CODEC não destrutivo para imagens

Estado de Arte

José Miguel Silva
Departamento de Eng. Informática
Universidade de Coimbra
Coimbra, Portugal
uc2017241700@student.uc.pt

José Miguel Soeiro
Departamento de Eng. Informática
Universidade de Coimbra
Coimbra, Portugal
uc2019224050@student.uc.pt

José Pedro Aguiar
Departamento de Eng. Informática
Universidade de Coimbra
Coimbra, Portugal
uc2019224624@student.uc.pt

***Abstract*—Desde o surgimento do mundo digital que a necessidade de obter ficheiros com o menor tamanho possível tem sido um dos maiores objetivos por parte de desenvolvedores. A ”explosão” das redes sociais levantam um grande problema: Como armazenar uma quantidade tão grande de imagens a um custo relativamente baixo preservando a sua qualidade? Neste documento é dado uma pequena visão do que é uma codificação e sobre os vários tipos de codificação não destrutiva e quais são as mais relevantes. O crescimento da importância da mesma foi provacado pelo rápido crescimento e presença no nosso quotidiano do campo da multimídia e das imagens digitais . A transmissão das imagens sem compactá-las ocupa mais espaço em disco e também muito tempo para transmissão pela rede. Portanto, a ideia básica é remover a redundância de dados apresentados dentro da imagem para que possamos reduzir ,por sua vez, o tamanho da imagem (salvaguardando os dados essenciais na imagem.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introdução

Com o objetivo de obter o menor ficheiro possível mantendo a qualidade, matemáticos e informáticos têm desenvolvido ao longo dos anos algoritmos de compressão. Este estudo revela o estado atual da compressão sem perdas, nomeadamente em imagens, abordando os algoritmos que dominam o presente bem como a sua estrutura.

A compressão de imagem consiste na redução de volume de dados do ficheiro inicial, para a qual é necessário comprimir a informação retirando a redundância (retirando informação que é “irrelevante”ou que apenas esta repetida). Existem dois tipos de compressão de imagem: (1)não destrutiva (lossless), em que é possível reconstruir exatamente a imagem original sem qualquer perda de informação, restaurando o seu estado inicial. Estes métodos embora obtendo uma boa taxa de compressão (2,7:1 no caso de PNG), em muitos casos não apresentam resultados suficientemente bons para serem utilizados em plataformas como redes sociais, onde tomando o exemplo do instagram são publicadas 55 milhões de imagens por dia [1]. É em situações dessas que entram algoritmos de compressão (2)destrutiva (lossy) que apresentando taxas de compressão bastante mais elevadas (10:1 no caso de JPEG) são muito mais convinientes na compressão de grandes quantidades de informação geradas pelo utilizador comum. No processo de compressão usando algoritmos destrutivos são perdidas características das imagens(normalmente são características algo irrelevantes), o que permite obter graus de compressão mais elevados. Este tipo de compressão basea-se na remoção de detalhes muito subtis como a diferença de cor entre duas folhas de uma árvore, movimentos muito rápidos que não conseguimos acompanhar num filme. Todos estes detalhes podem ser omitidos sem quese perceba que eles não estão lá.

Muitos destes métodos de compressão usam transformadas, que são usadas para a obtenção de uma melhor quantização (Fig. 1). A transformada em si é não destrutiva mas resulta numa quantização que produz uma cópia de menos qualidade que a original. O passo final deste processo consiste na codificação entrópica (compressão sem perdas), que apresenta a informação comprimida.

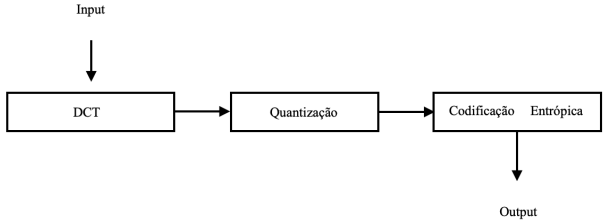

Fig. 1. Pipeline de compressão destrutiva.

A compressão de imagem baseia-se na remoção de informação redundante existente nas imagens.Existem três tipos de redundância nas imagens que são explorados pelos mecanismos de compressão:A codificação que é forma como a imagem é representada (codificada) introduz redundância.A inter-pixel que é redundância que a imagem apresenta através de repetições de padrões de pixeis. Psico-visual – A imagem inclui informação que visualmente não é relevante(utilizada por compressões destrutivas de imagens como .jpeg,etc..)

Este relatório tem como foco CODECs não destrutivos para imagens tomando uma abordagem seletiva dos algoritmos de compressão, expondo uma opinião crítica sobre os mesmos e descrevendo o seu funcionamento.

II. Análise de Técnicas e Métodos

Nesta secção são abordados Algoritmos de compressão mais usados, sendo eles tradicionais ou modernos. Os algoritmos modernos, embora apresentando melhores resultados no que toca à compressão, são muitas vezes os Algoritmos Tradicionais ”modificados”, sendo estes por sua vez usado como base.

A. Algoritmos Tradicionais

Run Length Encoding (RLE)

É o método bastante simples de compressão (muito utilizado em imagens). Onde os símbolos contíguos iguais são agrupados num conjunto onde é guardado o símbolo e o seu numero de ocorrências, portanto, no formato de um símbolo composto (do tipo símbolo; nº ocorrências).

80	80	80	56	56	56	56	56	78

{80,3}	{56,5}	{78,1}

Fig. 2. Funcionamento Run Length Encoding. [5]

Este método é bastante eficiente para imagens, principalmente as que apresentam uma grande redundância inter-pixel, ou seja, com mesma cor ao longo de uma grande área de pixeis. Por sua vez para imagens bastante pormenorizadas, com pouca redundância inter-pixel, esta compactação é ineficiente. Como RLE apenas resolve a redundância inter-pixels na codificação mas não resolve uma certa redundância na codificação como por exemplo quando uma imagem apresenta uma paleta de cores gigante mas assimetricamente distribuída ao longo da imagem , ou mesmo quando, uma imagem em preto e branco utiliza o Sistema RGB(0-preto,255-branco) em vez de uma codificação binaria em 0 e 1. RLE não vai resolver esta redundância na codificação ,para isso, é uma prática comum aplicar com RLE um mecanismo que resolva este tipo de redundância um exemplo desses mecanismos é Huffman-encoding.

Huffman-Encoding

Esquema de codificação com códigos de comprimento variável (VLC), que obtém o menor número médio de bits por símbolo quando não existe redundância inter-pixels, que usa a probabilidade de ocorrência dos símbolos da fonte a ser codificado para determinar códigos de tamanho variado para representar cada símbolo. Estes códigos são obtidos através da construção de uma árvore binária (Fig. 3):

1) O algoritmo cria uma tabela de frequência com os símbolos (Fig. 3).
2) Os 2 símbolos menos frequentes são removidos da tabela, agrupados e as suas frequências juntam-se formando um único nó e adiciona-se à tabela.
3) Repete-se o passo 2 até só existir um único nó na tabela.

A codificação de Huffman é pouco adequada quando existem muitos símbolos, uma vez que é necessário somar as probabilidades alem disso como foi dito anteriormente este é apenas eficiente quando já não existe redundância inter-pixels. Por estes motivos surgiram novos mecanismos como LZW.

Source symbol s_k	Probability p_k	Assigned codeword
s_0	0.1	111
s_1	0.3	10
s_2	0.4	0
s_3	0.2	110

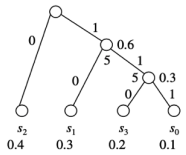

Fig. 3. Tabela de probabilidades e árvore binária. [4]

Codificação Aritmética

Codificação Aritmética é um método de codificação sem perdas que não sofre das desvantagens da codificação de Huffman e codificação de símbolos conjuntos e que tende a atingir uma taxa de compressão mais alta do que a codificação de Huffman. Os códigos aritméticos têm como ideia principal um intervalo [0,1[que ao ler símbolo a símbolo e usando a probabilidade de cada símbolo, ele diminui esse intervalo. Indicar um intervalo mais curto requer um maior número de bits, logo, sendo o nosso fim a compressão, declaramos que símbolos mais prováveis irão reduzir menos o intervalo, enquanto que símbolos menos prováveis reduzem bastante esse mesmo intervalo, o que provoca que símbolos mais prováveis sejam codificados com um número bastante inferior de bits.

Iteration # I	Encoded symbol s_k	Code subinterval $[L_c, H_c)$
1	s_1	[0.1, 0.4)
2	s_0	[0.1, 0.13)
3	s_2	[0.112, 0.124)
4	s_3	[0.1216, 0.124)
5	s_3	[0.12352, 0.124)

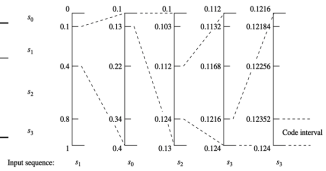

Fig. 4. Codificação Aritmética. [4]

Burrows-Wheeler Transform (Bwt)

O algoritmo manipula os símbolos da sequência de entrada através de várias permutações entre si com o intuito de criar uma nova sequência que irá permitir uma maior compressão. Pretende-se atingir um aumento da redundância espacial. A forma com que a técnica atinge este objetivo é, em primeiro lugar, construir uma matriz onde cada linha vai ser um rearranjo da sequência rotacionando sempre à esquerda uma posição. Depois ordena-se a matriz por ordem alfabética. A última coluna vai ser então a nova cadeia, pronta para comprimir. Este algoritmo opera em blocos, onde cada bloco é codificado separadamente como uma única string. Funciona bem com imagens, som e texto. Vários softwares de compressão utilizam-no como base. [6] [7]

B. Algoritmos Modernos

Deflate

Algoritmo com boa taxa de compressão que utiliza LZ77 e

Huffman Coding como base, usado nos arquivos ZIP.

1) A data é dividida em séries de blocos.
2) A codificação LZ77 é implementada para encontrar ocorrências repetidas em cada bloco e são postos ponteiros de referências no bloco.
3) Os símbolos encontrados são substituídos com códigos baseados nas suas ocorrências, usando a Codificação de Huffman.

Bzip2

Algoritmo que, embora de forma mais lenta, comprime a maioria dos formatos de arquivos mais eficientemente que o algortimo irmão ZIP.

1) Percorre o ficheiro caracter a caracter e aplica o Codec RLE nos dados iniciais.
2) Usa-se BWT é utilizado de modo a juntar os caracteres com contextos semelhantes.
3) Cada simbolo é substituído pelo seu index e passa para a frente
4) Comprime-se tudo com RLE
5) O output é sujeito à codificação de Huffman, onde são criadas múltiplas tabelas de Huffman num único bloco, visto que criar uma tabela é mais eficiente do que utilizar uma já existente.
6) É formado um bitmap para indicar os símbolos que são usados na tabela e arvore de Huffman.

PPM (Prediction By Partial Matching)

Este algoritmo percorre o ficheiro símbolo a simbolo, atribuindo uma probabilidade a cada um.

1) Um modelo de Markov de enésima ordem é assumido.(Markov- assumido que todos os símbolos são aleatórios ,ou seja, presume-se que os simbolos futuros dependem apenas do simbolo atual e não dos eventos que ocorreram antes dele (importanto áreas de modelagem preditiva e previsão probabilística) [3]
2) O arquivo de entrada (A) é percorrido desde o início, lendo um único símbolo por vez.
3) Para o símbolo Ai, que está na posição i, os n símbolos anteriores são usados para atualizar a probabilidade da sequência B (Ai-n, Ai-n-1 .... Ai-1, Ai).
4) Supondo que B seja um símbolo no alfabeto alvo, atualize todas as estruturas necessárias.
5) Se qualquer um dos símbolos em B não estiver presente, Ai é enviado de forma não codificado ou B é reduzido e adicionado ao novo alfabeto.

III. Conclusão do Estado de Arte

Em trabalhos futuros iremos testar alguns destes algoritmos de encoding e verificar quais codecs apresentam melhores resultados e o porquê. Irá ser feita uma análise para verificar se os resultados eram os esperados, bem como foi encontramos um método de compreensão não destrutivo mais eficiente que o png, o que é possivel verificar seja o PNG Otimizado seja o FLIF.

IV. Resultados

Atráves da testagem de métodos de codificação modernos nos nossos ficheiros, mais concretamente os codecs Bzip 2, PPM e Deflate (Codec usado em png), chegamos à conclusão que os mecanismos mais eficientes para os nossos ficheiros de imagem são os algoritmos Bzip2, PPM e AC quando utilizados individualmente, tal como é verificado na tabela I.

	Deflate	**Bzip2**	**PPM**
egg.bmp	2,838	3,925	3,704
landscape.bmp	2,597	3,306	3,210
pattern.bmp	21,145	27,907	23,762
zebra.bmp	2,322	3,123	2,952
Média	7,226	9,565	8,407

TABLE I

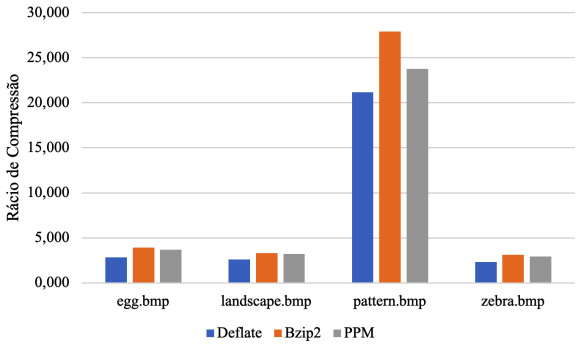

Fig. 5. Algoritmos individuais.

Como se pode observar, o codec BZIP2 foi o que apresentou melhores resultados, por isso é maioritariamente usado como base para as codificações usando vários codecs, sendo em algumas utilizados codecs modernos juntamente com clássicos. Sabendo que Bzip2 utiliza RLE+BWT+RLE+HUFFMAN, achamos por bem não utilizar os mesmos mecanismos, pois sentimos seria redundante, principalmente a utilização dos codecs RLE e Huffman, sendo que também foi descartado por não ser (teoricamente) considerado um dos melhores codecs para remoção de dados redundantes (Tabela II e III) e (Fig. 6). Os resultados, embora testados, apresentaram valores que não sendo satisfatórios não foram incluídos.

[!h]

	bzip2+AC	**PPM+bzip2**	**PNG**	**PPM+AC**
egg.bmp	3,925	3,688	4,023	3,704
landscape.bmp	3,316	3,191	3,473	NT
pattern.bmp	27,907	23,645	22,120	NT
zebra.bmp	3,123	2,937	3,213	NT
Média	9,568	8,365	8,207	3,704

TABLE II

Perante a análise dos dados, verifica-se que a melhor combinação foi Bzip2+Códigos Aritméticos (provavelmente devido a uma ligeira redundância na codificação). Com base neste resultados e na codificação do PNG, onde são usados

	bzip2+AC+LZ77	bzip2+LZ77	bzip2+PPM	bzip2+AC+PPM
egg.bmp	3,492	3,485	2,628	2,575
landscape.bmp	NT	NT	NT	NT
pattern.bmp	NT	NT	NT	NT
zebra.bmp	NT	NT	NT	NT
Média	3,492	3,485	2,628	2,575

TABLE III

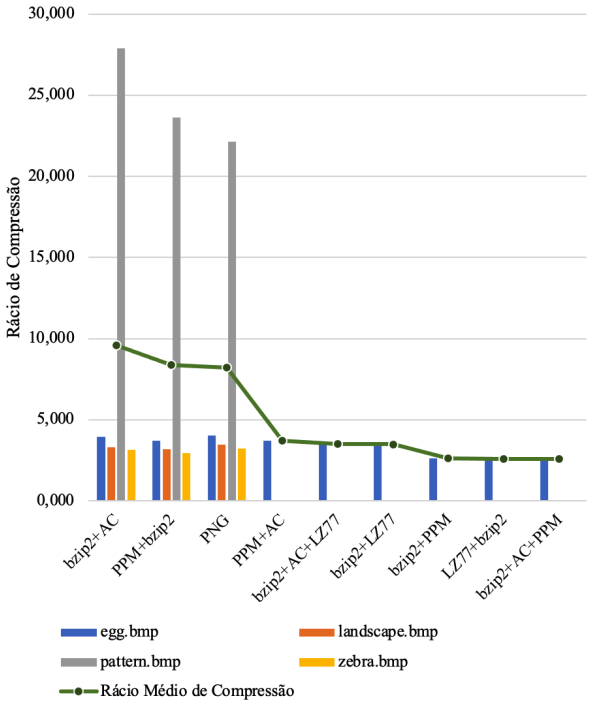

Fig. 6. Combinações de Algoritmos.

filtros nas imagens antes de usar o Deflate, decidimos que é possível melhorar os nossos resultados, usando um "filter" nos dados iniciais, com o propósito de melhorar a nossa taxa de compressão. Criamos 3 tipos de filtros (bastante simples).
Filtro 1- Calcula a moda, e subtrai esse valor em todos os pixels
Filtro 2- Subtrai todos os pixels pelo valor 127 (Valor intermédio da escala RGB, correspondente à tonalidade cinza).
Filtro 3- Subtrai o valor do pixel anterior, ao pixel atual.
Na tabela IV temos os rácios de compressão de cada filtro.

	Filtro 1	Filtro 2	Filtro 3	PNG
egg.bmp	3,925	3,933	4,059	4,023
landscape.bmp	3,306	3,306	3,823	3,473
pattern.bmp	27,907	27,907	25,131	22,120
zebra.bmp	3,123	3,141	3,123	3,213

TABLE IV

Com base nos valores, chegamos à conclusão de que a codificação com o filtro 2 é superior nos ficheiros de imagem pattern e zebra, apresentam maioritariamente dois tons, preto e branco, ou seja, valores à volta do 0 e 255, respetivamente. Dado que a subtração do valor médio com os tons pretos e brancos é quase a mesma (pois são os extremos da escala RGB), logo a imagem vai ficar como um "pano cinza", o que vai facilitar na codificação Huffman do codec BZIP2.
A codificação com o filtro 3 é superior nos ficheiros de imagem egg e landscape porque há pixeis muito semelhantes um ao lado do outro. Isso é visível, pois, perante a nossa visão conseguimos separar a imagem em zonas de tons brancos e tons pretos, o que vai levar a uma diferença entre zonas baixa, facilitando, mais uma vez a codificação de Huffman.
A codificação com o filtro 1 foi menos eficiente no ficheiro egg, pois é a imagem que tem mais tons diferentes, logo a subtração do pixel com a moda vai ser muito diferenciado ao longo da imagem.

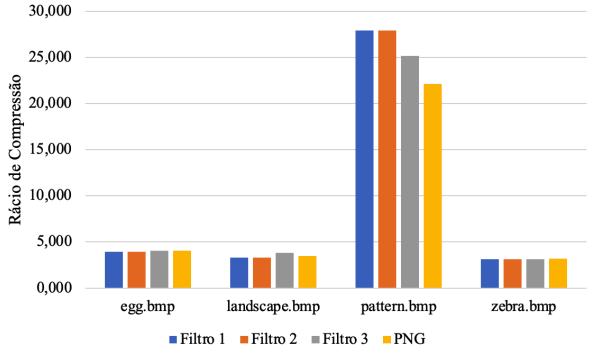

Fig. 7. Combinações de Filtros+Bzip2+AC

V. Conclusão do Trabalho

Concluimos que a codificação PNG é muito eficiente, sendo óbvia a razão pelo qual é considerada uma codificação standard.
Os nossos resultados foram satisfatórios, tendo obtido rácios de codificação superiores ao PNG em todos os ficheiros de imagem com a excessão do ficheiro Zebra.
Em futuros trabalhos, a intenção é encontrar um novo codec bastante eficiente para todos os tipos de imagem.

References

[1] Hu, Y., Manikonda, L., Kambhampati, S. (2014, June). What we instagram: A first analysis of instagram photo content and user types. In Icwsm.
[2] Ahmed, Nasir, T Natarajan, and Kamisetty R. Rao. "Discrete cosine transform." IEEE transactions on Computers 100.1 (1974): 90-93.
[3] https://en.wikipedia.org/wiki/Markov_model
[4] Karam, Lina. (2005). Lossless Coding. 10.1016/B978-012119792-6/50101-7.
[5] Ijmulwar, M. S., Kapgate, D. (2014). A Review on-Lossless Image Compression Techniques and Algorithms. IJCAT-International Journal of Computing and Technology, 1(9), 457-460.
[6] https://pt.wikipedia.org/wiki/M
[7] https://en.wikipedia.org/wiki/Burrows