Estado da Arte

Botelho João
Departamento de Eng. Informática
Universidade de Coimbra
Coimbra, Portugal
uc2019155348@student.uc.pt

Branco Guilherme
Departamento de Eng. Informática
Universidade de Coimbra
Coimbra, Portugal
mbranco@student.dei.uc.pt

Félix Dário
Departamento de Eng. Informática
Universidade de Coimbra
Coimbra, Portugal
dario@student.dei.uc.pt

I. Introdução

Os algoritmos de compressão *lossless* permitem armazenar a mesma quantidade de informação com um menor espaço de armazenamento ou com melhor capacidade de transmissão. Habitualmente aplicam transformadas aos dados para tornar redundâncias na informação mais evidentes, e em seguida exploram essas redundâncias codificando probabilidades ou através de mapeamento.

II. Estado da Arte

Vários *encoders* e transformadas são atualmente usados para compressão *lossless* de dados. Apresentam-se em seguida alguns dos principais algoritmos usados para compressão *lossless* de texto, e programas que os implementam.

A. Encoders

1) Entrópicos

Algoritmos que geram um modelo estatístico dos dados que permita mapeá-los para uma cadeia de *bits*, representando os dados mais frequentes com menos *bits*. [1]

a) Codificação de Huffman

Constrói uma árvore binária dos caracteres a partir das suas probabilidades de ocorrência *P(simbolo)*, usando-a em seguida para gerar um código de prefixo para cada caracter. A árvore deve ser associada ao ficheiro comprimido para permitir a sua descodificação. A codificação de Huffman aproxima obrigatoriamente o número de *bits* necessários para codificar um símbolo, $-log_2 P(simbolo)$, a um número inteiro. Símbolos com altas probabilidades de ocorrência podem ter um valor de *bits*/símbolo inferior a 1, pelo que a sua codificação é pouco eficiente. O impacto desta desvantagem pode ser reduzido por agrupamento de símbolos, mas o alfabeto mais extenso que resulta desse agrupamento leva a outros problemas como uma maior ocupação de memória. [1], [2], [3]

b) Codificação aritmética

Codifica o input com um único número *floating point*, a partir das probabilidades de ocorrência dos caracteres. Os caracteres com maior probabilidade de ocorrência passam a ocupar menos *bits*, enquanto aqueles com menor probabilidade de ocorrência passam a ocupar mais *bits*. A codificação final da totalidade dos dados ocupa um menor número de *bits*. A codificação aritmética tende a obter melhores resultados de compressão que a codificação de Huffman, pois codifica símbolos com probabilidade p num número de bits arbitrariamente próximo de $-log_2 p$. [1], [3]

c) Range Encoding

Generaliza a codificação aritmética, atuando em dígitos de qualquer base e não apenas em bits (base 2). Ao ser aplicado ao nível do *byte*, *Range Encoding* pode ser quase duas vezes mais rápida que codificação aritmética, mas resulta em maior ocupação de memória durante o processamento e numa menor taxa de compressão caso se pretenda a mesma precisão de probabilidades. [4]

2) Dicionário

Algoritmos que tiram partido da repetição do mesmo símbolo ou grupo de símbolos ao longo dos dados, substituindo ocorrências seguintes por uma referência à sua entrada num dicionário. [5]

a) LZ77

Explora a probabilidade de repetição de palavras ou frases. Quando isto acontece, os dados podem ser codificados com um ponteiro para uma ocorrência anterior dentro de um *buffer* ou janela deslizante, acompanhado pelo número de caracteres iguais encontrados. Tem a vantagem de ser muito simples e não requerer informação sobre as características da fonte *a priori*. Os algoritmos derivados do LZ77 compõem a família LZ77. [1], [2]

B. Transformadas

Algoritmos que transformam os dados de forma reversível, tornando-os mais favoráveis à aplicação de técnicas de compressão. [6]

1) Transformada de Burrows-Wheeler (BWT)

Reorganiza uma sequência de caracteres em conjuntos de caracteres semelhantes. É útil antes da aplicação de outras técnicas que comprimam sequências de caracteres repetidos. Tem como desvantagem a impossibilidade da sua aplicação a um *stream* de frases enviadas separadamente, pois a transformada é aplicada a um bloco inteiro. [6]

2) Move To Front (MTF)

Cada símbolo é substituído pelo seu índice numa pilha de "símbolos usados recentemente". No final, os dados foram transformados numa sequência de inteiros, que tendem a ser pequenos se a fonte tiver muitas correlações locais. [6]

C. Outros

1) Run Length Encoding (RLE)

Codifica símbolos repetidos e contíguos em pares "comprimento da cadeia, símbolo". Cada par pode ser posteriormente codificado com outro algoritmo. Apenas é

eficiente em ficheiros com muitos dados repetidos, podendo aumentar o seu tamanho se isto não se verifica. [1], [2]

D. Sistemas

1) bzip2

Este programa realiza múltiplas camadas de compressão, começando pela aplicação de RLE à fonte, seguida de BWT, MTF e uma RLE do resultado. Segue-se codificação de Huffman e termina com codificação unária, codificação Delta e disposição dos símbolos usados num *array*. A descodificação é realizada pela ordem inversa. A *performance* do bzip2 é assimétrica, dado que a codificação tem grandes custos computacionais e de memória pelo uso de BWT, mas a descodificação é muito rápida [7]. O uso inicial de RLE na fonte tende a não ser eficiente e é um ponto de criticismo do *bzip2*. [8]

2) rzip

Este programa codifica inicialmente dados duplicados usando um dicionário com um *buffer* de 900MB, várias ordens de magnitude maior que o de outros programas, procedendo à compressão usando bzip2 como *backend*. A capacidade de aproveitar redundância a longas distâncias leva a uma compressão mais rápida, pois a informação processada por bzip2 foi previamente reduzida. O rzip atinge taxas de compressão elevadas para ficheiros grandes, sendo a sua principal desvantagem a elevada ocupação de memória durante o processamento. [9]

3) Deflate

Aplica codificação de Huffman e codificação por dicionário estilo LZ77. A construção do dicionário pode procurar e criar entradas para sequências longas de caracteres se a taxa de compressão for valorizada, ou minimizar o número de entradas se a velocidade for valorizada. Algumas implementações de Deflate têm a opção de usar árvores de Huffman pré-definidas do próprio algoritmo, em vez de criar árvores que requerem espaço de armazenamento adicional. [10], [11]

4) Algoritmo Lempel-Ziv-Markov (LZMA)

É composto por um algoritmo de dicionário com janela deslizante baseado no LZ77, seguido de *Range Encoding* e árvores binárias para codificação entrópica. Os dicionários usados têm tamanho até 4GiB, muito superior aos de outros algoritmos baseados em LZ. LZMA atinge altas taxas de compressão e tem *performance* assimétrica, dado que a descodificação é mais rápida que a codificação. [12], [13]

REFERÊNCIAS

[1] N. Sharma, J. Kaur e K. Kaur, "A Review on various Lossless Text Data Compression Techniques," *Research Cell: An International Journal of Engineering Sciences,* vol. 2, pp. 58-63, December 2014.

[2] S. Shanmugasundaram e L. Robert, "A Comparative Study Of Text Compression Algorithms," *ICTACT Journal on Communication Technology,* vol. 02, nº 04, pp. 444-451, 2011.

[3] T. Bell, I. Witten e J. Cleary, "Modeling for text compression," *ACM Computing Surveys,* vol. 21, nº 4, pp. 557-591, 1989.

[4] T. B. Terriberry, "On the Overhead of Range Coders," 7 3 2008. [Online]. Available: https://people.xiph.org/~tterribe/notes/range.html#T1. [Acedido em 24 11 2021].

[5] A. Jain and K. I. Lakhtaria, "Comparative Study of Dictionary based Compression Algorithms," *IJCSNS International Journal of Computer Science and Network Security,* vol. 16, no. 2, pp. 88-92, 2016.

[6] R. Rădescu, "Transform Methods Used in Lossless Compression of Text Files," *Romanian Journal of Information Science and Technology,* vol. 12, nº 1, pp. 101-115, 2009.

[7] "bzip2 and libbzip2, version 1.0.8," Sourceware.org, [Online]. Available: https://sourceware.org/bzip2/manual/manual.html. [Acedido em 21 Nov 2021].

[8] "bzip2," Bzip.org, [Online]. Available: http://www.bzip.org/. [Acedido em 21 Nov 2021].

[9] "rzip," Rzip.samba.org, [Online]. Available: https://rzip.samba.org/. [Acedido em 21 Nov 2021].

[10] L. P. Deutsch, "DEFLATE Compressed Data Format Specification version 1.3," 03 1996. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1951. [Acedido em 25 11 2021].

[11] A. Feldspar, "An Explanation of the Deflate Algorithm," 23 08 1997. [Online]. Available: https://zlib.net/feldspar.html. [Accessed 25 11 2021].

[12] "7z Format," 7-zip.org, [Online]. Available: https://7-zip.org/7z.html. [Acedido em 24 Nov 2021].

[13] I. Pavlov, *LZMA specification (DRAFT version),* 2015.