COMPRESSAO/DESCOMPRESSAO LZW MODIFICADA

Francisco V. Brasileiro
Jacques P. Sauvé
J.Antão B. Moura

UFPb - CCT - DSC - COPIN/GRC
58100 - Campina Grande - Pb

SUMARIO

Descreve-se um algoritmo de compressão em um passo que explora localidade de referência, como a que ocorre quando padrões são usados frequentemente em uma determinada porção de mensagens ou arquivos e depois caem em desuso. O algoritmo é uma modificacão daquele apresentado por Lempel, Ziv e Welch (LZW). Medidas obtidas com arquivos reais, apontam um melhor desempenho que o das técnicas LZW e de Huffman.

ABSTRACT

A one-pass compression algorithm that exploits locality of reference, such as occurs when data patterns are used frequently over certain portions of messages or files and then fall in disuse, is described. The algorithm is a modification of that by Lempel, Ziv and Welch (LZW). Measurements with real files indicate that it performs better than LZW and Huffman coding.

I - INTRODUCAO

A disseminação do uso de computadores e a expansão da quantidade de informação que deve ser armazenada e acessada para a automação das atividades científicas, comércio-industriais, econômicas e militares levam a uma demanda enorme e crescente para meios de armazenamento secundário e de canais de comunicação com elevadas capacidades.

As unidades de armazenamento a laser e os canais de comunicação com fibras óticas são evoluções na tecnologia de hardware para atender a esta demanda. Uma alternativa à aquisição de hardware para armazenamento e comunicações é a adoção de técnicas que reduzam a quantidade de redundância nos dados a serem manipulados. A diminuição de redundância na representação da informação reduz a necessidade de armazenamento e os custos com comunicação de dados, e pode ser alcançada pelas técnicas de compactação ou de compressão de dados.

Na compactação, elimina-se fisicamente "informação irrelevante", preservando-se a integridade da representação da informação relevante. Ilustrando: os "zeros à extrema esquerda" em preços de produtos (p.ex., Cz$ 000 1000,00) podem ser descartados sem prejuízo para o cálculo de faturamento (no caso, Cz$ 1000,00). O arquivo com dados de entrada para um aplicativo de controle de faturamento poderia pois, ser compactado (eliminado-se os zeros irrelevantes, no presente exemplo), reduzindo a área em disco por ele ocupada ou baixando a utilização de linhas de comunicação quando de sua transferência. O emprego de técnicas de compactação tem os inconvenientes de depender do contexto (semântica) da informação e não permitir restaurar a representação original dos dados. As técnicas de compressão não apresentam estas limitações.

Na compressão, toda informação é considerada relevante e a redução física no "tamanho" dos dados é feita mudando-se a sua representação. O processo inverso, de descompressão, recupera a representação original dos dados. O preço de Cz$ 1000,00 no exemplo acima, seria comprimido com a substituição dos sete caracteres componentes do preço ("1000,00") por um código de

apenas um caractere ("M", por exemplo). A descompressão neste caso, é realizada acessando-se uma tabela com o código comprimido ("M") e fazendo-se a conversão para o símbolo ou representação inicial ("1000.00"). A compressão apresenta pois, as vantagens da compactação, sem suas limitações. A favor das técnicas de compressão, pode-se ainda citar a contribuição para segurança dos dados, pois os arquivos comprimidos têm representação de dados alterada da original (com a remoção de redundância) o que dificulta a sua inteligibilidade por quem desconheça a tabela de conversão para os códigos empregados.

Algumas técnicas de compressão/descompressão são ineficientes para reduzir significativamente o tamanho de arquivos com tipos diferentes de redundância (p.ex., texto e dados numéricos). Outras técnicas fazem compressão em "múltiplos passos" - isto é, necessitam primeiro examinar todo o conteúdo do arquivo para "aprender" a distribuição de redundância e então, comprimí-lo. A compressão em múltiplos passos é inadequada para comunicação "on-line" (uma sessão de trabalho em terminal remoto, por exemplo), mas pode resultar em reduções significativas nos tamanhos dos arquivos.

O esquema proposto por Huffman [HUFF 52] é um algorítmo de compressão em dois passos e resulta em códigos de tamanho variável (os caracteres mais frequentes no arquivo são representados por códigos curtos - resultando em redução, e os mais infrequentes por códigos longos - ocasionando expansão). O esquema de Huffman tem sido alvo de muito interesse na literatura devido aos seus altos índices de compressão e eficiência de implementação [REGH 81]. O algorítmo de Huffman, contudo, além de ser inadequado para uso em comunicação "on-line", tem códigos de tamanho variável, dificultando a execução da descompressão [WELC 84].

A literatura [REGH 81, LANG 83, CLEA 84, BASS 86, BENT 86 e BELL 86] traz propostas de variantes da técnica de Huffman, bem como técnicas adaptativas (em um passo), sendo estas últimas, adequadas também à comunicação "on-line".

Dentre as técnicas adaptativas, destaca-se a denominada de LZW [WELC 84] por apresentar: (1) índices de compressão semelhantes aos de Huffman independentemente do tipo de conteúdo dos arquivos; (2) velocidades de compressão/descompressão elevadas; e, (3) implementação em software bastante simples.

Este artigo apresenta uma modificação à técnica de compressão LZW para se conseguir melhores índices de compressão, mantendo-se as características de facilidade de implementação e rapidez de execução. A técnica modificada pode ser empregada em sistemas comerciais pois possibilita alcançar o dobro ou o triplo em densidades efetivas nos dados armazenados ou transmitidos em canais de comunicação. A técnica de compressão/descompressão LZW e a modificação a ela proposta são detalhadas nas seções II e III, respectivamente.

A seção IV traz uma análise comparativa do desempenho da técnica LZW modificada face às técnicas LZW e a de Huffman. As medições feitas indicam a superioridade da técnica LZW modificada em termos de índices de redução e de velocidade de processamento. A seção V aborda a utilização de compressão em transmissão de dados. Finalmente, a seção VI faz conclusões e sugestões para extensão do trabalho.

II - Técnica de compressão/descompressão LZW

Compressão

O algoritmo de compressão de dados LZW, baseia-se na manipulação de uma Tabela de Símbolos (TS) que é dinâmicamente montada à medida que a informação a ser comprimida é lida. Esta tabela é a responsável pelo mapeamento de uma sequência de caracteres de comprimento aleatório em um código com um número constante de bits (comumente se usa um código de 12 bits).

Cada símbolo presente na TS é formado por um prefixo e um caractere de extensão. O prefixo por sua vez, é um símbolo que já pertence à TS, ou seja, se a sequência wK formada pelo prefixo w e pelo caractere de extensão K pertence à TS, então o prefixo w também pertence à TS.

Na geração dos códigos comprimidos, a técnica LZW usa um algoritmo "avarento", que recebe os caracteres da entrada um a um, tentando reconhecer um símbolo com a maior sequência de caracteres que já existe na TS. Quando finalmente se encontra um sequência que não pertence à TS, esta é incluida na TS e o código associado à última sequência (símbolo) reconhecida é emitido. Em seguida o processo é repetido utilizando como prefixo o último caratere lido. O algoritmo de compressão LZW é mostrado abaixo.

```
        Inicialize TS com todos símbolos de um caractere
        Leia o primeiro caractere do arquivo a ser comprimido
        Atribua caractere a prefixo
laço:   Leia próximo caractere do arquivo a ser comprimido
        Se não existem mais caracteres
          Emita o codigo associado a prefixo
          Fim
        Senão
          Se existe prefixo+caractere em TS
            Atribua prefixo+caractere a prefixo
          Senão
            Emita o codigo associado a prefixo
            Inclua prefixo+caractere em TS
            Atribua caractere a prefixo
            Vá para laço
```

O exemplo a seguir servirá para esclarecer melhor a operação de compressão LZW.

Supondo que os dados sejam representados apenas pelos caracteres a, b e c, e que o conteúdo de um arquivo a ser codificado é ababba, teríamos os seguintes passos:

```
Inicilaizar TS com todos os caracteres (1-a, 2-b, 3-c)
Lê o primeiro caractere (a)
Atribui (a) a prefixo
Lê o próximo caractere (b)
Verifica se (a)(b) está em TS
Como (a)(b) não está em TS então
  Emite (1) (código associado a (a))
  Inclui (a)(b) em TS (1-a, 2-b, 3-c, 4-ab)
  Atribui (b) a prefixo
Lê o próximo caractere (a)
Verifica se (b)(a) está em TS
Como (b)(a) não está em TS então
  Emite (2) (código associado a (b))
  Inclui (b)(a) em TS (1-a, 2-b, 3-c, 4-ab, 5-ba)
  Atribui (a) a prefixo
Lê o próximo caractere (b)
Verifica se (a)(b) está em TS
Como (a)(b) está em TS então
  Atribui (a)(b) a prefixo
Lê o próximo caractere (b)
Verifica se (ab)(b) está em TS
Como (ab)(b) não está em TS então
  Emite (4) (código associado a (ab))
  Inclui (ab)(b) em TS (1-a, 2-b, 3-c, 4-ab, 5-ba, 6-abb)
  Atribui (b) a prefixo
Lê o próximo caractere (a)
Verifica se (b)(a) está em TS
Como (b)(a) está em TS então
  Atribui (b)(a) a prefixo
Lê o próximo caractere (EOF)
  Emite (5) (código associado a (ba))
Fim
```

Admitindo-se 8 bits por caractere e por código comprimido, o conteúdo original do arquivo tem 48 bits, enquanto que o conteúdo comprimido resulta em apenas 32 bits.

Obs: Uma melhor representação de TS seria armazenar cada símbolo de TS na forma código do prefixo mais um caractere de extensão. A TS do exemplo acima seria então: (1-0a, 2-0b, 3-0c, 4-1b, 5-2a, 6-4b). esta representação tem como vantagem o fato de que cada entrada de TS terá um tamanho fixo. Esta será a representação usada daqui em diante.

Descompressão

A descompressão LZW constroi, à medida que os códigos comprimidos vão sendo interpretados e a informação original vai sendo restaurada, a mesma TS gerada pela compressão. Cada código recebido é desmembrado atraves da TS em um prefixo e um caractere de extensão. O prefixo encontrado é também desmembrado em um outro prefixo e um outro caractere de extensão recursivamente até que se chegue a um prefixo com um único caractere. Cada código

recebido (exceto o primeiro) gera uma atualizacão na TS. O novo código que é acrescentado à TS é formado pelo código que antecedeu o que está sendo decodificado (que será o prefixo) e o último caractere obtido desta decodificação (que será o caractere de extensão). Note que a sequência de caracteres obtida na decodificação se encontra invertida, portanto o último caractere decodificado é na realidade o primeiro caractere lido na compressão. Abaixo temos o algoritmo de descompressão LZW.

```
          Inicialize TS com todos os símbolos de um caractere
          Leia o primeiro código do arquivo a ser descomprimido
          Atribua código a código_anterior
          Emita o caractere associado ao código lido
 codigo:  Leia o próximo código do arquivo a ser descomprimido
          Atribua código a codigo_entrada
          Se não existem mais códigos
            Fim
          Senão
 simbolo:   Se código não está associado a um único caractere
              Emita o caractere de extensão associado a código
              Atribua o prefixo associado a código a código
              Vá para simbolo
            Senão
              Emita o caractere associado a código
              Inclua  código_antigo+caractere em TS
              Atribua código_entrada a código_antigo
              Vá para código
```

O exemplo abaixo é a descompressão do exemplo apresentado na compressão. A sequência de códigos a ser decodificada é 1 2 4 5, teríamos então os seguintes passos:

```
Inicializar TS com todos os caracteres (1-0a, 2-0b, 3-0c)
Lê o primeiro código (1)
Atribui código a código_antigo
Emite (a) (símbolo associado a (1))
Lê o próximo código (2)
Atribui código a código_entrada
Verifica se (2) está associado a um único caractere
Como está, então
     Emite (b) (caractere associado a (2))
     Inclua código_antigo+caractere em TS (4-1b)
     Atribua código_entrada a código_antigo
Lê o próximo código (4)
Atribui código a código_entrada
Verifica se (4) está associado a um único caractere
Como não está, então
     Emite (b) (caractere de extensão associado a (4))
     Atribui (1) a código (prefixo associado a (4))
Verifica se (1) está associado a um único caractere
Como está, então
     Emite (a) (caractere associado a (1))
     Inclua código antigo+caractere em TS (5-2a)
     Atribua código_entrada a código_antigo
Lê o próximo código (5)
Atribui código a código_entrada
Verifica se (5) está associado a um único caractere
Como não está, então
     Emite (a) (caractere de extensão associado a (5))
     Atribui (2) a código (prefixo associado a (5))
Verifica se (2) está associado a um único caractere
Como está, então
     Emite (b) (caractere associado a (2))
     Inclua código_antigo+caractere em TS (6-4b)
     Atribua código_entrada a código_antigo
Lê o próximo código (EOF)
Fim
```

Obs: Deve-se introduzir uma estrutura do tipo LIFO para resolver o problema da geração invertida da sequência de caracteres descomprimida.

III - Técnica de compressão/descompressão LZW modificada

A tabela de símbolos (TS) da técnica LZW tem tamanho limitado a 2^n símbolos onde n é o número de bits nos códigos comprimidos. É possível pois, no processo de compressão, gerarmos novos símbolos que não possam ser incluídos na TS, por falta de códigos. Tal situação é mais facilmente encontrada com longas sequências de caracteres (p.ex: em atividades de backup de discos ou transferência de longos arquivos).

Os símbolos encontrados após o total preenchimento da TS serão descartados, ou equivalentemente, as redundâncias que aparecerem a partir deste ponto serão desperdiçadas, concorrendo para que o percentual de compressão diminua, principalmente em

arquivos que apresentem "redundâncias localizadas". isto é. redundâncias distintas em trechos distintos. Constatou-se que usando uma TS com 4K entradas (códigos de 12 bits). a compressão começaria a não se aproveitar de redundâncias em arquivos maiores que 10K bytes em média. A modificação proposta para a técnica LZW surgiu desta constatação.

O total preenchimento da TS parece indicar uma falha no dimensionamento da mesma. Este problema poderia ser resolvido se a TS fosse alocada dinâmicamente. porém o crescimento ilimitado da TS esbarra em vários problemas.

No caso de arquivos onde se apresentam "redundâncias localizadas". apenas os últimos símbolos acrescidos seriam constantemente consultados. tornando a TS um recurso mal utilizado. Outro problema é que neste caso o tempo necessário para identificar se um símbolo pertence à TS pode ser proporcional ao seu tamanho e portanto aumentará. e finalmente restrições de projeto podem. na prática. impossibilitar o crescimento da TS (falta de memória por exemplo).

Partindo agora do princípio que a TS deve ter um tamanho fixo. devemos de alguma forma retirar códigos da TS para dar lugar aos novos códigos gerados.

A idéia mais óbvia é substituir os primeiros códigos incluídos pelos novos códigos gerados. ou seja implementar uma política FIFO de atualização da TS. Esta solução também não se aplica. já que desta forma não podemos garantir que o prefixo de um símbolo que pertence à TS também pertence à TS (o mesmo poderia ter sido removido para dar lugar a um novo código).

Chegamos então à modificação por nós sugerida. que consiste em reinicializar a TS quando a mesma encher. Assim. quando um novo símbolo tiver de ser incluído na TS e esta estiver cheia. toda a informação nela armazenada será desprezada e novos símbolos (que talvez representem novos tipos de redundância) serão incluídos na mesma. à partir de então. A perda da informação já armazenada na TS provoca uma queda temporária na capacidade de compressão do algorítmo. Esta queda porém. é

bastante aceitável, pois a TS será rapidamente reconstruida com as novas redundâncias.

Os procedimentos para compressão e descompressão da técnica LZW incorporando a reinicialização da TS são os mesmos apresentados na secão anterior, com as seguintes modificacões:

Tanto na compressão quanto na descompressão, o comando

```
+-----------------------------------------------------------+
|        Inclua código+caractere em TS                      |
+-----------------------------------------------------------+
```

deve ser substituido por:

```
+-----------------------------------------------------------+
|        Se TS não estiver cheia                            |
|             Inclua código+caractere em TS                 |
|        Senão                                              |
|             Reinicialize TS                               |
+-----------------------------------------------------------+
```

IV - Avaliacão de desempenho

A avaliacão do desempenho do algorítmo LZW modificado, foi feita confrontando-se uma implementacão do mesmo com implementacões de algoritmos baseados na técnica LZW pura e na técnica de Huffman. As implementacões foram executadas em uma maquina ED-680 da EDISA com microprocessador 68000 e memória de 2 Mbytes sob o sistema operacional EDIX.

Os arquivos utilizados na ánalise de desempenho foram os seguintes:

Manual - (267.613 bytes) manual da biblioteca INFOSCREEN, contendo texto, figuras e trechos de programas escritos em C.

Apêndice - (11.160 bytes) um dos apêndices do manual descrito acima.

C - (347.949 bytes) concatenacão de vários arquivos contendo programas escritos em C.

Iltd - (122.544 bytes) módulo executável do programa ILTD.

Backup - (4 mega bytes) "dump" de um disco, correspondente a diretórios de usuários de um sistema de arquivos em ambiente UNIX.

As medidas de desempenho utilizadas foram:

a) Indice de Redução (%) definido por:

$$IR = 100 \times \frac{(S_o - S_c)}{S_o}$$

onde S_o é o tamanho do arquivo original e S_c é o tamanho do arquivo comprimido.

b) Tempo de Resposta Total (TRT) o qual é a soma do tempo de compressão e tempo de descompressão em segundos.

Os resultados obtidos estão sumariados na tabela abaixo:

ARQUIVO	HUFFMAN		LZW		LZW-MODIFICADO	
	IR	TRT	IR	TRT	IR	TRT
manual	38.60	115.57	40.37	58.00	49.41	55.60
apendice	40.70	5.32	51.16	2.75	51.16	2.78
c	32.10	166.89	39.73	74.85	45.77	74.53
iltd	26.10	61.37	26.01	28.29	40.25	26.92
backup	32.80	1977.99	14.47	1014.02	48.72	928.92

Pode-se observar que quanto mais "redundâncias localizadas" apresentarem os arquivos (iltd, backup) melhor será o índice de compressão de LZW-MODIFICADO frente aos demais. Observa-se também que em arquivos onde a redundância é praticamente constante (manual, c) a diferença do índice de compressão de LZW e LZW-MODIFICADO é pequena, contudo ambos alcançam índices melhores que HUFFMAN. E finalmente, arquivos pequenos (como apêndice, com aproximadamente 10K) não são suficientes para encher a TS, e portanto os índices alcançados por LZW e LZW-MODIFICADO são os mesmos.

Para os arquivos considerados, o algorítmo LZW-MODIFICADO apresenta um ganho médio geral no índice de Redução de 39.8% sobre HUFFMAN e 65.8% sobre LZW. Quanto ao Tempo de Resposta Total, os ganhos médios gerais foram 52.8% e 3.3% respectivamente.

Os gráficos abaixo, representam a variacão do índice de Reducão com o tamanho de arquivos. O Indice de Reducao do arquivo backup foi medido em vários pontos, para obter os valores indicados no gráfico 1. O gráfico 2 ilustra o Índice de Reducão obtido em um arquivo resultante de sucessivas concatenacões dos primeiros 10K bytes do arquivo apêndice.

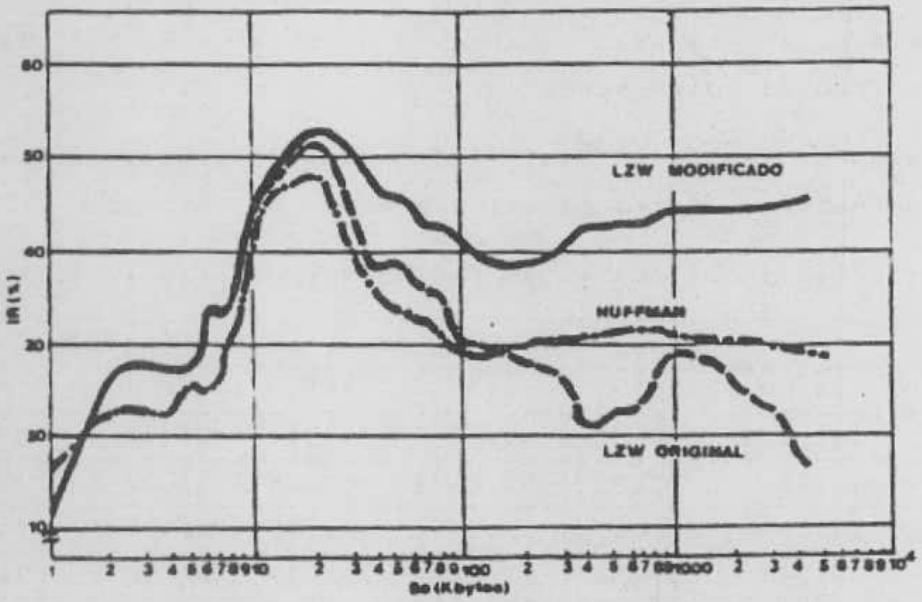

Gráfico 1 - (IR x S_o) Arquivo backup

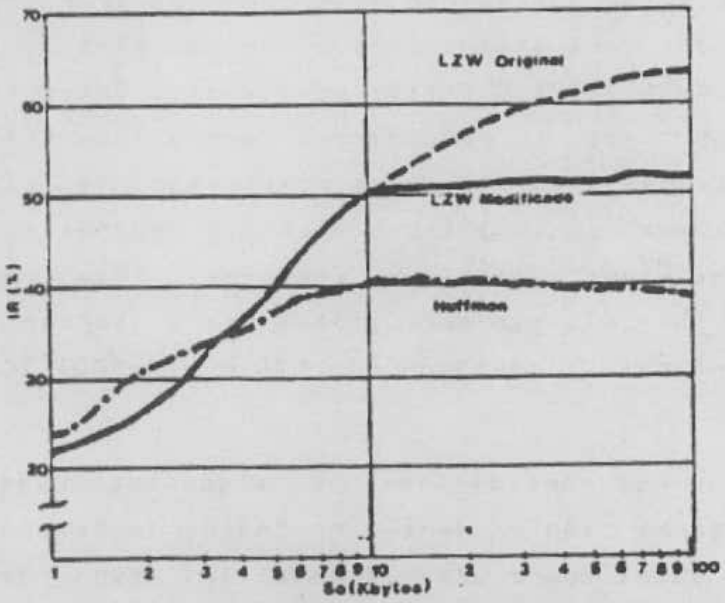

Gráfico 2 - (IR x S_o) Concatenacões sucessivas dos primeiros 10K bytes do arquivo apêndice

O arquivo gerado pela concatenação de blocos iguais (gráfico 2), favorece a compressão LZW, mas este não é um caso de muito interesse na prática. O Índice de Redução para HUFFMAN, nesse caso, é o mesmo sempre que o tamanho do arquivo é um múltiplo de 10K (desprezando-se o percentual referente à tabela usada na decodificação, que deve ser armazenada junto com as informações comprimidas), já que nesses pontos a distribuição de frequência dos caracteres no arquivo é igual . Arquivos que apresentam tipos de redundâncias distintos em trechos distintos (gráfico 1) favorecem a compressão LZW-MODIFICADO, pois a reinicialização da TS permite que uma novo tipo de redundância seja utilizado.

V - UTILIZAÇÃO DE COMPRESSÃO EM TRANSMISSÃO DE DADOS

A compressão de dados pode de fato reduzir o tempo, e consequentemente os custos, com transmissão de dados, já que após comprimida, a massa de dados a ser transmitida é substancialmente menor. Entretanto temos como desvantagem o aumento de processamento associado com a compressão no lado emissor e com a descompressão no lado receptor, isto é, há um "trade-off" entre a redução dos tamanhos dos arquivos e o tempo de processamento em CPU para compressão/descompressão. Por outro lado, o canal de comunicação pode ser um gargalo na transmissão de dados, e neste caso a compressão e a descompressão podem ser processadas paralelamente à transmissão.

Vejamos um exemplo prático. Sob o mesmo ambiente descrito na seção IV, foram feitas transmissões de arquivos, em uma linha de comunicação trabalhando a 9600 bauds, utilizando as versões 1.1 e 2.1 do software de comunicação AGIX[1]. A tabela abaixo, onde os tempos são expressos em segundos e as velocidades em bytes por segundo, ilustra os resultados obtidos.

ARQUIVO	VERSÃO 1.1 (sem comp.)		VERSÃO 2.1 (com comp.)	
	TEMPO	VELOCIDADE	TEMPO	VELOCIDADE
manual	401.22	667	166.53	1607
apêndice	16.73	667	7.56	1475
c	521.66	667	157.94	2203
iltd	183.72	667	134.81	909

1. AGIX é um produto da Sciencia Informática e Tecnologia e da Infocon Software

A versão 1.1, que não utiliza compressão de dados, apresenta uma velocidade de transmissão constante, aproximadamente igual a 667 bytes por segundo. Já a versão 2.1, que utiliza nossa implementação do algorítmo LZW, apresenta uma velocidade de transmissão variável, dependendo do percentual de compressão obtido em cada arquivo. Os arquivos maiores e com maior taxa de compressão (manual, c) conseguem uma velocidade de transferência maior que os arquivos pequenos e/ou com baixas taxas de compressão (apêndice, iltd). Para os arquivos analisados, a versão 2.1 apresentou uma velocidade de transmissão média de 1548 bytes por segundo. Esta velocidade é superior à própria velocidade da linha (1200 bytes/s, desprezando-se o overhead). Este aparente absurdo se deve ao fato do cálculo da velocidade de transferência ser feito sobre o tamanho original dos arquivos e não sobre a quantidade de informação que realmente passou no canal de comunicação.

Hamaker, em [HAMA 86], faz comentários sobre ganhos em processamento de aplicações usando compressão em situações de solicitações frequentes de operações de entrada/saída e em sistemas de tempo compartilhado.

VI - CONCLUSÃO

A simples modificação aqui sugerida é bastante adequada para aplicações em que a informação a ser comprimida apresenta muita variação no tipo de redundância presente, permitindo que o algorítmo se readapte aos novos tipos de redundâncias encontrados. Backup de discos, por exemplo, envolve uma grande quantidade de informação apresentando geralmente características distintas, sendo portanto um exemplo pratico da utilidade do algorítmo discutido.

A implementação simples também é outro aspecto que deve ser lembrado, além da característica de ser um algorítmo em "um passo", permitido portanto o uso em sistemas "on-line".

Um dos principais problemas existentes neste algoritmo, é que a perda de sincronização entre emissor e receptor impossibilitará a recuperação da mensagem original enviada. Este fato

merece atencão especial quando se trata de transferência de informacão comprimida em canais de comunicacão suceptíveis a erros.

Alguns pontos sobre LZW-MODIFICADO não foram ainda suficientemente explorados, entre eles destacamos:

a) A reinicialização da TS não tem necessariamente que descartar todos os símbolos ja inseridos; alguns símbolos poderiam ser mantidos a fim de diminuir a queda temporária da capacidade de compressão do algoritmo.

b) Desenvolvimento de um modelo estatístico para avaliar o desempenho do algoritmo em funcão das características dos arquivos.

REFERENCIAS

[BASS 86] - M. A. Bassiouni e B. Ok. "Double Encoding - A Technique for Reducing Storage Requirement of Text". Inform. Systems. Vol. 11. No. 2. 1986. pag. 177-184.

[BELL 86] - T. C. Bell. "Better OPM/L Text Compression". IEEE Trans. on Comm.. Vol. Com-34. No. 12. Dez. 1986. pag. 1176-1182.

[BENT 86] - J. L. Bentley, D. D. Sleator, R. E. Tarjan e V. K. Wei. "A Locally Adaptive Data Compression Scheme". Comm. of ACM. Vol. 29. No. 4. Apr. 1986. pag. 320-330.

[CLEA 84] - J. G. Cleary e I. H. Witten. "Data Compression Using Adaptive Coding and Partial String Matching". IEEE Trans. on Comm.. Vol. Com-32. No. 4. Apr. 1984. pag. 396-402.

[HAMA 86] - D. W. Hamaker. "Data Compression". Comm. of ACM. Vol. 29. No. 3. Mar. 1986. pag. 173.

[HUFF 52] - D. A. Huffman. "A Method for the Construction of Minimum Redundancy Codes". Proceedings of de IRE 40. Set. 1952. pag. 1098-1101.

[LANG 83] - G. G. Langdon e J. J. Rissanem, "A Double-Adaptive File Compression Algorithm", IEEE Trans. on Comm., Vol. Com-31, No. 11, Nov. 1983, pag. 1253-1255.

[REGH 81] - H. K. Reghbatti, "An Overview of Data Compression Techniques", Computer, Vol. 14, No. 4, Apr. 1981, pag. 71-76.

[WELC 84] - T. A. Welch, "A technique for high performance data compression", Computer, Vol. 17 No. 6, 1984, pag. 8-19.