Estado da Arte de algoritmos de compressão lossless para texto e imagem

Afonso Serra
Dept. Engenharia Informática (FCTUC)
Coimbra, Portugal
uc2018279154@student.uc.pt

Dinis Carvalho
Dept. Engenharia Informática (FCTUC)
Coimbra, Portugal
uc2018278118@student.uc.pt

João Teixeira
Dept. Engenharia Informática (FCTUC)
Coimbra, Portugal
joao.teixeira@student.uc.pt

***Abstract*—Neste artigo apresentaremos o atual estado da arte relativo aos Codecs utilizados na compressão de imagens e de texto, bem como as metodologias usadas pelos mesmos, com ênfase nos procedimentos mais eficientes. O foco principal serão os métodos de compressão não destrutivos.**

Keywords—Codec, Lossless, Image compression, Text compression

I. INTRODUÇÃO

A Teoria da Informação pode ser usada com bastante eficácia na compressão de dados. O contexto que vamos estudar é a de utilização de CODEC (relação de Coder/Decoder) na compressão de texto e imagens monocromáticas. Aqui, iremos apenas discutir técnicas *lossless:* significa isto que, ao contrário dos métodos *lossy*, o resultado da descompressão não verificará qualquer perda de dados originais.

Na fase atual de desenvolvimento em que nos encontramos, com a resolução (e, consequentemente, o tamanho) das imagens continuamente a aumentar, é importante o desenvolvimento das tecnologias de compressão de forma a, por exemplo, a qualidade das imagens não estar limitada pela largura de banda de uma ligação. Tomemos como exemplo a NASA Deep Space Network. As ligações estabelecidas por esta rede chegam a tomar valores de velocidade de transmissão tão baixos como 500 bits por segundo. Isto significa que uma imagem capturada pelo Telescópio Hubble com aproximadamente 100MB poderia demorar até 444 horas a chegar à Terra. Neste panorama, é fácil compreender a importância da compressão sem perda de informação.

Para aqui falarmos sobre o estado da arte deste ramo da ciência computacional, ecolhemos algumas ténicas utilizadas pelos mais vanguardistas Codecs da atualidade. Como sabemos que estamos de facto a falar de programas novos e experimentais? Tirámos partido dum *benchmark construído* por um especialista na área, Matt Mahoney [1]. Neste *ranking* observamos vários Codecs a serem comparados com base na sua performance diante de uma massa de dados, igual para todos os testes.

Cada codec tira pratido de uma ou mais técnicas de compressão. Entre as várias que existem, falaremos aqui de quatro algoritmos: CM (Context Mixing), PPM (Prediction By Partial Match), BWT (Burrows – Wheeler Transform) e LZ (Lempel – Ziv Transform, algoritmo 77)

Porque é que escolhemos estes?

Segundo o nosso trabalho de pesquisa, ficámos a saber que estes são os algoritmos com melhor output nos seguintes critérios: Taxa de compressão, Velocidade de compressão/descompressão e Custo de memória. Cada técnica terá resultados ótimos num parâmetro; O algoritmo LZ terá particular detalhe no que toca apenas à compressão de imagens. Serão estas as questões que servirão de guia para compararmos cada um destes algoritmos, bem como a sua prática.

II. CRITÉRIOS

Segue-se uma breve descrição para cada um dos critérios a seguir:

1. A taxa de compressão é definida por

$$\text{TC} = \frac{L_{orig} - L_{comp}}{L_{orig}} \times 100$$

onde L(orig) representa o comprimento do ficheiro original e L(comp) o comprimento do ficheiro comprimido.

2. As velocidades de compressão e descompressão correspondem ao tempo necessário para o algoritmo de compressão / descompressão terminarem o seu trabalho.

3. O custo de memória refere-se estritamente à memória necessária para o algoritmo poder correr. É uma das razões para programas como o WinRAR exigirem certas especificações de hardware para poderem ser utilizados. O estado da arte da compressão lossless é ainda mais exigente: o programa *cmix*, que será falado mais à frente, exige um mínimo de 32Gb para poder correr (ainda que para só um ficheiro).

III. FUNCIONAMENTO

Segue-se uma breve explicação para cada uma das técnicas:

1. PPM – Prediction by Partial Matching

Esta técnica vai tentar realizar predições sobre o símbolo seguinte, analizando o ficheiro símbolo por símbolo, utilizando cadeias de Markov. Os símbolos únicos serão tabelados e organizados numa espécie de *ranking* onde os mais frequentes estão no topo e os menos frequentes em posições inferiores, que irão intuitivamente representar os mais prováveis no topo e os menos prováveis no fundo – note-se que um *símbolo* pode na verdade ser uma cadeia que será

tratada como uma unidade por motivos de redundância. À medida que o programa avança, esse tabelamento é feito e atualizado, e as predições futuras serão feitas com base nessa espécie de lista metamórfica. Este método necessitará de pouco tempo para completar quer a compressão quer a descompressão e vai utilizar pouca memória, no entanto a taxa de compressão que devolve não será a melhor no que toca ao estado da arte, ainda que com resultados relevantes, na medida em que a sua utilização é viável para o utilizador comum, como se comprova com o parágrafo que se segue.

Dois exemplos de programas que utilizam este método são o 7zip e o WinRAR, dois dos compressores mais populares do mundo. [1]

2. Context Mixing

Esta abordagem é que atualmente das que produz as melhores taxas de compressão já alguma vez testemunhadas. O modo de operação deste tipo de compressão é o seguinte: a compressão é feita bit por bit, e o bit seguinte é previsto pela aplicação de um ou mais modelos estatísticos. Segue-se uma explicação superficial deste método:

Fundamentalmente, deseja-se estimar a probabilidade P(X | A,B,C,…,Z) onde X é o evento que descreve a probabilidade do próximo símbolo assumir um dado valor, e A,B,C,…,Z são diferentes *contextos*, cada um representativos de diferentes modelos estatísticos. Acontecerá, durante o *runtime* dos programas que usufruem desta técnica, que os contextos isolados já terão ocorrido vezes suficientes para se estimar P(X | A), P(X | B), etc. , mas não terão ocorrido em conjunto vezes suficientes para se estimar P(X | A,B,C,…,Z). Este último ponto deve-se várias vezes a falta de recursos, nomeadamente tempo e memória.

Para uma mais fiel predição, são utilizadas duas formas de *misturar (mixing)* os contextos (que foram independentemente estimados):

1) Linear Mixing

Sob cada contexto, são contados os números de 0's e 1's do espaço de dados em análise, e deles se realizam estimativas probabilísticas com maior peso para o contexto que ocorrer mais vezes. Se o contexto A ocorrer mais vezes que o contexto B, então a probabilidade do próximo bit será calculada tal que P(X|A) terá mais peso que P(X|B) – se A colecionou mais informação, A é a que nos vai dizer mais sobre o próximo bit. [2]

2) Logistic Mixing

As predições são transformadas para o *domínio logístico* [2] log(p/(1-p)), através de redes neurais, e os resultados probabilísticos das predições serão mais próximos de 0 ou de 1 – ou seja, as decisões são feitas com maior eficácia.

Os contextos a utilizar antes do algoritmo começar efetivamente a funcionar vão depender do tipo de ficheiro. Um ficheiro de texto não utilizará um conjunto de contextos igual aos que seriam utilizados para um ficheiro de imagem – e dentro de cada tipo de ficheiro, existem formas diferentes para cada extensão (em imagem: .tiff, .gif, .jpeg…)

O mais importante a notar acerca desta abordagem é que é a que devolve os melhores resultados relativamente à taxa de compressão – o *cmix v18* é neste momento o codec com maior taxa de compressão de sempre – mas o custo vem bipartido: demora quantidades de tempo demasiado grandes e consome demasiada memória: o autor desta aplicação recomenda que seja apenas utilizada em sistemas com pelo menos 32Gb de memória RAM.

A família de codecs *PAQ*, desenvolvida por M. Mahoney e A. Rhatushnyak, usufrui desta técnica e tem produzido resultados cada vez mais satisfatórios [4]. O *cmix*, acima falado, tem por base estes codecs.

3. BWT – Burrows-Wheeler transform

A transformada Burrows-Wheeler por si só não atua como compressor, tal como o nome indica é meramente uma forma de transformar os dados, de os *preparar* para serem comprimidos com melhor eficiência.

Este processo vai pegar num bloco de informação e vai-lhe aplicar *rotações* sucessivas de modo a descobrir quais os seus símbolos únicos, transformando esse bloco numa versão simplificada de si próprio no ponto de vista quantitativo: contém a mesma informação, mas reestruturada, tal que só as sucessões mais elementares dos símbolos na cadeia apareçam, ordenadamente segundo a ordem no bloco original.

4. LZ77

A transformada Lempel-Ziv deu origem a um dos mais conhecidos algoritmos de compressão de sempre – o LZ77, empregado por um grande número de aplicações.

Este algoritmo funciona através do tabelamento de ocorrências de códigos únicos no ficheiro original, que será depois útil na formação da codificação de todo o ficheiro. Normalmente, o ficheiro comprimido tem entre ¼ e 1/3 do seu tamanho original; no entanto, comprovando o *modus operandi* do algoritmo, caso haja padrões no ficheiro de entrada, o ficheiro comprimido pode vir a ter 1/10 do tamanho original. Um exemplo duma aplicação que usufrui deste algoritmo é o *gzip*, um compressor popular no Linux; também é o algoritmo utilizado nas imagens do tipo .PNG ou .GIF. [5]

IV. Codificação

No caso da transformada Burrows – Wheeler, a aplicação do algoritmo não comprime dados nenhuns por si só, mas "prepara-os" de modo a que possam ser comprimidos com melhores resultados, usando códigos aritméticos e códigos de Huffman. O mesmo acontece com Prediction by partial match, ou seja, poderemos encarar estes métodos como sendo algoritmos de pré processamento. [6]

O LZ77 não necessita de ser conjunto com codificações entrópicas por já realizar o trabalho de compressão. Devido ao facto da tecnica Context Mixing estar relacionada com Prediction By Partial March, nota-se que também irá usufruir de codificação entrópica, consoante a implementação (dado ser um método mais experimental, o codec pode vir com formas de codificação especificas).

Resumir-se-ão brevemente os códigos aritméticos e os códigos de Huffman:

1) A codificação aritmética é um método de codificação que usa uma fração para representar a mensagem original, quando a distribuição estatística dos dados a comprimir é conhecida. No processo de codificação, as probabilidades cumulativas são calculadas e o intervalo é criado no início. Durante a leitura carater a carater da origem, o correspondente intervalo é dividido em subpartes de acordo com as probabilidades do

alfabeto. De seguida, o próximo caracter é lido e o correspondente subintervalo é selecionada. Desta maneira, os caracteres são lidos, de forma repetitiva, até ao final da mensagem ser encontrada. Finalmente, um número deve ser retirado do final do subintervalo como a produção do processo de codificação. Esta vai ser a fração do intervalo. Desta maneira, a mensagem original inteira consegue ser representada por forma de uma fração. Para descodificar a mensagem codificada, o número de caracteres da mensagem original e a probabilidade/distribuição de frequências são precisas.

2) Codificação De Huffman

O Algoritmo de Codificação de Huffman usa a distribuição de probabilidades do alfabeto da fonte para desenvolver códigos de comprimento variável para os símbolos. Códigos mais pequenos são usados para símbolos com maior probabilidade e códigos maiores são atribuídos a símbolos com menor probabilidade. É criada uma árvore binária usando os símbolos como folhas da árvore, de acordo com as suas probabilidades. São propostas duas famílias de Códigos de Huffman: o Algoritmo Adaptativo de Huffman e o Algoritmo Estático. O Algoritmo Adaptativo constrói a árvore enquanto processa os símbolos e calcula as frequências. O compressor e o descompressor têm de estar constantemente sincronizados, pois quando um símbolo é atualizado no compressor também tem de ser alterado no descompressor.

V. Conclusões

Pela comparação das várias técnicas, torna-se claro que o estado da arte deste ramo da teoria da informação está assegurado pelo método PPM: ainda que não seja maximizante das taxas de compressão, ao contrário do método CM, é o que melhor equilibra os critérios que utilizámos neste trabalho. Portanto, o PPM é o mais praticável. Não se reprova, no entanto, o potencial extremo do método CM, unicamente devido às enormes taxas de compressão que tem vindo a apresentar.

References

[1] *Matt Mahoney (2015-09-25). "Large Text Compression Benchmark"*

[2] Mahoney, M. (2005), "Adaptive Weighing of Context Models for Lossless Data Compression", Florida Tech. Technical Report CS-2005-1

[3] http://www.squeezechart.com/

[4] *Mahoney, M. "PAQ8 Data Compression Program"*

[5] Suman M. Choudhary, Anjali S. Patel, Sonal J. Parmar, “Study of LZ77 and LZ78 Data Compression Techniques”

[6] http://compressions.sourceforge.net/Entropy.html

[7] Apoorv Gupta1, Aman Bansal, Vidhi Khanduja, “Modern Lossless Compression Techniques: Review, Comparison and Analysis”

[8] Ms. Shilpa Ijmulwar, Deepak Kapgate, “A Review on - Lossless Image Compression Techniques and Algorithms”

[9] S.R. Kodituwakku, “Comparison of Lossless Data Compression Algorithms for Text Data”

[10] Senthil Shanmugasundaram, “A Comparative Study Of Text Compression Algorithms”

[11] Alistair Moffat, Timothy C. Belly, Ian H. Wittenz, “Lossless Compression for Text and Images”