Técnicas de compressão não destrutiva: Estado da arte

Gabriel Fernandes, Maria Dias, Pedro Rodrigues

Departamento de Engenharia Informática, Universidade de Coimbra, Portugal

gabrielf@student.dei.uc.pt, mddias@student.dei.uc.pt, pedror@student.dei.uc.pt

***Abstract*- Devido ao crescente número de utilizadores de meios informáticos, a cada dia que passa é gerada mais informação que circula pela internet e/ou que é armazenada nos nossos dispositivos pessoais. Verificamos então que se torna importante encontrar formas de codificar a informação por forma a compactá-la facilitando a sua transmissão. Neste documento fizemos uma breve abordagem da eficiência de algoritmos de compressão sendo as nossas conclusões suportadas por estudos feitos sobre o Silesia Corpus. Concluímos que existem vários algoritmos mas que a nossa escolha deve ser feita tendo em conta o local da aplicação do algoritmo e as limitações do equipamento compressor.**

***Palavras chave*-** Algoritmos de compressão não destrutivos, Codificação, Silesia Corpus,Taxa de compressão, Velocidade de compressão/descompressão

I. INTRODUÇÃO

Os algoritmos de compressão de informação foram criados devido à necessidade de resolver problemas relacionados com a transferência de informação através da internet. A transmissão de informação nos canais existentes não atingia velocidades aceitáveis, devendo-se ao facto da largura de banda disponível não ser suficientemente grande para transmitir a informação em tempo útil. Assim, foi necessário diminuir o tamanho da informação, usando para o efeito técnicas de compressão.

Existem dois tipos de compressão: destrutiva e não destrutiva. A principal diferença entre ambos reside no facto de um permitir comprimir sem qualquer perda de informação (codificação não destrutiva), e outro leva a perda de informação no processo de compressão (compressão destrutiva). Enquanto que o primeiro método de compressão é baseado em técnicas que visam a reorganização da fonte de dados original através do uso de várias ferramentas matemáticas e estatísticas como a codificação entrópica, o segundo consegue obter uma compressão dos dados removendo bits que são desnecessários, reduzindo consequentemente o tamanho do ficheiro. Podemos assim concluir que após a descompressão não é possível obter o ficheiro inicial utilizando este método. A compressão de ficheiros é possível devido à redundância e dependência estatística dos símbolos do ficheiro fonte.

Neste documento vamos analisar as principais etapas de compressão, os algoritmos mais usados e as combinações mais usuais destes.

II. ETAPAS DA CODIFICAÇÃO

Como referimos anteriormente, a compressão da informação de forma não destrutiva só é possível devido à redundância (informação repetida, que pode ser escrita de forma reduzida, recorrendo a uma forma de representação diferente) e dependência estatística (o símbolo seguinte está estatisticamente ligado ao símbolo anterior, por exemplo, numa imagem se um pixel está a preto, é muito provável que o seguinte também esteja).

Hoje em dia existem diversos algoritmos de compressão não destrutiva e, aquando da escolha de qual usar, deve ter-se em consideração múltiplos fatores, nomeadamente: taxa de compressão e tempo de compressão/tempo de descompressão.

As fórmulas de cálculo de cada um encontram-se explicitadas abaixo:

$$Compression\ Ratio = \frac{Uncompressed\ file\ size}{Compressed\ file\ size}$$

$$Decompression\ Speed = \frac{Compressed\ file\ size}{Time\ to\ decompress}$$

$$Compression\ Speed = \frac{Uncompressed\ file\ size}{Time\ to\ compress}$$

Taxas de compressão mais elevadas, normalmente levam a um maior uso da capacidade de processamento dos dispositivos que são usados quer para a compressão, quer para a descompressão. Assim sendo, também o tipo de dispositivos utilizados pesam na escolha do algoritmo de compressão a utilizar.

Na compressão dos dados são percorridas várias etapas sendo utilizados vários algoritmos em cada uma delas, algoritmos esses que vão ser mais detalhados na próxima parte. Mas, de uma forma geral a primeira etapa consiste na transformação da fonte através de algoritmos como os Predictors e o BW (Burrows - Wheeler). Através desta transformação conseguimos estabelecer uma ordem na informação fornecida o que vai melhorar a eficiência das etapas seguintes. A segunda etapa consiste na análise da dependência estatística e tratamento dos dados gerando agrupamentos e tirando partido da redundância da fonte explorada anteriormente. São utilizadas técnicas como a

codificação por dicionário, modelação através de cadeias de Markov e algoritmos como o Run Length Encoding. Com estes agrupamentos já obtidos podemos partir então para a etapa final! A etapa final corresponde à representação desses agrupamentos utilizando técnicas de codificação entrópica como por exemplo codificação aritmética e árvores de Huffman que se servem dos agrupamentos gerados anteriormente e da sua probabilidade para os melhor representar servindo-se do menor número de bits possível reduzindo assim o espaço ocupado pela informação original.

III. ALGORITMOS USADOS

1ª Etapa: Redundância da Fonte

Preditors

Este tipo de algoritmos tem como objectivo fazer uma transformação da fonte através de previsões e de relações entre elementos adjacentes da mesma fonte. Algoritmos como os de Delta Encoding são exemplo de bons Preditors. De forma sucinta o Delta Encoding consiste em exprimir uma fonte de dados através das semelhanças que elementos têm com os anteriores ou seja olhando para os dados e exprimindo-os através de variações.

BW (Burrows -Wheeler)

Este algoritmo é utilizado para aumentar a redundância da fonte de informação.

Funcionamento do algoritmo:

- S é a fonte de informação, que contém um caracter cuja função é marcar o final de S (EOF), S tem tamanho n (algumas versões deste algoritmo já não precisam do EOF).
- S é submetida a um total de n rotações (reversões) obtemos, assim, uma matriz de strings. Depois a matriz n por n com as strings obtidas a partir de S são ordenadas por ordem lexical ou numérica.
- Entre todas as linhas dessa matriz existirá uma igual a S. Selecionamos então a última coluna da matriz ordenada e o índice da string da matriz igual a S, o que nos permite reverter o processo posteriormente.

2ª Etapa: Dependência estatística

LZ77

Este algoritmo utiliza umas search window, de tamanho n (2 KiB, 4 KiB ou 32 KiB por exemplo), nesta são mantidos os últimos n símbolos enviados/recebidos e uma look-ahead window de determinado tamanho.

- Aquando do envio de mais símbolos, é efetuada uma procura na janela pela maior sequência começada pelo símbolo da sequência a enviar, a maior correspondência é então codificada da seguinte forma {<offset>, <length>, <next_symbol>}.

Offset: número de símbolos que têm de ser percorridos para trás na janela deslizante até chegar ao símbolo onde é começada a cópia.

Length: Número de símbolos a ser copiado da janela deslizante;

Next_symbol: O próximo símbolo que tem de ser enviado e que não se encontrava presente na sequência copiada da janela deslizante.

- Assim, dependendo do tamanho da janela a ser copiada, esta codificação pode ser deveras eficiente no envio de informação.
- Se não for encontrada uma sequência para copiar no search window, o offset e a length são 0,diminuindo a eficiência do algoritmo

LZW

LZW é uma versão do LZ78, recorrendo também a um dicionário. Um dos diferenciais do LZW para o LZ78 está na maneira como o dicionário é inicializado, no LZW este é inicializado com os caracteres do alfabeto da fonte, enquanto que no LZ78 este encontra-se vazio.

O dicionário é construído de forma adaptativa da seguinte forma:

- Procura-se no dicionário qual é a maior cadeia que mais caracteres têm iguais ao que se pretende enviar;
- Envia-se o índice do dicionário respectivo à melhor correspondência;
- Adiciona-se mais uma entrada no dicionário com o código da mensagem enviada + caracter seguinte da fonte de informação.

RLE

Run length encoding, consiste em reduzir uma sequência de símbolos iguais a uma sequência de 3 símbolos. Desses 3 símbolos fazem parte o símbolo que se repete na fonte de informação, o número de vezes que o símbolo é repetido e um símbolo especial que permite diferenciar esta forma de comprimir informação de uma sequência idêntica na fonte que em nada tem a ver com o reduzir de uma sequência.

3ª Etapa: Codificação Entrópica

Huffman Coding

Algoritmo que tira partido da probabilidade de ocorrência dos símbolos da fonte de informação, atribuindo aos que mais se repetem códigos de comprimento menor, enquanto que os que menos se repetem são codificados com códigos mais extensos.

Os códigos são obtidos por via da construção de uma árvore a partir das folhas agrupando sempre os que têm menor probabilidade e, quando há mais do que dois com a mesma probabilidade e esta é a mais baixa, escolhem-se os dois que estão mais longe da raiz (assim garantimos que a variância dos códigos é mínima). Depois,

continua-se este raciocínio até que a probabilidade depois da soma dos nodos dê 1.

Os códigos resultantes são instantâneos e unicamente decodificáveis (códigos de prefixo).

Arithmetic Coding

Este algoritmo codifica a informação com base na probabilidade de cada símbolo na fonte de informação.

Para codificar o primeiro símbolo da fonte de informação divide-se o intervalo [0, 1] pelos símbolos do alfabeto da fonte (com base na probabilidade de cada um), para o segundo símbolo dividem-se todos os subintervalos criados para o símbolo anterior com base nas probabilidades de cada símbolo e assim sucessivamente, até termos um intervalo que corresponda à fonte a enviar. Depois, basta escolher um valor que pertença a esse intervalo para enviar a fonte, normalmente escolhe-se o valor médio do intervalo.

Ainda são utilizados outros algoritmos de compressão que não vamos detalhar. O LZMA e o PPMd são dois desses algoritmos. O primeiro é um algoritmo da família Lempel-Ziv que utiliza modelação de Markov na codificação do elementos do dicionário, o segundo utiliza também esta modelação fazendo uma previsão probabilística através de uma cadeia de ordem n.

IV. COMBINAÇÕES MAIS HABITUAIS

1. Deflate

Usando o Deflate a compressão é alcançada em duas etapas principais: associação e substituição de séries de bytes (usando para o efeito o LZ77 com janela deslizante de 32KB e buffer de look-ahead de 258 bytes) e ainda substituição de símbolos tendo em conta a sua frequência de uso (usando codificação de Huffman).

O Deflate consiste numa série de blocos com um cabeçalho de 3 bits. O primeiro bit dá informação sobre se existem (bit a 1) ou não (bit a 0) mais blocos a seguir. O segundo e terceiro bit, por outro lado, dão informação sobre o tipo de codificação usado no bloco. Recebendo 00 representa um bloco que contém uma sequência de bits (que não foram repetidos) com tamanho entre 0 e 65.535 bytes, 01 representa um bloco codificado usando uma árvore de Huffman pré acordada, 10 representa um bloco comprimido com uma tabela de Huffman e 11 representa um bloco reservado que não pode ser usado.

2. PNG

O processo de compressão PNG envolve 2 partes, sendo elas: pré compressão (usando Predictor) e a conversão propriamente dita (usando Deflate). Na pré compressão da imagem é usado um *filter method* que é aplicado a toda a imagem e ainda um *filter type* que é aplicado a cada uma das linhas. O *filter method 0* (único método atualmente definido para o PNG) prevê a cor de um pixel com base nas cores dos pixels ao seu redor e subtrai ao valor real do pixel o valor previsto. Na compressão propriamente dita é usado o Deflate com uma janela de 32KB. Na compactação usando deflate pode ser utilizado um algoritmo de Huffman de comprimentos fixos ou adaptativo (personalizado) sobre os blocos de dados gerados pela codificação anterior através de LZ77.

V. CONCLUSÃO

Após o estudo dos vários algoritmos e das combinações mais usuais chegamos a conclusão que alguns algoritmos têm vantagens em termos de velocidade de compressão como por exemplo o Deflate mas que têm défices no rácio de compressão. Outros algoritmos como os da família Lempel-Ziv (LZMA que deriva do LZ77 por exemplo) atingem rácios de compressão e velocidades bastante bons o que os tornam bastante versáteis. E algoritmos como os da família PPmd atingem uma elevada compressão mas são muito lentos e exigem bastantes recursos de uma máquina. Como em tudo cada caso é um caso e temos de escolher o algoritmo que nos traz maior vantagem para a situação em que o queremos aplicar. Reparamos que os algoritmos são utilizados de uma forma encadeada para que se consiga obter melhores resultados. Softwares de compressão modernos implementam estes algoritmos!

Algoritmo de compressão de dados	Taxa de compressão (média)	Velocidade de compressão (média) MB/s	Velocidade de descompressão (média) MB/s
Deflate	3.825	9.732	91.342
LZMA	5.88	1.669	57.742
PPMd	6.095	6.457	5.748

Tab1. Aplicação destes algoritmos no Silesia Corpus

VI. BIBLIOGRAFIA

[1] https://en.wikipedia.org/wiki/Predictive_analytics

[2] https://en.wikipedia.org/wiki/Burrows–Wheeler_transform

[3] https://en.wikipedia.org/wiki/LZ77_and_LZ78

[4] https://en.wikipedia.org/wiki/Lempel–Ziv–Welch

[5] https://pt.wikipedia.org/wiki/Codificação_run-length

[6] https://en.wikipedia.org/wiki/Huffman_coding

[7] https://en.wikipedia.org/wiki/Arithmetic_coding

[8] https://en.wikipedia.org/wiki/Portable_Network_Graphics

[9] Modern Lossless Compression Techniques: Review,Comparison and Analysis

[9] https://en.wikipedia.org/wiki/Portable_Network_Graphics

[10]https://en.wikipedia.org/wiki/DEFLATE

[11] https://www.w3.org/TR/REC-png.pdf

[12]https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Markov_chain_algorithm

[13]https://en.wikipedia.org/wiki/Prediction_by_partial_matching