paper1.txt'. Os *flags* e *matches* são protegidos usando o código Reed-Solomon (255,249,3) definido sobre GF(256), enquanto os *offsets* e os *caracteres* não foram protegidos. A SER foi de apenas 0.11%, enquanto a redundância adicional inserida foi de apenas 0.51% do tamanho do arquivo original [15].

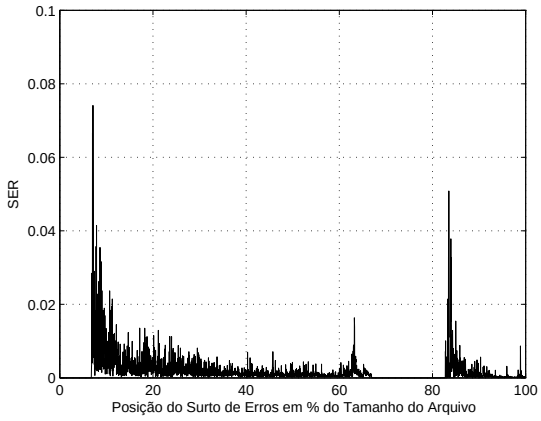

Fig. 8. SER depois da descompressão, em função da posição do surto de erros de 20 bits, considerando o esquema em [15], e o arquivo de texto 'paper1.txt'.

V. UEP USANDO CÓDIGOS BCH E PROCESSAMENTO EM BLOCOS

Como visto nas seções anteriores, os esquemas UEP propostos em [14] e [15] são muito eficientes em relação à proteção de erros. Contudo, ambos métodos apresentam importantes deficiências que podem limitar sua utilização prática. O esquema em [14] tem uma baixa eficiência de compressão, enquanto o método em [15] requer um *buffer* muito longo.

O esquema UEP introduzido nesse trabalho procura aproveitar as vantagens dos métodos anteriores, evitando ao mesmo tempo suas fraquezas. Nosso objetivo é propor um esquema capaz de conseguir uma dada capacidade de correção de erros, com uma alta eficiência de compressão sem a necessidade de um *buffer* muito longo. Para evitar a utilização desse *buffer*, foi considerado um processamento em blocos, cada bloco contendo B palavras-código, como em [14]. Contudo, aqui a estrutura a ser usada é consideravelmente diferente da estrutura na Figura 5. Aqui, em cada bloco as classes são agrupadas seguidas de suas correspondentes redundâncias, como mostrado na Figura 9.

Fig. 9. Estrutura geral do arquivo compactado depois da aplicação da proteção desigual em blocos.

Para codificar cada classe foram utilizados códigos BCH binários. Esses códigos são definidos sobre GF(2) e são representados pela tripla (n, k, t), onde n é o número de bits de saída, k é o número de bits de entrada, e t é a capacidade de correção de erros. Códigos BCH tem um bom desempenho na correção de surtos de erros [12].

Para alcançar uma alta capacidade de correção de erros com códigos BCH, foram utilizados entrelaçadores de bloco em cada classe em um bloco de B palavras-códigos. Entrelaçadores de bloco com tamanho λ permitem a construção de um código cíclico $(n \cdot \lambda, k \cdot \lambda, t \cdot \lambda)$ dado um código de bloco regular cíclico (n, k, t) [12]. Isto é feito arranjando λ saídas do codificador BCH em uma matriz com λ linhas, e então transmitir e armazenar em colunas. A utilização de uma entrelaçador de bloco aumenta consideravelmente a capacidade de correção de erro dos códigos cíclicos como códigos BCH, enquanto mantém a mesma taxa k/n. Além disso, permite uma grande flexibilidade na definição do tamanho da palavra código, sem alterar o codificador ou decodificador.

A. Comprimento de Blocos Iguais para Flags e Matches

Para o caso dos *flags* o tamanho do sub-bloco **F** é sempre o mesmo, $k \cdot \lambda = B$, exceto para o último bloco. Nesse bloco é necessário adicionar zeros no fim do bloco até que ele tenha B bits. O comprimento do sub-bloco dos *matches* **M** é variável, uma vez que algumas saídas do codificador LZSS não são codificadas, contendo apenas *flags* e caracteres. Se o número de bits de *matches* é menor que $k \cdot \lambda$, são adicionados zeros até o bloco ser completado. Caso contrário, se o número de bits de *matches* é maior que $k \cdot \lambda$, são acrescentados zeros até o próximo múltiplo de k. Considerando que os *offstes* e os caracteres têm uma contribuição muito pequena para a SER final, como visto na Seção III, não é adicionada redundância nessas classes. Essa estratégia garante pelo menos uma capacidade correção de erros de $t \cdot \lambda$ bits de *flags* e *matches*.

Na descompressão, como o comprimento das palavras-código LZSS por bloco é fixo, a reconstrução é realizada com a decodificação das classes com redundância adicional e reorganizando os bits na estrutura original da Figura 2. Esse processo é implementado bloco por bloco, e a decodificação LZSS pode iniciar em paralelo com a recepção do próximo bloco.

Para avaliar o desempenho do esquema proposto, foi considerado o caso de um surto de erros de 20 bits de comprimento,

como usado em [14]. Portanto, para proteger o arquivo compactado do surto foi necessário garantir pelo menos $t_f = 20$ e $t_m = 20$. Baseado nos códigos listados em [12], decidimos pela aplicação do código BCH $(31, 21, 2)$, além de um entrelaçador de bloco com pelo menos $\lambda = 10$ linhas. Com tais parâmetros pode ser garantida uma capacidade de correção de erros de pelo menos 20 bits para ambas as classes, dos *matches* e *flags*. O número de palavras código em cada bloco é portanto $B = k \cdot \lambda = 210$.

A Figura 10 mostra a SER depois da descompressão em função da posição do surto de erros, para o caso do arquivo 'paper1.txt'. A SER média nesse caso foi de 0.11%, o que é muito similar às obtidas pelos dois esquemas discutidos na Seção IV (0.16% para o método em [14] e 0.11% para o método em [15]). Em termos de redundância adicional o método proposto requer 4.94% do tamanho do arquivo original, o que está entre o esquema em [14] (12.57%) e o esquema em [15] (0.51%).

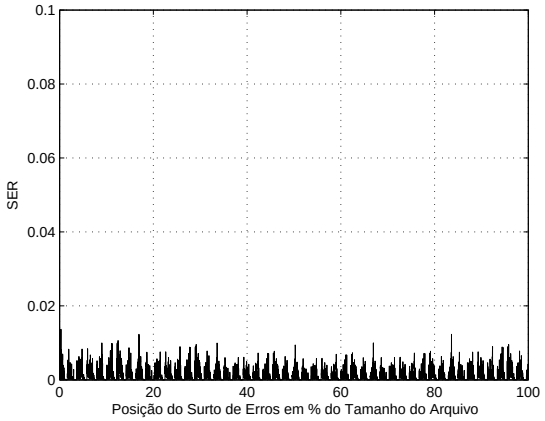

Fig. 10. SER depois da descompressão, em função da posição do surto de erros de 20 bits, considerando o esquema proposto, e o arquivo de texto 'paper1.txt'.

A eficiência de compressão do esquema proposto pode ser melhorada consideravelmente aumentando o comprimento do código BCH e mantendo a mesma capacidade de correção de erros. Por exemplo, substituindo o código (31,21,2) pelo código (63,51,2) a redundância adicional vai de 4.94% para 4.35%. É importante dizer que, desde que mantidos $\lambda = 10$, agora o tamanho do bloco é de $k \cdot \lambda = 510$. Aumentando ainda mais o comprimento do código seria aumentada ainda mais a eficiência de compressão. Contudo, isso estaria em contraste com um dos objetivos do esquema proposto: trabalhar com comprimentos de bloco curtos. Portanto, foi decidido não usar blocos maiores que $B = 510$ bits.

B. Diferentes Comprimentos de Blocos para os Matches

É bastante esclarecedor observar o comportamento do números de bits de *match* gerados durante o processo de compressão. No início da codificação LZSS o dicionário é ainda muito pequeno, e o número de caracteres não-codificados é relativamente alto, rendendo uma quantidade pequena de *matches*. A Figura 11 mostra o número de símbolos de *match* para o arquivo 'paper1.txt' em função do número de blocos ($B = 210$ bits). A partir da Figura 11 pode-se ver que o número de símbolos de *match* aumenta com o número de blocos até se estabilizar. Esse resultado também permite concluir que nos primeiros blocos é muito ineficiente considerar o mesmo comprimento de bloco para os símbolos de *match* que é considerado para os de *flags*. Como nos primeiros blocos seriam poucos símbolos de *match*, muitos zeros deveriam ser adicionados no fim do bloco reduzindo a eficiência de compressão.

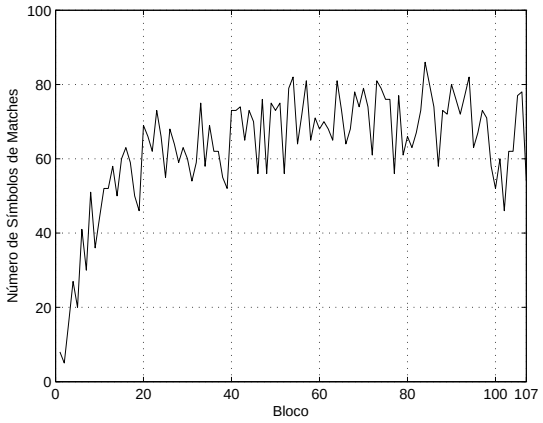

Fig. 11. Números de símbolos de *match* em cada bloco de $B = 210$ palavras-código para o arquivo compactado 'paper1.txt'.

Baseado nos fatos anteriores, foi considerada outra estratégia onde para a primeira parte dos *matches* foi utilizado um tamanho de bloco menor ($B_{m1} = 210$) com o código BCH $(31, 21, 2)$, e para os *flags* e para a segunda parte dos *matches* foi utilizado um tamanho maior ($B_{m2} = 510$) com o código BCH $(63, 51, 2)$. Em todos os casos o entrelaçador de bloco tem $\lambda = 10$ linhas. Nas simulações seguintes foram utilizados um tamanho de bloco de $B_{m1} = 210$ bits somente para os primeiros 20 blocos. Usando essa estratégia, a redundância adicional requerida vai de 4.35% para 3.03%.

A Tabela I apresenta os resultados de SER média e a redundância adicional inserida no arquivo compactado, para vários arquivos do Calgary Corpus, considerando o mesmo surto de erros de 20 bits. Como pode ser visto, com o custo da redundância adicional em torno de 4% é possível reduzir a SER média depois da descompressão para valores menores que 0.3%. O comportamento global é muito similar ao que foi discutido anteriormente para o arquivo 'paper1.txt'. A SER média depois da descompressão é a mesma alcançada pelos outros dois métodos, enquanto a redundância adicional (em termos do tamanho do arquivo original) está entre as redundâncias inseridas pelo método em [14] (mais redundância) e o método em [15] (menos redundância).

A Tabela II compara a redundância adicional para o método proposto, e as requeridas para os esquemas apresentados em [14] e [15], para três diferentes tipos de arquivos. Os três métodos foram projetados para proteger os *flags* e os *matches* contra surtos de erros de 20 bits, e conseguem SERs médias

TABELA I
SER MÉDIA E REDUNDÂNCIA ADICIONAL USANDO O ESQUEMA PROPOSTO.

	paper1	paper2	paper3	paper4	paper5	paper6	progc	progp	obj1
SER Média	0.11%	0.09%	0.10%	0.24%	0.26%	0.14%	0.16%	0.23%	0.13%
Redundância Adicional	3.03%	2.97%	3.71%	4.30%	3.93%	3.19%	3.08%	2.58%	4.32%

TABELA II
REDUNDÂNCIA ADICIONAL PARA OS TRÊS MÉTODOS E PARA TRÊS DIFERENTES TIPOS DE ARQUIVOS.

	paper1	progp	obj1
Método em [14]	12.57%	8.40%	20.01%
Método em [15]	0.51%	0.40%	0.52%
Esquema Proposto	3.03%	2.58%	4.32%

muito próximas depois da descompressão. Como pode ser visto a partir da Tabela II, o método em [15] e o proposto nesse artigo alcançam uma eficiência de compressão muito melhor que o esquema em [14]. Entretanto, o esquema proposto nesse artigo trabalha com blocos relativamente curtos, enquanto o método em [15] trabalha com o arquivo inteiro. Portanto, o novo método é capaz de alcançar desempenho próximo do esquema em [15], e ainda ser apropriado para tempo real como o esquema em [14].

Os resultados apresentados nesse trabalho consideraram um surto de erros de 20 bits. Entretanto, para outros tamanhos de surtos, tais como 48 e 96, fazendo as modificações apropriadas, o desempenho relativo entre os três métodos é muito similar ao que é visto na Tabela II.

Por fim, é importante fazer uma colocação relativa aos esquemas que exploram a redundância residual resultante do processo de compressão [16], [17]. De acordo com [16] a redundância residual usando o algoritmo LZ77 está entre 1.0% e 2.5% para diversos arquivos do Calgary Corpus. Sendo assim, seria possível implementar uma dada UEP tal que a redundância adicional necessária seja em torno desse valor. Por exemplo, com essa redudância para o arquivo 'paper1.txt' seria possível proteger contra um surto de erros de tamanho pequeno (entre 9 e 16 bits) usando o esquema proposto neste trabalho, considerando LZSS. Porém, não seria possível garantir a operação em tempo-real, já que os blocos teriam tamanho variável. Dessa forma, a comparação direta dos métodos baseados em redundância residual com aqueles semelhantes ao proposto neste artigo nos parece ser injusta. Entretanto, uma junção das duas filosofias deveria ser investigada com detalhes.

VI. CONCLUSÃO

Neste trabalho foi proposto um novo esquema de proteção desigual de erros para dados compactados com LZSS. O esquema é capaz de proteger os dados contra surtos de erros de um dado tamanho de bits, com um custo de redundância adicional muito pequeno. Duas diferentes estratégias de implementação foram propostas. A primeira considera tamanho de blocos iguais para os *flags* e *matches*. A segunda aumenta a eficiência de compressão usando dois tamanhos diferentes de blocos para os *matches*.

Comparado com outros métodos similares, o esquema proposto apresenta claras vantagens. Tanto em termos de eficiência de compressão, quando comparado com o método introduzido em [14], como em termos de operação em tempo real, quando comparado com o esquema em [15]. Além disso, o esquema proposto alcança basicamente o mesmo desempenho de SER que os outros métodos discutidos nesse artigo.

REFERÊNCIAS

[1] T. C. Bell, J. G. Cleary, and I. H. Witten, *Text compression*, Prentice Hall, 1990.

[2] W. K. Ng, and C. V. Ravishankar, "Block-oriented compression techniques for large statistical databases", *IEEE Trans. on Knowledge and Data Engineering*, vol. 9, no. 2, pp. 314–328, 1997.

[3] D. Greene, M. Vishwanath, F. Yao, and T. Zhang, "A progressive Ziv-Lempel algorithm for image compression", *Proceedings of Compression and Complexity of Sequences*, 1997.

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

[5] J. A. Storer, and T. G. Syzmanski, "Data Compression via Textual Substituion", *Journal of the ACM*, vol. 29, pp. 928-951, 1982.

[6] K. Sayood: *Introducion to Data Compression*, Morgan Kaufmann Publishers, 2000.

[7] T. Bell, "Better OPM/L text compression", *IEEE Transactions on Communications*, vol. 34, no. 12, pp. 1176–1182, 1986.

[8] J. C. Lo, M. Kitakami, and E. Fujiwara, "Reliable Logic Circuits with Byte Error Control Codes – A Feasibility Study", *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Nov. 1996.

[9] W. Coene, H. Pozidis, M. Van Dijk, J. Kahlman, R. Van Woudenberg, and B.Stek, "Channel coding and signal processing for optical recording systems beyond DVD", *IEEE Transactions on Magnetics*, vol. 37, no. 2, pp. 682–688, 2001.

[10] E. Biglieri, J. Proakis, and S. Shamai, "Fading channels: information-theoretic and communications aspects", *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2619–2692, 1998.

[11] W. Zhu, and J. Garcia-Frias, "Stochastic context-free grammars and hidden Markov models for modeling of bursty channels", *IEEE Transactions on Vehicular Technologies*, vol. 53, no. 3, pp. 666–676, 2004.

[12] S. Lin and D. J. Costello Jr., *Error Control Coding*, Prentice-Hall, 2004.

[13] E. Fujiwara and M. Kitakami, "Unequal error protection in Ziv-Lempel coding", *IEICE Transactions on Information & Systems*, vol. E86-D, no. 12, pp. 2595–2600, 2003.

[14] M. Kitakami, and T. Kawasaki, "Error Recovery Method for LZSS Coding", *Proceedings of International Symposium on Theory and its Applications*, Parma, pp. 1158–1163, Oct. 2004.

[15] Z. C. Pereira, M. E. Pellenz, R. D. Souza and M. A. A. Siqueira,"Unequal Error Protection for LZSS Compressed Data Using Reed-Solomon Codes", *IET Communications*, vol. 1, no.4, pp.612-617, 2007.

[16] S. Lonardi, W. Szpankowski, and M. D. Ward,"Error Resilient LZ'77 Data Compression: Algorithms, Analysis, and Experiments", *IEEE Transactions on Information Theory*, vol. 53, no. 5, 2007.

[17] M. O. C. Altoé and M. S. Pinho, "Transmissão de Informação Embutida em Arquivos Comprimidos com o Algoritmo LZW", *Anais do XXII Simpósio Brasileiro de Telecomunicações SBT'05*, 2005.

[18] http://www.data-compression.info/Corpora/CalgaryCorpus/

[19] S. Perkins and D. H. Smith, "Robust data compression: variable length codes and burst errors", *The Computer Journal*, vol 20, no.3, pp. 315–322, 2005.

[20] D. Maniezzo, M. Cesana, P. Bergamo, M. Gerla, and K. Yao, "Real-time caption streaming over WiFi network", *Proceedings IEEE International Conference on Information Technology: Research and Education*, 2003.