

Estado da arte dos codecs não-destrutivos (*lossless*) de imagem

Carlos Jordão

Departamento de Engenharia
Informática

Faculdade de Ciências e Tecnologias
da Universidade de Coimbra
Coimbra, Portugal
cejordao@student.dei.uc.pt

Francisco Carreira

Departamento de Engenharia
Informática

Faculdade de Ciências e Tecnologias
da Universidade de Coimbra
Coimbra, Portugal
mcarreira@student.dei.uc.pt

Rodrigo Machado

Departamento de Engenharia
Informática

Faculdade de Ciências e Tecnologias
da Universidade de Coimbra
Coimbra, Portugal
ramachado@student.dei.uc.pt

Abstract — A compressão de imagens de forma não destrutiva é um tema já bastante estudado e trabalhado ao longo das últimas décadas, como resultado da popularização da internet. A necessidade de acesso rápido e de armazenamento fácil impulsionou a criação de vários algoritmos e codecs de imagem, uns melhores em comprimir e descomprimir rapidamente, outros a valorizar mais o rácio de compressão, enquanto alguns tentam encontrar um equilíbrio. Para perceber o estado atual da compressão de imagens de forma não-destrutiva, são estudadas as diferentes componentes algorítmicas e os codecs existentes para este efeito, descrevendo o método e os resultados obtidos.

Keywords— *compressão não-destrutiva, codecs de imagem, lossless, velocidade de compressão, algoritmos.*

INTRODUÇÃO

No mundo atual, toda e qualquer informação, útil ou não, está prontamente disponível com a proliferação massiva de dispositivos móveis como os *smartphones*, os *laptops*, e com o desenvolvimento de redes móveis de alta velocidade – 4G ou até 5G. Toda essa informação é o resultado de quantidades massivas de dados que têm de ser armazenadas e facilmente acedidas.

Em 2020, o vírus SARS-CoV-2 obrigou as pessoas a permanecer em casa, o que levou a que muitas das atividades, como o trabalho, o ensino ou o entretenimento, passassem a ser feitas com o recurso da internet. Mesmo com o avanço tecnológico das redes, este uso massivo seria um teste exigente das redes e das soluções de armazenamento de dados caso as transmissões não sofressem nenhum tipo de compressão. Neste trabalho estudamos a compressão não-destrutiva de imagens.

Geralmente uma imagem de uso pessoal não necessita de muitos dos dados originais, pois existem codecs de compressão *lossy* que reduzem bastante o tamanho destas, reconstruindo uma imagem com fidelidade suficiente. No entanto, existem campos em que a reprodução fiel de uma imagem é necessária, como na medicina – por exemplo, uma radiografia ou uma ressonância magnética, em que todo o detalhe pode ser importante. O armazenamento de imagens de alta fidelidade torna-se mais eficiente e barato quando se consegue uma compressão que não destrua os dados originais, assim como o envio destas pela internet se torna mais rápido, o que poderá ser crítico para o exercício da medicina remota.

Para estudar métodos atuais de compressão *lossless* iremos primeiramente rever algoritmos de compressão de

dados; posteriormente analisaremos codecs populares, que transformações aplicam e algoritmos usam.

TRANSFORMADAS DE DADOS

As transformações de dados são úteis para agrupar símbolos semelhantes e/ou descorrelacionar dados, de forma a diminuir a entropia e a conseguir um rácio de compressão mais elevado. Para ser útil, a transformada tem de admitir uma inversa, para recuperar os dados originais.

Transformada de Burrows-Wheeler (BWT)

A transformada cria um N-1 sequências a partir de shifts cíclicos de uma sequência original de tamanho N, resultando em N-1 novas sequências. Estas e a original são então ordenadas lexicograficamente, sendo o resultado da transformada a última coluna das sequências ordenadas.

O processo inverso é um pouco mais moroso, mas concatenando o resultado e uma versão organizada lexicalmente N vezes recria a lista de shifts cíclicos criada. A sequência original pode ser então identificada usando um símbolo de escape, por exemplo um EOF no final da sequência.

Esta transformada tende a agrupar símbolos iguais, criando sequências de repetições que podem ser aproveitadas por outras transformadas ou algoritmos de compressão.

Move-To-Front (MTF)

Geralmente é precedido por uma transformada de Burrows-Wheeler e procedido por um algoritmo de compressão *Run-Length Encoding* (RLE) ou outro codificador entrópico.

Com o auxílio de um alfabeto, substitui cada símbolo pela sua posição no alfabeto um a um, mas sempre que a substituição ocorre esse símbolo é movido para o início do alfabeto.

Em sequências de dados com repetições, a transformada irá criar sequências de 0s, das quais um algoritmo como o RLE beneficia.

Sendo uma transformada de baixa complexidade computacional e rapidez de execução, o seu uso em processos de compressão é popular.

Delta filters

A transformada mais simples que permite descorrelacionar dados é a codificação por diferenças ou deltas. Assim, um símbolo é a diferença do símbolo original com o símbolo original anterior.

ALGORITMOS DE COMPRESSÃO

A. Run-Length Encoding (RLE)

O RLE é um algoritmo muito simples que procura reduzir o tamanho de repetições de símbolos, criando pares <valor, tamanho> em repetições compridas, sendo *valor* o símbolo repetido e tamanho o número de ocorrências consecutivas do símbolo. Como o par tem um tamanho mínimo, nem todas as repetições são substituídas por um par.

B. Codificação de Huffman

Criado por David A. Huffman em 1952, é um algoritmo que permite a criação de códigos de prefixo de tamanho variável por símbolo. Aos símbolos de maior probabilidade de ocorrência faz corresponder os menores códigos, e vice-versa. Os códigos podem ser visualizados como uma árvore binária

C. Algoritmo Lempel-Ziv – 1977 (LZ-77)

Publicado em 1977 por Abraham Lempel e Jacob Ziv, este algoritmo usa uma janela deslizante (*search buffer*), de tamanho fixo. O conhecimento dos dados vistos na stream previamente é então usado para substituir padrões encontrados durante a compressão, criando o trio <offset, comprimento, símbolo>, sendo offset a distância a recuar, comprimento o número de símbolos a copiar, e símbolo o próximo símbolo a enviar.

D. Algoritmo Lempel-Ziv – 1978 (LZ-78)

Em 1978, Abraham Lempel e Jacob Ziv publicaram um novo algoritmo de compressão, este usando um dicionário construído durante a compressão. Cria pares <índice, símbolo>, sendo *índice* o índice do dicionário a enviar e *símbolo* o próximo símbolo. Procura-se no dicionário a maior entrada igual ao que se pretende codificar. Se existir, codifica-se o par e cria-se uma entrada no dicionário igual a maior encontrada concatenada com o próximo a enviar. Caso não se encontre uma entrada, envia-se um par <0, símbolo> e cria-se uma entrada igual ao símbolo.

E. Algoritmo Lempel-Ziv-Welch (LZW)

Publicado em 1984 por Terry Welch, é uma melhoria do LZ78. Em vez de iniciar a compressão com um dicionário vazio, inicializa-se o dicionário com todos os símbolos do alfabeto.

F. Algoritmo Lempel-Ziv-Markov (LZMA)

Entre 1996 e 1998, Igor Pavlov (criador do programa de arquivação 7-Zip) desenvolveu um algoritmo de compressão com um dicionário de compressão parecido com o LZ77 com uma taxa de compressão elevada e uma velocidade similar aos mais utilizados algoritmos de compressão.

A sequência a codificar é transformada em deltas (diferença entre um símbolo e o seu antecedente). Essa

transformada é então codificada com um algoritmo de dicionário parecido ao LZ77. Finalmente, codifica-se o resultado anterior com um *range encoder* (um codificador aritmético que usa uma base de dígitos qualquer – por exemplo, um byte).

G. Prediction by Partial Matching (PPM)

Desenvolvido em 1984 por John Cleary e Ian Witten, o algoritmo tenta usar o contexto passado no processo de compressão para prever os símbolos seguintes da sequência a comprimir.

O maior contexto tem um tamanho predefinido. Se o símbolo a codificar não for encontrado no maior contexto, um símbolo especial é enviado para iniciar a procura num contexto menor. Caso o símbolo não tenha ocorrido na sequência, assume-se um contexto 0 em que os símbolos do alfabeto são equiprováveis. Cada contexto mantém uma contagem das ocorrências do símbolo, e a codificação pode ser feita com codificação aritmética inteira, por exemplo.

Este algoritmo permite rácios de compressão elevados, mas é mais lento do que outros algoritmos como os LZ. Variantes mais eficientes têm vindo a ser desenvolvidos, como o PPMd.

H. Deflate

O algoritmo Deflate, desenvolvido por Philip Katz, é uma evolução do LZ77, juntando-lhe codificação de Huffman.

Os dados são repartidos em blocos, os quais, após um codificador semelhante ao LZ77, podem ser alvo de compressão com Huffman fixo ou adaptativo, ou até não ser comprimido.

I. BZip2

Criado por Julian Seward em 1996, o Bzip2 é um algoritmo de compressão *open-source* de várias camadas, que oferece rácios de compressão mais elevados que os algoritmos LZW e Deflate, no entanto é significativamente mais lento.

O algoritmo divide os dados em blocos de tamanho entre 100KB e 900KB, e aplica-lhes as seguintes operações: i) RLE, ii) BWT, iii) MTF, iv) RLE, e v) Huffman.

J. CALIC (Context Adaptive Lossless Image Coding)

O CALIC, criado por Xiaolin Wu e Nasir Memon, tem como objetivo principal a obtenção de rácios de compressão extremamente elevados, mesmo que seja com o uso de técnicas complexas e/ou lentas para sua implementação.

O algoritmo opera em dois modos: i) binário e ii) continuous-tone, e suporta a mudança on-the-fly entre estes, dependendo do contexto do pixel a ser codificado no momento.

Caso a região do pixel conste de apenas dois valores de intensidade distintos, o codificador usa o modo binário. Neste modo, um codificador aritmético adaptativo ao contexto é usado para codificar três símbolos, incluído um símbolo de escape para sair do modo binário.

No segundo modo, o processo de codificação usa uma vizinhança de píxeis das duas linhas anteriores para fazer

uma previsão não linear do pixel a codificar. Posteriormente, os erros de previsão são modelados para redução de redundância e codificados por um qualquer algoritmo de codificação entrópica.

K. *FELICS (Fast, Efficient & Lossless Image Compression System)*

O FELICS utiliza os dois pixels vizinhos mais próximos de um pixel para o prever. O algoritmo é cerca de 5x mais rápido que o JPEG Lossless ('93), mas com uma compressão equiparável.

É feita uma aproximação da distribuição de probabilidade de cada pixel usando os seu vizinho diretos - superior e esquerdo. Se o pixel estiver dentro do intervalo de intensidades dos seus vizinhos, é codificado com um código binário ajustado simples, caso contrário recorre-se aos códigos de Golomb Rice devido à natureza exponencial da distribuição.

L. *LOCO-I (Low Complexity Image Compression)*

O algoritmo LOCO-I foi criado por Marcelo Weinberger, Gadiel Seroussi e Guillermo Sapiro com o objetivo de atingir rácios de compressão semelhantes a algoritmos mais complexos baseados em códigos aritméticos, mantendo a complexidade algorítmica do *FELICS*.

O sistema preditivo tem 3 preditores que fazem uso de 3 pixéis vizinhos – superior, esquerdo, e superior esquerdo. Os erros resultantes são depois codificados com códigos de Golomb-Rice.

CODECS EXISTENTES

A. *Lossless JPEG (1993)*

O antigo modo lossless do JPEG é uma adição ao *standard JPEG* que permite compressão não-destrutiva de imagens.

Este codec usa um esquema de previsão baseado nos três pixéis mais próximos (superior, esquerdo, e superior esquerdo). Os erros da previsão são depois comprimidos usando um codificador entrópico como Huffman ou códigos aritméticos. Dispõe de 7 esquemas preditivos, e de um modo sem esquema preditivo. O esquema utilizado é decidido pelo utilizador e serve de modelo para a imagem inteira.

É mais ou menos obsoleto, tendo sido substituído pelo JPEG-LS, que oferece uma maior eficiência na compressão que o seu antecessor.

B. *JPEG-LS*

Introduzido em 1994, o novo *standard lossless* consegue uma compressão mais rápida e obtém rácios de compressão maiores do que o antigo modo lossless JPEG.

Ainda em 2003 foram introduzidas extensões ao algoritmo, como por exemplo codificação aritmética.

O codec usa o algoritmo LOCO-I em dois modos: i) normal, e ii) *run mode*. O *run mode* permite o uso de símbolos compostos, útil para a codificação de regiões na imagem sem variação, dado que estas podem ser codificadas com menos de 1 bit por símbolo.

C. *GIF (Graphics Interchange Format)*

Criado pela CompuServe, uma equipa americana liderada pelo cientista Steve Wilhite em 1987, pretendia ser um método de compressão/descompressão mais rápido do que os utilizados na altura, para que imagens com tamanhos maiores pudessem ser descarregados mais rapidamente, mesmo com modems mais lentos.

Cada imagem dispõe de uma paleta de 256 cores, sendo que cada pixel é representado como um índice dessa mesma paleta. A imagem era depois comprimida com recurso a LZW.

A Unisys patenteou o algoritmo LZW em 1985 e a controvérsia resultante do acordo entre a CompuServe e a Unisys, que obrigava ao pagamento de licenças para uso comercial, culminou na criação do *standard PNG*.

D. *PNG (Portable Network Graphics)*

O formato PNG, criado em 1996 por uma colaboração num fórum online entre vários entusiastas e *experts*, como forma de substituir o GIF pois este i) usava um algoritmo de compressão patenteado e ii) dada a popularização de monitores mais avançados, o limite a 256 cores do formato tornava-se obsoleto.

Pré-compressão, o codec usa um sistema preditor com 5 métodos distintos (sendo um deles sem previsão) que permite o uso de um método de previsão diferente em cada linha da imagem. A diferença entre a previsão e o valor real em módulo de 256 é então codificada com o algoritmo Deflate.

E. *TIFF (Tag Image File Format)*

O formato TIFF, criado pela Aldus Corporation (agora Adobe Systems), foi criado como uma tentativa para os fabricantes de scanners usarem um formato *standard* de ficheiro para as digitalizações. No início, era apenas um formato para imagens binárias, visto que era o que os scanners eram capazes de capturar. À medida que as máquinas foram evoluindo, o formato também evoluiu e agora, juntamente com JPEG e o PNG, é considerado um formato popular para imagens de cor.

A implementação baseline de TIFF dispõe 3 modos: i) sem compressão, ii) uma versão modificada de Huffman, ou iii) PackBits – uma modificação simples de RLE. Existem ainda extensões que permitem a compressão com LZW

COMPARAÇÃO ENTRE OS DIFERENTES ALGORITMOS DE COMPRESSÃO

Todos os testes aqui expostos foram realizados numa máquina com um processador Intel® i5-9300H a uma frequência de 2.40GHz, com 8Gb de RAM e no sistema operativo Microsoft Windows 10 Home.

Está em uso para testes o ficheiro “mr” do dataset *Silesia Corpus* com a seguinte descrição:

Nome	Tipo	Tamanho (bytes)	Descrição
mr	Imagem	9,970,564	Imagem de uma ressonância magnética.

Para obter estes resultados usámos o software popular de compressão 7-Zip, juntamente com a biblioteca gzip. Foram obtidos os seguintes resultados:

Método de Comp.	Tamanho Final (bytes)	Rácio de Comp.	Vel. de Comp. (MB/s)	Vel. de Descomp. (MB/s)
<i>Deflate</i>	3,485,895	2.860	7.960	98.028
<i>LZMA</i>	2,749,292	3.627	3.400	48.762
<i>Bzip2</i>	2,440,928	4.085	12.628	22.857
<i>PPMd</i>	2,354,047	4.235	4.483	4.222

Destes resultados, podemos observar que:

1. O algoritmo Deflate possui uma velocidade de decompressão bastante elevada, mas oferece um rácio de compressão baixo.

2. LZMA oferece um bom rácio de compressão e uma boa velocidade de decompressão, sendo assim aproveitável para, por exemplo, playback de vídeo online.

3. Bzip2 oferece um bom rácio de compressão juntamente com a mais elevada velocidade de compressão, sendo útil para, por exemplo, anexamento de emails.

4. PPMd, um algoritmo baseado em PPM, oferece o melhor rácio de compressão, mas velocidades de compressão/decompressão muito baixas. São apenas úteis quando a largura de banda é extremamente limitada e é necessária uma compressão muito elevada.

BIBLIOGRAFIA

- [1] A. Gupta, A. Bansal and V. Khanduja, "Modern lossless compression techniques: Review, comparison and analysis," 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT), Coimbatore, 2017, pp. 1-8, doi: 10.1109/ICECCT.2017.8117850.
- [2] S. Ijmulwar, D. Kapgate (2014): "A Review on – Lossless Image Compression Techniques and Algorithms", *International Journal of Computing and Technology*, vol. 1 issue 9, pp. 1-4.
- [3] "Transformada de Burrows-Wheeler" May 23, 2016. [Online]. Disponível em: <http://multimedia.ufp.pt/codecs/compressao-sem-perdas/supressao-de-sequencias-repetitivas/metodo-de-burrows-wheeler/>. [Lido: Nov. 23, 2020].
- [4] Sayood, Khalid (2017): "The Burrows-Wheeler Transform", in *Introduction to Data Compression*. Cambridge: Morgan Kaufman, 175-178
- [5] Sayood, Khalid (2017): "Move-to-Front Coding", in *Introduction to Data Compression*. Cambridge: Morgan Kaufman, 178-179
- [6] "Algoritmo de Huffman" Oct. 13, 2020. [Online]. Disponível em: <http://multimedia.ufp.pt/codecs/compressao-sem-perdas/codificacao-estatistica/algoritmo-de-huffman/> [Lido: Nov. 23, 2020]
- [7] "Codificação de Huffman" Apr. 02, 2016. [Online]. Disponível em: https://pt.wikipedia.org/wiki/Codifica%C3%A7%C3%A3o_de_Huffman [Lido: Nov. 23, 2020]
- [8] "Huffman Coding" Apr. 10, 2001. [Online]. Disponível em: <https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node210.html/> [Lido: Nov. 23, 2020].
- [9] Sayood, Khalid (2017): "The LZ77 approach", in *Introduction to Data Compression*. Cambridge: Morgan Kaufman, 135-139
- [10] Sayood, Khalid (2017): "The LZ78 Approach", in *Introduction to Data Compression*. Cambridge: Morgan Kaufman, 139-140
- [11] Sayood, Khalid (2017): "Variations on the LZ78 Theme—The LZW Algorithm", in *Introduction to Data Compression*. Cambridge: Morgan Kaufman, 140-146
- [12] "How LZMA decompression algorithm works?" Oct. 12, 2020. [Online]. Disponível em: <https://stackoverflow.com/questions/64314593/how-lzma-decompression-algorithm-works> [Lido: Nov. 23, 2020]
- [13] "Lempel–Ziv–Markov chain algorithm" Oct. 13, 2020. [Online]. Disponível em: https://en.wikipedia.org/wiki/Lempel-Ziv-Markov_chain_algorithm [Lido: Nov. 23, 2020]
- [14] Sayood, Khalid (2017): "Prediction with Partial Match (ppm)", in *Introduction to Data Compression*. Cambridge: Morgan Kaufman, 167-175
- [15] "Prediction by partial matching" Sep. 12, 2020. [Online]. Disponível em: https://en.wikipedia.org/wiki/Prediction_by_partial_matching/ [Lido: Nov. 23, 2020]
- [16] "Deflate" Oct. 13, 2020. [Online]. Disponível em: <http://multimedia.ufp.pt/codecs/compressao-sem-perdas/codificacao-estatistica/codificador-qm/deflate/> [Lido: Nov. 23, 2020]
- [17] Tsai, Joe: "BZip2: Format Specification" March 17, 2016 [Online]. Disponível em: <https://github.com/dsnet/compress/blob/master/doc/bzip2-format.pdf> [Lido: Nov 23, 2020]
- [18] "Bzip2" Jul. 31, 2002. [Online]. Disponível em: <https://en.wikipedia.org/wiki/Bzip2/> [Lido: Nov. 23, 2020]
- [19] Sayood, Khalid (2017): "CALIC", in *Introduction to Data Compression*. Cambridge: Morgan Kaufman, 190-194
- [20] X. Wu and N. Memon, "Context-based, adaptive, lossless image coding," in *IEEE Transactions on Communications*, vol. 45, no. 4, pp. 437-444, April 1997, doi: 10.1109/26.585919.
- [21] P. Howard and J. Vitter, "Fast and efficient lossless image compression," in 1993 Data Compression Conference, Snowbird, UT, USA, 1993 pp. 351,352,353,354,355,356,357,358,359,360. url: <https://doi.ieeecomputersociety.org/10.1109/DCC.1993.253114>
- [22] "FELICS" Nov. 14, 2020. [Online]. Disponível em: <https://pt.gaz.wiki/wiki/FELICS> [Lido: Nov. 23, 2020]
- [23] M. J. Weinberger, G. Seroussi and G. Sapiro, "The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS," in *IEEE Transactions on Image Processing*, vol. 9, no. 8, pp. 1309-1324, Aug. 2000, doi: 10.1109/83.855427.
- [24] "LOCO-I algorithm" Jan. 17, 2006. [Online]. Disponível em: https://en.wikipedia.org/wiki/Lossless_JPEG#LOCO-I_algorithm/ [Lido: Nov. 23, 2020]
- [25] "Lossless JPEG" Jan. 17, 2006. [Online]. Disponível em: https://en.wikipedia.org/wiki/Lossless_JPEG/ [Lido: Nov. 23, 2020]
- [26] Sayood, Khalid (2017): "JPEG-LS", in *Introduction to Data Compression*. Cambridge: Morgan Kaufman, 194-196
- [27] Sayood, Khalid (2017): "Lossless Image Compression Formats", in *Introduction to Data Compression*. Cambridge: Morgan Kaufman, 202-204
- [28] W3C®, "Portable Network Graphics (PNG) Specification (Second Edition)" Nov. 10, 2003. [Online]. Disponível em: <https://www.w3.org/TR/PNG/#9Filters/> [Lido: Nov. 25, 2020]
- [29] Colt McAnlis, "How PNG Works" Apr. 6, 2006. [Online]. Disponível em: <https://medium.com/@duhroach/how-png-works-f1174e3cc7b7/> [Lido: Nov. 25, 2020]
- [30] Adobe, "TIFF Revision 6.0" June 03, 1992 [Online]. Disponível em: https://www.adobe.io/content/dam/udp/en/open/standards/tiff/TIFF_6.pdf [Lido: Nov 25, 2020]