CODECs não destrutivos

Técnicas de compressão aplicadas a imagens monocromáticas

Universidade de Coimbra, Licenciatura em Engenharia Informática
Segundo ano, primeiro semestre, Teoria da Informação

João Afonso Vieira de Sousa, 2019224599, uc2019224599@student.uc.pt

José Domingos da Silva, 2018296125, uc2018296125@student.uc.pt

Sancho Amaral Simões, 2019217590, uc2019217590@student.uc.pt

Tiago Filipe Santa Ventura, 2019243695, uc2019243695@student.uc.pt

***Abstrato*—Este relatório pretende abordar métodos de compressão destrutivos e não destrutivos, focando-se sobretudo nestes últimos. Relativamente à parte mais teórica do mesmo, proceder-se-á à enumeração de alguns filtros, transformadas e algoritmos mais utilizados no âmbito da compressão de dados. Estes serão então explanados, analisados e comparados a fim de se deduzirem as aplicações mais adequadas para cada um destes. Quanto à parte experimental deste relatório, diversas soluções para a compressão de imagens monocromáticas (em especial, do *dataset* proposto) serão apresentadas, devidamente justificadas e comparadas com a solução *baseline* fornecida, o *PNG*.**

Palavras-chave—destrutiva, não destrutiva, compressão, filtros, transformadas, monocromáticas, dataset, PNG

I. Introdução

Na atualidade, com a quantidade incomensurável de informação processada e enviada diariamente, tornou-se cada vez mais impreterível a engenharia de métodos para a flexibilização e utilização menos dispendiosa e menos exigente, em termos de infraestrutura, do fluxo dessa mesma informação. A compressão, mais que uma mais-valia, transformou-se numa necessidade. A menos que se trabalhe com supercomputadores no quotidiano, unidades de armazenamento de gigantescas dimensões e exageradas larguras de banda, nunca seria possível acompanhar o ritmo necessário ao funcionamento do próprio mundo, tão dependente da rápida partilha de conhecimentos e saberes entre indivíduos separados por grandes distâncias. Tomemos como exemplo a corrente pandemia do COVID-19: a elaboração de uma vacina provou ser um trabalho hercúleo, sendo extremamente dependente de estudos e ensaios clínicos sobre as mais variadas receitas para a vacinação. Empresas, farmácias e centros clínicos de todo o mundo, mesmo enquanto este relatório está a ser escrito, trabalham sem parar, confrontando as suas descobertas com as dos seus pares do outro lado do globo. Este é um claro exemplo da necessidade de transmissão de informação em tempo e espaço reduzido. Para tal objetivo ser concretizado é, portanto, necessário recorrer a técnicas de compressão de dados, que se podem dividir em dois tipos: *lossless* (não destrutiva) e *lossy* (destrutiva).

II. Compressão *lossy* vs compressão *lossless*

A. Compressão lossless

Em muitos casos, é necessário comprimir um determinado conjunto de dados de modo a que, quando descomprimido, a informação seja idêntica à inicial, isto é, durante o processo de compressão, a informação não é adulterada – compressão *lossless* (não destrutiva ou entrópica). Atente-se neste exemplo: a compressão do texto não deve alterar o seu conteúdo, uma vez que isto implicaria a modificação de carateres e, portanto, poderia fazer com que certos segmentos textuais deixassem de fazer sentido e/ou adquirissem sentidos diferentes. Entre as várias técnicas de compressão *lossless* encontram-se os algoritmos de *Lempel-Ziv* (*LZ77*, *LZ78*, *LZMA*, *LZW*, …), para propósito geral, *Dolby TrueHD*, para áudio, e *PNG*, para imagens.

B. Compressão lossy

Os sentidos do corpo humano, tais como a visão e a audição, apresentam limitações, isto é, existem limiares (frequências), a partir dos quais os sinais (eletromagnéticos e sonoros) se tornam indetetáveis. A compressão *lossy* (destrutiva) tira partido deste facto: trechos de um conjunto de dados considerados impercetíveis ou descartáveis para o ser humano são removidos em troca de um ganho substancial de espaço. Alguns exemplos de algoritmos de compressão *lossy* são: *JPEG*, para imagens e *Dirac*, para vídeo.

III. Técnicas de compressão *Lossless*

O processo utilizado por um típico *codec lossless* pode ser resumido através do seguinte diagrama:

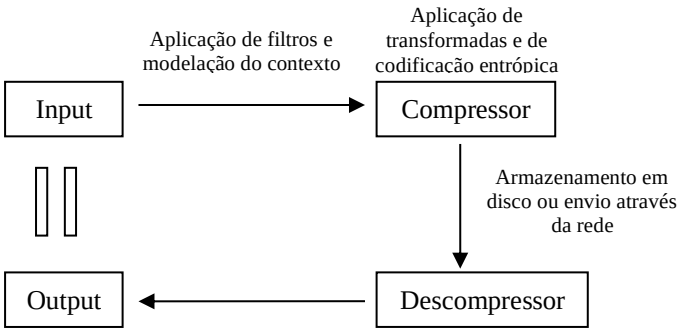


Fig.1: Diagrama simplificado da estrutura de um *codec*

Os filtros e transformadas são transmutações reversíveis aplicadas aos dados que possuem o propósito de tornar o *input* mais comprimível. É de salientar que, no entanto, as últimas apresentam uma vertente algorítmica mais complexa. Alguns exemplos destes mecanismos serão apresentados adiante.

A modelação do contexto é uma etapa na qual ocorre a definição de um modelo probabilístico que utiliza o contexto onde os símbolos ocorrem (através dos símbolos precedentes) para determinar o número de bits utilizados para os codificar.

O compressor e descompressor possuem o papel de, como o próprio nome indica, comprimir e descomprimir os dados, de modo a que o *Input* seja idêntico ao *Output*, uma vez que se trata de compressão não destrutiva.

A. Filtros

Os filtros, também designados por métodos de predição, quando aplicados a um conjunto de dados, reduzem a entropia deste, isto é, potenciam a sua redundância estatística.

De seguida, apresentam-se alguns destas transformações comummente utilizados no algoritmo de compressão *PNG - delta filters*. [1].

1) Sub

Cada *byte* é substituído pela sua diferença com o anterior/posterior.

2) Up

Cada *byte* é substituído pela sua diferença com o de cima.

3) Average

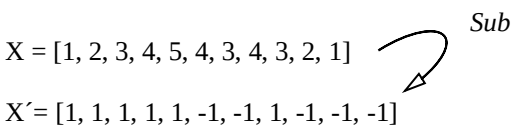
Cada *byte* é substituído pela sua diferença com a média entre o símbolo anterior e acima, truncando-se qualquer parte decimal resultante.

4) Paeth (Alan W. Paeth, 1956)

Cada byte é substituído pela sua diferença com o preditor de *Paeth* dos bytes anterior, acima e superior esquerdo.

P = *left* + *above* – *upperleft*
P_{Paeth} = p, tal que |P – p| é mínima, onde p é um dos vizinhos do *byte* corrente (*left*, *above* ou *upperleft*)

Segue abaixo um exemplo claro de como os filtros, neste caso o filtro *sub*, reduzem substancialmente a dispersão de uma fonte.



$H(X) \approx 2.231$, $H(X´) \approx 0.994$

B. Transformadas

As transformadas são aplicadas a um conjunto de dados de modo a que os algoritmos de compressão possam depois atuar de forma mais eficiente. Algumas permitem reduzir a dispersão dos dados e outras apenas agrupam símbolos de contexto semelhante e, como a entropia não tem em conta a ordem, permanece inalterada. De seguida, são apresentadas algumas das transformadas mais utilizadas no âmbito da compressão de dados.

1) Burrows-Wheeler Transform (BWT)

Esta transformada permite aumentar a redundância espacial de um dado conjunto de dados, agrupando símbolos de contexto idêntico. [2]

Aplicação da *BWT*:

a) Gerar a matriz de rotações da *string* a codificar.

b) Ordenar lexicograficamente as linhas da matriz obtida.

A *string* codificada é a última coluna. É necessário guardar o índice da linha da matriz onde a sequência original aparece juntamente com a *string* codificada para possibilitar uma sua posterior reconstrução.

Inversão da *BWT*:

Reconstrução da matriz de rotações da *string* original:

i) Começar com a *string* codificada na última coluna

ii) Ordenar lexicograficamente as linhas da matriz.

iii) Prefixar a(s) coluna(s) atuais com a coluna correspondente à *string* codificada.

iv) Voltar ao passo ii) até que se complete a matriz de rotações (ordenada).

A *string* original (descodificada) é a que consta na linha de índice igual ao guardado na fase de codificação.

Nota: Existem algoritmos mais eficientes para a aplicação da BWT, nomeadamente através de um *array* de sufixos da *string* a codificar.

2) Move to front (MTF)

A ideia principal por detrás deste algoritmo consiste em ter uma lista com os carateres do alfabeto da fonte a codificar. Habitualmente, esta transformada é procedida da aplicação de uma transformada *BWT* ou de um *RLE* (*Run Length Encoding*). [3]

Aplicação da *MTF*:

i) Ler o símbolo atual da *string* a codificar. O código que lhe é atribuído corresponde ao índice da posição onde aparece na lista de símbolos.

ii) Mover o símbolo em questão para a frente da lista.

iii) Voltar ao passo i) até que se complete o processamento de toda a *string*.

3) Transformada discreta de cosseno (DCT – Discrete Cosine Transform - Nasir Ahmed, 1972)

A *DCT* permite definir um conjunto discreto de dados (no caso das imagens, utilizando-se usualmente blocos de 8x8 pixeis) através da sobreposição de funções de cosseno, definida numa sequência finita de pontos. É de notar que cada função destas possui uma componente em x e em y.

Dada a finitude do conjunto em questão, é possível enumerar todas as possibilidades de funções de cosseno. Qualquer bloco depois pode ser gerado através de uma combinação linear dessas mesmas funções, cujos coeficientes são integrados na codificação.

É importante salientar que a DCT foi inicialmente desenvolvida para ser aplicada em algoritmos de compressão *lossy*, uma vez que permite remover detalhes impercetíveis pelo olho humano (correspondentes às frequências mais elevadas). No entanto, é também utilizada no âmbito da compressão não destrutiva. [4][8]

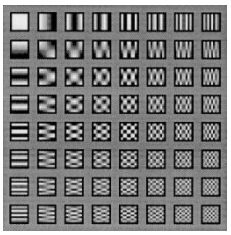


Fig.2: Tabela com todas as possibilidades das funções de cosseno para blocos de 8x8 píxeis
Fonte: Adaptado de [4]

C. Algoritmos de compressão

Com a evolução das necessidades, os *codecs* (não destrutivos) foram-se especializando para tratar determinados tipos de dados. No entanto, atualmente, a maioria apresenta visíveis similitudes, nomeadamente, no que diz respeito aos algoritmos base utilizados. Dentro destes algoritmos base, estão: codificação de *Huffman*, codificação aritmética, codificação por dicionários (família *Lempel-Ziv*). Ou seja, qualquer técnica de compressão *lossless* moderna, contém, algures no seu núcleo, pelo menos um dos algoritmos referidos (ou variantes destes) a correr. Alguns destes mesmos algoritmos serão a seguir brevemente descritos.

1) Algoritmos base de compressão lossless

a) Run Length Encoding (RLE)

Este algoritmo tira partido da redundância espacial de um dado conjunto de dados. Corridas de um dado símbolo *a*, de tamanho $n > 3$, são substituídas pelo *token* ($\vdash$, *a*, *n*), onde $\vdash$ é um carater de escape predefinido. [6]

b) Codificação de Huffman (Huffman Encoding - David A. Huffman, 1952

A codificação de *Huffman* parte do princípio que, aos símbolos com maior probabilidade de ocorrência, deve ser atribuído um código de menor comprimento. É construída uma árvore binária cujas folhas correspondem aos símbolos a codificar e os nós intermédios são símbolos compostos, cujo valor de probabilidade é a soma das probabilidades dos seus ascendentes. Esta técnica permite obter códigos de prefixo e, portanto, unicamente descodificáveis. [7]

c) LZ-77 (Abraham Lempel, Jacob Ziv, 1977)

O *LZ-77* é um código de dicionário que permite tirar partido da repetição de sequências de símbolos ao longo de um conjunto de dados e, em oposição ao que acontece nos códigos de *Huffman*, não necessita de conhecimento à *priori* da fonte. São utilizadas duas estruturas: o *search buffer* (*SB*), correspondente à janela com os últimos símbolos N_S processados e o *look-ahead buffer* (*LB*), que contém os próximos N_L símbolos por processar.

A ideia central deste algoritmo é a de procurar no *SB* o maior padrão que ocorre no *LB*. Caso esse padrão se encontre, é transmitido o *token* (*offset*, *l*, *next*), onde o *offset* é a distância relativa do padrão encontrado à posição atual do *SB*, *l* o seu comprimento e *next* o código do próximo símbolo.

Apesar de o *LZ-77* não utilizar qualquer conhecimento estatístico sobre a fonte, o seu desempenho aproxima-se muito do de codificadores entrópicos. Uma vez que o tamanho do *SB* é limitado, não é possível encontrar padrões repetidos de grandes dimensões. Portanto, como alternativas mais eficientes e também mais usadas na atualidade, existem o *LZ-78* e o *LZW*. [9]

2) Algoritmos modernos de compressão lossless

a) Bzip2 (Julian Seward, 1996)

O *Bzip2* é um algoritmo de compressão capaz de comprimir qualquer tipo de ficheiro individualmente, não podendo atuar sobre arquivos. Esta técnica de compressão *lossless* apresenta múltiplas camadas de compressão. [6][11]

Etapas de compressão do *Bzip2*:

i) *RLE* dos dados iniciais.
ii) Transformada *BWT* em blocos de até 900 KB.
iii) Transformada *MTF*.
iv) *RLE* do resultado da transformada *MTF*.
v) Codificação de *Huffman*.
vi) Seleção entre várias tabelas de *Huffman*.
vii) Codificação unária da seleção da tabela de *Huffman* (corrida de 1´s seguida de um 0).
viii) Aplicação do filtro *sub* nos tamanhos dos símbolos de cada tabela de *Huffman* selecionada.

b) Deflate (Phil Katz)

O algoritmo de compressão *deflate* foi inicialmente desenvolvido para ser utilizado em ferramentas de compactação de arquivos como *PKZIP* e *GZIP* (*.zip*). [7]

Etapas de compressão do *Deflate*:

i) Divisão dos dados em blocos.
ii) Aplicação do *LZ-77* em cada bloco.
iii) Codificação de *Huffman* com geração de duas árvores: uma para codificar os símbolos e comprimentos das cadeias encontradas pelo *LZ-77* e outra para codificar os *offsets* contidos em cada *token* formado pelo mesmo.

c) Lossless JPEG (1993)

O *Lossless JPEG* foi uma nova funcionalidade adicionada ao *standard JPEG* para permitir a compressão não destrutiva de imagens. [5]

Etapas de compressão do *Lossless JPEG*:

i) Descorrelação dos dados através da aplicação de filtros (métodos de predição).
ii) Aplicação de um codificador entrópico nos resíduos resultantes (códigos de *Huffman* ou códigos Aritméticos, por exemplo).

d) O algoritmo de compressão utilizado no PNG (Portable Network Graphics - 1996)

O *PNG* é o formato de dados para imagens mais utilizado na atualidade, não só pela sua elevada versatilidade, mas também porque dispõe de até 2^{24} cores (*RGB*) e 2^8 graus de transparência (canal *alpha*). Além do mais, a compressão de imagens é efetuada sem qualquer perda de informação, alcançando taxas de compressão bastante elevadas quando comparadas com as de algoritmos de propósito mais geral, como o *Bzip2* ou o *Gzip*.

O *PNG* será novamente mencionado mais à frente neste relatório, uma vez que constitui a solução *baseline* para a compressão do *dataset* de imagens monocromáticas fornecido. [1]

Etapas de compressão do *PNG*:

i) Aplicação de preditores nos dados.
ii) Aplicação do algoritmo *deflate* nos resíduos resultantes do passo anterior.

3) Aplicações dos algoritmos de compressão lossless

Após a enumeração de alguns algoritmos de compressão não destrutiva e sua breve descrição, há que destacar que a existência da diversidade destas técnicas permite satisfazer a variedade de tipos de dados que existem e as particularidades que cada um destes apresenta. No caso das imagens, será mais adequado utilizar algoritmos como o implementado no formato *PNG* ou o *Lossless JPEG* em detrimento de métodos de compressão de propósito mais geral como o *Bzip2*, enquanto que nos ficheiros de texto será mais eficiente utilizar códigos de *Huffman* ao invés de *RLE*.

Dentro do domínio das imagens, podemos ter também especificidades que necessitam de algoritmos mais *well suited*, na medida em que apresentam um melhor desempenho no que toca à compressão de imagens do subdomínio associado. Por exemplo, falando-se do *dataset* que foi proposto (conjunto de imagens em tons de cinzento), que é um subdomínio dentro do domínio das imagens, é possível, certamente, encontrar uma técnica de compressão não destrutiva mais eficiente do que a utilizada no *PNG*, ao serem estudadas meticulosamente as particularidades das imagens monocromáticas. É exatamente este trabalho de investigação que foi desenvolvido e será explanado adiante, neste relatório.

4) Critérios de escolha de um algoritmo de compressão

Habitualmente, a escolha de um algoritmo de compressão para uma determinada finalidade, tem em conta os seguintes critérios: [8]

- Eficiência de compressão (Taxa de compressão).
- Velocidade de compressão/descompressão.
- Complexidade de implementação.
- Robustez.
- Escalabilidade.

Relativamente à parte experimental deste relatório, será dada preferência a soluções que priorizem a eficiência de compressão do *dataset* fornecido.

IV. COMPRESSÃO DE IMAGENS MONOCROMÁTICAS

Este trabalho prático possui, como objetivo central, a pesquisa/implementação de técnicas de compressão não destrutivas para imagens monocromáticas *.bmp* e posterior comparação com a solução base proposta, o *PNG*, após aplicadas no *dataset* fornecido. Para tal efeito, foram selecionados os seguintes algoritmos de compressão: *JPEG2000* (*Jasper*) [12] – variante do *Lossless JPEG*, que foi mencionado anteriormente – e o *Bzip2* [11]. Além disso, foi desenvolvido um *CODEC* experimental, em *Python*, intitulado de *CMP*, que permite o teste de várias combinações de filtros, transformadas e algoritmos de compressão sobre os dados alvo.

Com o objetivo adicional de se testar a disparidade de eficiência de algoritmos de compressão *lossy* e algoritmos de compressão *lossless*, foi também utilizado o *JPEG* (*Lossy*) [10].

A. CMP

O *CMP* é um *CODEC* experimental para imagens *.bmp* monocromáticas desenhado com o intuito de, dado um *input*, ser possível aplicar diversas combinações de transformações ao mesmo e recolher os dados de *performance*. Estes dados são depois utilizados para determinar qual o algoritmo ideal (ou pelo menos, próximo do ideal) para o *input* fornecido.

Este *CODEC* foi desenvolvido na linguagem *Python*. A biblioteca *NumPy* foi extensivamente utilizada de modo a proporcionar um *speedup* maior nos processos de compressão e descompressão. No entanto, sendo o *Python* uma linguagem significativamente mais lenta do que, por exemplo, as linguagens *Java* e *C*, é de notar que os tempos de execução do *CMP* nem sempre foram os desejados.

O funcionamento do *CMP* foca-se nas classes *CMPCompressor* e *CMPDecompressor*, que comunicam com os vários módulos implementados (filtros, transformadas e algoritmos de compressão), abaixo listados.

1) Filtros

- *Sub* *
- *Up* *
- *Paeth* * simplificado (resíduo = atual – P, P = *left + above - upperleft*)

2) Transformadas

- *MTF* *

3) Algoritmos de compressão

- *RLE* *
- Codificação de *Huffman* *
- *LZW* *
- *LZMA (*versão melhorada do *LZ-77*, que utiliza um codificador aritmético – *range encoder)*

* - previamente introduzido neste relatório

Nota: A transformada *BWT* foi também implementada mas, dado a sua elevada complexidade temporal, não foi incluída nas classes acima referidas.

B. JPEG

O *JPEG*, além de um conhecido formato de imagens, é também um algoritmo de compressão destrutivo muito eficiente. A sua pilha de compressão é a seguinte [10]:

1. Transformação do espaço de cor *RGB* para um espaço de cor de luminância/crominância.
2. *Downsampling* da crominância.
3. Divisão da imagem em blocos de 8x8 píxeis.
4. Aplicação da *DCT* em cada bloco formado no passo anterior.
5. Quantização dos dados – componentes da *DCT* previamente aplicada, quando abaixo de um limiar predefinido, são removidas.
6. Aplicação de *RLE* e códigos de *Huffman* nos coeficientes da *DCT* que restam.

C. Dataset

O *dataset* fornecido consiste num conjunto de imagens monocromáticas *.bmp* de 8 *bits* de dimensões variáveis.

Ficheiro - Dimensões	**Tamanho (*MB*)**
egg.bmp – 3439x5158	16.9
landscape.bmp – 4452x2472	10.4
pattern.bmp – 7936x6049	45.7
zebra.bmp – 4724x3543	15.9

Fig.3: *Dataset*

D) Resultados obtidos e discussão

1. Aplicação de filtros e transformadas disponíveis no CMP

Antes da implementação do processo de compressão e descompressão das imagens, foi levado a cabo um estudo do efeito da aplicação dos filtros e transformadas disponíveis no *CMP* sobre a entropia de cada imagem do *dataset*.

	Entropia (bits)				
Ficheiro \Filtro	**Nenhum**	***Sub***	***Up***	***Paeth* (simplificado)**	***MTF***
egg.bmp	5,7242	2,6994	2,7506	2,2614	3,4178
landscape.bmp	7,4205	2,8242	2,7652	2,3807	3,4654
pattern.bmp	1,8292	0,5957	0,6191	0,5645	0,6177
zebra.bmp	5,8312	3,2219	3,1808	2,5748	4,3309

Fig.4: Entropia das fontes

É claramente vantajosa a utilização de filtros/transformadas nas imagens, uma vez que reduzem substancialmente a sua dispersão (entropia), tornando-as mais comprimíveis.

Neste *dataset*, o filtro de *Paeth* simplificado é dominante, uma vez que mais aumenta a redundância estatística dos dados.

Verifica-se também que a imagem *pattern.bmp* é a que apresenta um maior potencial de compressão, uma vez que apresenta uma entropia muito inferior a $M = log_2 \#A = 8$ (M é o majorante da entropia para qualquer fonte e A é o alfabeto da fonte).

2. Gerações CMP

Foram testadas várias combinações de transformações nos dados utilizando o *CODEC CMP* com o objetivo de encontrar qual a melhor sequência de algoritmos para cada imagem em específico. Convencionou-se que, cada uma dessas combinações se intitularia de geração *i*, onde *i* é o seu índice. Foram produzidas, no total, 15 gerações.

Geração *i*	Sequência de algoritmos
0	*MTF* - *RLE* - Codificação de *Huffman*
1	Filtro *Up* - *RLE* - Codificação de *Huffman*
2	*LZW* com tamanho máximo de dicionário de 4096 e sem *reset* do mesmo - Codificação de *Huffman*
3	*LZW* com tamanho máximo de dicionário de 1024 e sem *reset* do mesmo - Codificação de *Huffman*
4	Filtro de *Paeth* simplificado - *RLE* - Codificação de *Huffman*
5	*LZW* com tamanho máximo de dicionário de 8192 e sem *reset* do mesmo - Codificação de *Huffman*
6	*LZW* com tamanho máximo de dicionário de 16384 e sem *reset* do mesmo - Codificação de *Huffman*
7	Filtro *Up* - Filtro de *Paeth* simplificado - *RLE* - Codificação de *Huffman*
8	Filtro *Up* - *LZMA*
9	Filtro *Up* - Codificação de *Huffman* - *LZMA*
10	Filtro *Sub* - *LZMA*
11	Filtro *Sub* - *RLE* - *LZMA*
12	Filtro *Up* - *RLE* - *LZMA*
13	Filtro de *Paeth* simplificado - *LZMA*
14	Filtro de *Paeth* simplificado - *RLE* - *LZMA*

Fig.5: Mapeamento de gerações – sequências de transformações

Tamanho após compressão (*MB*) / Taxa de compressão (%)									
Ficheiro	***egg.bmp***		***landscape.bmp***		***pattern.bmp***		***zebra.bmp***		**Média**
Geração	T_f	***TC***	T_f	***TC***	T_f	***TC***	T_f	***TC***	***TC***
0	6,86	59,48	4,63	55,9	2,65	94,22	8,46	46,97	64,1
1	5,49	67,55	3,84	63,4	2,63	94,26	6,18	61,26	71,6
2	5,24	69,05	3,39	67,74	2,47	94,61	5,75	63,95	73,8
3	5,37	68,29	3,55	66,13	2,51	94,52	6,04	62,17	72,8
4	4,5	73,42	3,4	67,59	2,28	95,01	5,06	68,3	76,1
5	5,2	69,26	3,33	68,27	2,48	94,57	5,69	64,34	74,1
6	5,2	69,25	3,34	68,14	2,54	94,46	5,68	64,42	74,1
7	7,84	53,67	5,09	51,53	5,78	87,38	8,56	46,41	59,7
8	4,74	72,02	2,89	72,48	2,39	94,79	5,57	65,09	76,1
9	4,98	70,58	3,22	69,29	2,4	94,75	5,84	63,4	74,5
10	4,59	72,87	2,87	72,65	2,44	94,67	5,55	65,21	76,4
11	4,45	73,72	2,89	72,47	1,88	95,9	5,47	65,71	77
12	4,64	72,57	2,97	71,67	1,83	96	5,29	66,83	76,8
13	3,98	76,5	2,59	75,33	2,18	95,24	4,44	72,17	79,8
14	3,98	76,51	2,78	73,54	2,01	95,62	4,57	71,39	79,3
Média	5,14	69,65	3,39	67,74	2,56	94,4	5,88	63,17	

Fig.7: Tamanho após compressão (T_f) e taxa de compressão ($TC = (T_i - T_f) / T_f * 100$)

3. Resultados obtidos pelo CMP

Foi implementando um mecanismo de *logging* e *benchmarking* no *CMP, o que* permitiu uma maior facilidade na recolha e tratamento de dados relativos à compressão e descompressão das imagens do *dataset*.

Tempo de compressão/descompressão (s)										
Ficheiro	***egg.bmp***		***landscape.bmp***		***pattern.bmp***		***zebra.bmp***		**Média**	
Geração	Δt_c	Δt_d	Δt_c	Δt_d	Δt_c	Δt_d	Δt_c	Δt_d	Δt_c	Δt_d
0	45,14	69,28	29,65	44,36	78,39	109,8	53,53	74,95	51,7	74,6
1	22,09	31,21	16,01	22,82	20,29	21,11	21,77	33,34	20	27,1
2	26,40	31,43	10,54	23,31	246,83	21,70	16,70	31,75	75,1	27
3	24,34	30,95	9,77	18,20	150,08	18,40	18,44	29,56	50,7	24,3
4	19,77	69,6	15,07	56,28	22,21	160,34	21,04	75,35	19,5	90,4
5	31,51	26,84	11,47	17,35	263,63	19,24	17,94	27,63	81,1	22,8
6	32,96	25,95	10,30	15,57	281,63	20,65	18,34	26,50	85,8	22,2
7	27,41	83,63	18,41	55,34	29,66	122,80	27,96	82,57	25,9	86,1
8	19,70	0,75	15,52	0,45	16,96	0,96	22,51	1,03	18,7	0,8
9	11,18	127,9	7,75	81,01	23,26	167,19	11,50	133,28	13,4	127,3
10	19,65	0,79	14,86	0,41	15,62	0,86	21,22	0,95	17,8	0,8
11	44,06	22,68	35,63	19,37	25,44	15,93	46,07	25,48	37,8	20,9
12	43,97	24,24	36,49	20,37	21,45	16,77	54,68	28,39	39,1	22,4
13	23,57	44,75	18,36	44,60	18,00	134,14	25,06	54,78	21,2	69,6
14	41,81	75,07	34,17	57,29	24,7	153,85	43,27	79,23	36,0	91,4
Média	28,90	44,34	18,93	31,78	82,54	65,58	28,00	46,99		

Fig.6: Tempos de compressão (Δt_c) e de descompressão (Δt_d) do *CMP* em cada geração

4. Discussão dos resultados obtidos pelo CMP

A partir da análise dos resultados obtidos, é possível afirmar o seguinte:

• Como previsto pela tabela da *Fig.4*, as gerações que utilizam o filtro *Paeth* simplificado e possuem a restante *stack* de compressão igual, apresentam uma maior taxa de compressão. No entanto, uma vez que a inversa do filtro mencionado foi implementada através de *Python* puro, em detrimento do *NumPy*, as mesmas gerações apresentam elevados tempos de descompressão.

• A menor capacidade de gerar redundância estatística da *MTF* nas imagens *.bmp*, quando comparada com os filtros *Up*, *Sub* e *Paeth* (ver *Fig.4*), é refletida na taxa de compressão média da geração 0, sendo a segunda menor.

• As gerações que utilizam *RLE* e codificação de *Huffman* são as que apresentam menores tempos de compressão / descompressão médios. Isto deve-se ao facto de o *RLE* reduzir o número de símbolos da fonte, o que permite uma codificação de *Huffman* mais rápida.

Nota: no pior dos casos, o número de símbolos da fonte é igual ao número de símbolos da fonte após a passagem do *RLE*, caso não existir nenhuma corrida e um símbolo de comprimento superior a 3, o que é altamente improvável acontecer.

• O tempo de compressão é significativamente elevado nas gerações que utilizam *LZW*. Isto poderá acontecer devido a colisões de *hash* durante a ação de *lookup* no dicionário construído pelo *LZW*.

• A taxa de compressão, nas gerações que utilizam *LZW*, é ideal quando o tamanho máximo do dicionário é de 4096.

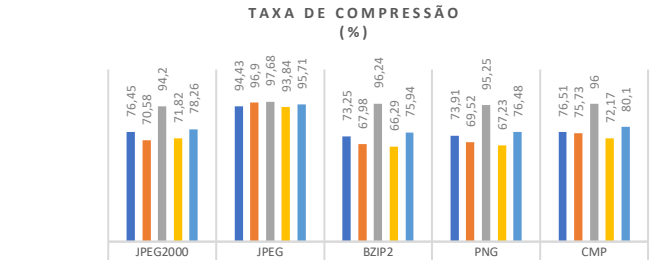
•Deduz-se também que a *performance* do *LZW* começa a deteriorar-se quando o tamanho máximo do dicionário aumenta a partir de um certo limite. Isto acontece porque os índices transmitidos pelo *LZW* passam a ocupar mais espaço do que a *substring* que têm como objetivo comprimir.

• As gerações que utilizam *LZMA* são as que apresentam as maiores taxas de compressão. Tal acontece, pois, o *LZMA* serve-se de um codificador aritmético (*range encoder*) que é geralmente mais eficiente do que um codificador de *Huffman*.

• A compressibilidade da imagem *pattern.bmp*, prevista na *Fig.4*, coincide com os resultados obtidos, uma vez que se trata da imagem na qual maiores taxas de compressão foram conseguidas.

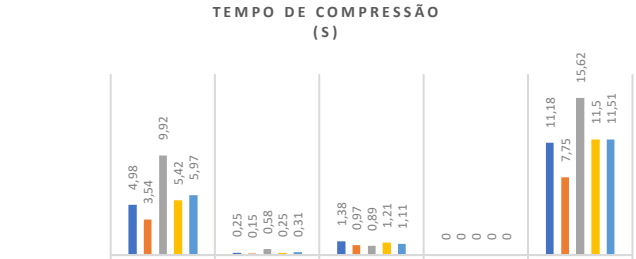
5. Comparação dos melhores resultados do CMP com os restantes CODECs

Foram selecionadas as gerações com os melhores resultados, tanto em termos de eficiência de compressão como de eficiência temporal, para posterior comparação com os resultados obtidos com os restantes *CODECs*.



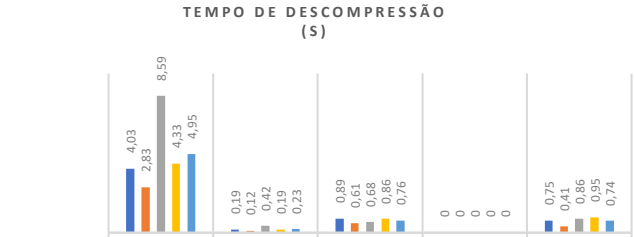
	JPEG2000	JPEG	BZIP2	PNG	CMP
egg.bmp	76,45	94,43	73,25	73,91	76,51 Ger. 14
landscape.bmp	70,58	96,9	67,98	69,52	75,73 Ger. 13
pattern.bmp	94,2	97,68	96,24	95,25	96 Ger.12
zebra.bmp	71,82	93,84	66,29	67,23	72,17 Ger.13
Média	78,26	95,71	75,94	76,48	80,1

Fig.8: Taxas de compressão, em percentagem, dos ficheiros do *dataset*, com cada um dos *CODECs* selecionados



	JPEG2000	JPEG	BZIP2	PNG	CMP
egg.bmp	4,98	0,25	1,38	0	11,18 Ger. 9
landscape.bmp	3,54	0,15	0,97	0	7,75 Ger. 9
pattern.bmp	9,92	0,58	0,89	0	15,62 Ger.10
zebra.bmp	5,42	0,25	1,21	0	11,5 Ger.9
Média	5,97	0,31	1,11	0	11,51

Fig.9: Tempos de compressão, em segundos, dos ficheiros do *dataset*, com cada um dos *CODECs* selecionados



	JPEG2000	JPEG	BZIP2	PNG	CMP
egg.bmp	4,03	0,19	0,89	0	0,75 Ger. 8
landscape.bmp	2,83	0,12	0,61	0	0,41 Ger. 10
pattern.bmp	8,59	0,42	0,68	0	0,86 Ger.10
zebra.bmp	4,33	0,19	0,86	0	0,95 Ger.10
Média	4,95	0,23	0,76	0	0,74

Fig.10: Tempos de descompressão, em segundos, dos ficheiros do *dataset*, com cada um dos *CODECs* selecionados

Notas:
- Os resultados correspondentes ao *JPEG* foram obtidos através da compressão destrutiva das imagens com um fator de qualidade de 50%.
- A compressão/descompressão dos ficheiros do *dataset* com o *PNG* são tomadas como instantâneas.

6. Discussão sobre a performance dos diversos CODECs

• Partindo da análise do gráfico da *Fig.*8, verifica-se que, com o *CODEC CMP*, foram obtidas taxas de compressão superiores ao restantes *CODECs*, exceto, obviamente, o *JPEG*, por ser do tipo *lossy*. Isto foi alcançado devido à uma análise exaustiva do *dataset* que foi efetuada, nomeadamente, através da produção de diferentes combinações de algoritmos de compressão. Esta análise levou à seleção da *compression stack* mais eficiente, de entre as várias testadas, para cada imagem *.bmp*.

• Pelo gráfico da *Fig.* 8 verifica-se que os melhores tempos de compressão relativos ao *CMP* são muito superiores aos dos restantes *CODECs*. Esta disparidade deve-se maioritariamente à menor eficiência da linguagem *Python* quando comparada com a das linguagens nas quais os referidos *CODECs* foram implementados (*JPEG-C++*, *JPEG2000 - C*). Hipoteticamente, alcançar-se-ia uma eficiência ótima se a implementação do *CMP* fosse totalmente feita recorrendo ao módulo *NumPy*, no entanto, tal coisa não foi possível.

• A taxa de compressão do *JPEG* destaca-se das restantes. Isto deve-se à perda de informação substancial que ocorre durante a etapa em que alguns dos coeficientes da *DCT* quantizados são truncados.

V. CONCLUSÃO E TRABALHO FUTURO SUGERIDO

Neste trabalho prático, foram adquiridas competências que permitem uma melhor compreensão de teoria de informação, nomeadamente, no campo da compressão de dados. Com o aumento exponencial do fluxo diário destes, torna-se cada vez mais fulcral haver algoritmos de compressão que os comprimam de forma mais eficiente. Foi dada também especial atenção ao desenvolvimento dos filtros uma vez que, é a partir destes que a compressibilidade final da imagem é decidida.

Para concluir, é necessário reforçar que, apesar da relativa facilidade com que se ultrapassou as taxas de compressão dos algoritmos *Bzip2*, *JPEG2000* e *PNG*, verificou-se uma limitação notável pela linguagem *Python*, na medida em que, geralmente, o tempo de compressão e descompressão se encontram longe do ideal num contexto profissional e/ou numa infraestrutura crítica. Uma sugestão seria extrapolar a metodologia algorítmica utilizada no *CMP* para uma linguagem mais eficiente do ponto de vista de velocidade de execução, como por exemplo, a linguagem C.

Uma proposta de melhoramento da implementação do *CMP* seria a emancipação entre o compressor e o descompressor, de modo a não terem de participar os dois na mesma execução. Para tal, ter-se-ia que escrever, no ficheiro comprimido, um cabeçalho (*header*) que, além de conter as tabelas de codificação e dimensões da imagem, teria outras informações impreteríveis para a sua descompressão, tal como a pilha de compressão utilizada. Além disso, o *CMP* tornar-se-ia de mais fácil uso se se tratasse de um programa executável no qual o utilizador insere, pela linha de comandos, a ação a executar (compressão/descompressão) juntamente com as *flags*/parâmetros desejados.

REFERÊNCIAS

[1] G. Roelofs, "PNG: The Definitive Guide", Cap. 9, 1999.

[2] A. Martubs, F. Laranjeira, J. Cunga, L. Silva, "Transformada de Burrows-Wheeler", Universidade Fernando Pessoa, http://multimedia.ufp.pt/codecs/compressao-sem-perdas/supressao-de-sequencias-repetitivas/metodo-de-burrows-wheeler/, acedido em 25/11/2020.

[3] A. Kaur, "Move To Front Data Transform Algorithm", GeeksForGeeks, https://www.geeksforgeeks.org/move-front-data-transform-algorithm/, acedido em 25/11/2020

[4] D. Marshal, "The Discrete Cosine Transform (DCT)", 10/04/2001, https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node231.html, acedido em 26/11/2020.

[5] "Lossless JPEG", Wikipedia, https://en.wikipedia.org/wiki/Lossless_JPEG, acedido em 25/11/2020.

[6] S. Ikmulwar, D. Kapgate, "A Review on: Lossless Image Compression Techniques and Algorithms", outubro 2014.

[7] A. Gupta, A. Bansal, V. Khanduja, "Modern Lossless Compression Techniques: Review, Comparison and Analysis", Dept. of Information Technology of Delhi

[8] L. J. Karam, "Lossless Image Compression", Arizona State University, Cap. 16, pp 385-419.

[9] P. Carvalho, "Teoria da Informação", Faculdade de Ciências e Tecnlogia da Universidade de Coimbra, Cap. 2, 2020/2021

[10] kornleski, jpeg-compressor, https://github.com/kornelski/jpeg-compressor/releases, acedido em 21/12/2020

[11] philr, bzip2_windows, https://github.com/philr/bzip2-windows/releases, acedido em 21/12/2020

[12] Michael Adams Department of Electrical and Computer Engineering University of Victoria, 'The JasPer Project Home Page', https://github.com/philr/bzip2-windows/releases, acedido em 21/12/2020