CODEC não destrutivo para Imagens

Gonçalo Ferreira
DEI
FCTUC
Coimbra, Portugal
2019219213

João Melo
DEI
FCTUC
Coimbra, Portugal
2019216747

Miguel Faria
DEI
FCTUC
Coimbra, Portugal
2019216809

***Abstract*—Neste documento estão apresentados os principais CODECs lossless de imagem com o intuito de descobrir quais os melhores tipos de compressão não-destrutiva de imagens para cada tipo específico de imagem, assim como os seus algoritmos. Sem compressão, a transmissão de imagens era muito mais lenta, além de ocuparem muito mais espaço em memória. Assim, a solução reside em eliminar a redundância presente nos dados de uma imagem de forma a que o seu conteúdo não seja alterado.**

***Keywords*—compressão não-destrutiva de imagens, redundância, codificação, descodificação, algoritmo**

I. Introdução

A compressão de imagem desempenha um papel muito importante em inúmeras aplicações. O motivo da compressão da imagem é reduzir a quantidade de dados necessários para representar amostras de imagens digitais e, portanto, reduzir o custo para armazenamento e transmissão. Existem diferentes técnicas para compactar imagens, com e sem perdas de informação. Neste documento discutimos apenas acerca da compressão de imagem sem perdas, usada quando os dados devem ser descompactados exatamente como eram antes da compressão. Reduz também a possibilidade de ocorrer erros de transmissão, uma vez que são transferidos bits de menor quantidade. Um algoritmo de compressão precisa de encontrar um equilíbrio entre a taxa de compressão e o tempo de compressão/descompressão. Técnicas de compressão populares, como LZ77 e Codificação Huffman são exemplos de técnicas de compressão sem perdas.

II. Codificação não destrutiva de imagens

A. Definição

A codificação não-destrutiva de imagens tem como objetivo representar uma imagem com o mínimo de bits possível, sem perder nenhuma informação, ou seja, ao ser descomprimida, fica exatamente igual à versão original. Isto é possível devido à presença de uma redundância significativa na imagem.

B. Conceitos básicos

O codificador recebe uma imagem e cria uma *bitstream* comprimida. Neste processo, são utilizadas 3 operações principais:

• Transformação - conversão da informação da imagem, para que seja compressa mais eficientemente;

• Mapeamento de informação para símbolos - conversão da informação em símbolos que possam ser codificados;

• Compressão de símbolos não-destrutiva - criação de uma *bitstream* binária ao associar palavras-código binárias aos símbolos de entrada.

O descodificador tem um processo semelhante, sendo este quase inverso.

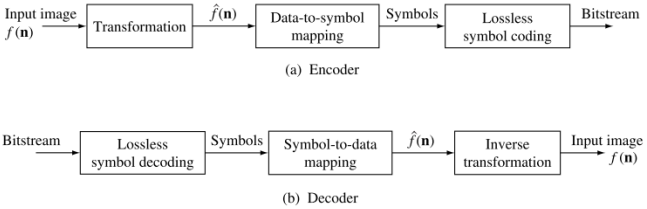

Figura 1 – Sistema geral de codificação não destrutiva

Para escolher o tipo de compressão não-destrutiva mais adequado, existem vários fatores que devem ser considerados, nomeadamente: eficiência de compressão, *delay* de codificação, complexidade da implementação, robustez e escalabilidade.

III. Algorítmos de compressão de dados

As várias estratégias de codificação não-destrutiva de símbolos podem ser agrupadas em 3 categorias principais:

• Estratégias estatísticas – é necessário saber a distribuição de probabilidade dos símbolos, de modo a que símbolos com maior ocorrência ocupem menos espaço. Algumas destas estratégias são a codificação Huffman e Aritmética.

• Estratégias baseadas em dicionários – são criadas dinamicamente tabelas de codificação e descodificação (dicionários) de símbolos com comprimento variável, à medida que ocorrem nos dados de entrada. Uma estratégia deste tipo é codificação Lempel-Ziv.

• Estratégias de codificação estruturada universal – não requerem conhecimento prévio da fonte e baseiam-se na produção de palavras-código de comprimento-variável presumindo-se que a sua distribuição decresce de forma uniforme. Estas estratégias são, por exemplo, códigos Elias e Exponencial-Golomb.

A. Codificação baseada em contexto

A codificação baseada em contexto pode ser dividida em 2 partes: um modelador baseado em contexto e um *coder*. Em cada ciclo, o modelador calcula a probabilidade do símbolo seguinte que irá ser codificado de acordo com o contexto observado. Dado que o contexto é formado por símbolos já codificados e este tipo de codificação é não-destrutivo, o contexto está também disponível no *decoder*, não sendo necessário a transmissão de informação. Neste tipo de codificação não é necessário saber nenhuma informação antes de ser executado.

B. Codificação Run-Length (RLE)

Este método é uma forma simples de compressão de dados não destrutiva, utilizada apenas em informação sequencial. Consiste na procura de símbolos repetidos e trocam-se os caracteres por um caracter que contém o símbolo e o contador binário no local da sequência original.

Este tipo de codificação é mais eficiente em ficheiros que tenham muita informação repetida, como por exemplo imagens com grandes áreas de preto ou branco, sendo de fácil e leve implementação. Porém, possui algumas desvantagens, uma vez que a informação original não pode ser imediatamente acessível e que observando a informação codificada é difícil saber o tamanho que o ficheiro descodificado irá ocupar.

C. Codificação Huffman

A codificação Huffman é baseada na frequência de caracteres: a um caracter com maior frequência é atribuída uma palavra-código menor, enquanto que um com menor frequência tem uma palavra-código maior.

O código Huffman de uma fonte consegue ser criado ao construir iterativamente uma árvore binária segundo os passos seguintes:

- o algoritmo percorre os dados e cria uma tabela com as probabilidades de cada símbolo ordenadas;
- os 2 símbolos menos frequentes são retirados desta tabela, formando um nó em que a sua probabilidade é a soma das probabilidades dos seus símbolos; o nó com a soma das probabilidades é adicionado à árvore, sendo que está ligado aos símbolos que o compõem formando um ramo;
- a árvore vai sendo formada ao repetir o processo anterior até restar apenas um único nó, sendo este a raiz da árvore;
- obtém-se a palavra-código de um símbolo ao percorrer a árvore do nó raiz até à folha que é a frequência do símbolo.

Apesar do processo de codificação ser realizado através de uma tabela de símbolos e palavras-código, o processo de descodificação é mais complexo. Existem várias formas de fazer a descodificação, sendo uma delas a reconstrução da árvore binária através da tabela de símbolos e palavras-código. Á medida que a *bitstream* é lida, de bit em bit, percorre-se a árvore desde a raiz até a folha, onde o símbolo correspondente a esta irá ser o output do descodificador. Este processo é repetido até se chegar ao fim da bitstream.

Este tipo de codificação é computacionalmente simples: tanto a codificação e descodificação são rápidos e usam poucos recursos. No entanto, usa bastante memória e técnicas como a codificação aritmética, que iremos abordar a seguir, são mais eficientes, pois não usam um número inteiro para cada símbolo.

D. Codificação Aritmética

Neste tipo de codificação, assim como na codificação Huffman, os caracteres mais usados vão ser armazenados com menos bits, no entanto, consegue uma maior compressão na maioria das vezes, apesar de necessitar de mais recursos.

O algoritmo de codificação pode ser realizado segundo os seguintes passos:

- constrói-se uma tabela de probabilidade de cada símbolo em intervalos adjacentes;
- começa-se do intervalo [0, 1), identificando-se o sub-intervalo ao qual corresponde o primeiro símbolo lido;
- para os símbolos seguintes, subdivide-se o intervalo atual em sub-intervalos proporcionais às probabilidades da tabela e encontra-se novamente o intervalo que corresponde ao próximo símbolo;
- repete-se este processo, obtendo no final um intervalo que corresponde à probabilidade de ocorrência de todos os símbolos na ordem correta;

O processo de descodificação é semelhante, possuindo mais um passo, onde é descoberto o símbolo, a partir dos sub-intervalos, e parando quando atinge um símbolo especial, adicionado previamente à fonte.

Uma das desvantagens deste processo é que existe um limite do número de símbolos que se consegue comprimir numa palavra código. Para além disso, também é necessário que a palavra código seja recebida na totalidade para que se possa começar a descodificação dos símbolos, e caso haja algum bit corrompido, a informação pode também ela ficar corrompida. Por outro lado, as vantagens são o elevado rácio de compressão que se obtém e o facto de ser um mecanismo eficiente para remover a redundância na fonte.

E. Codificação Lempel-Ziv

A codificação Lempel-Ziv é baseada em dicionários de palavras-código de comprimento-variável que vão sendo atualizadas com palavras-código binárias de tamanho pré-definido. Os algoritmos mais conhecidos deste tipo são LZ77, LZ78 e LZW (variante do método LZ, desenvolvido por Welch). Este processo é bom para comprimir dados com grandes blocos de bits do mesmo valor e tem uma boa performance em termos de tempo e espaço, embora o uso de processamento seja um pouco elevado.

O algoritmo LZ77, usado em programas de compressão como PNG, é baseado em 3 passos:

- a fonte é percorrida desde o início, caracter a caracter;
- são definidas 2 estruturas: um dicionário (que armazena as partes da fonte que já foram lidas) e um buffer (que funciona como uma janela "deslizante" que irá ler as partes da fonte que ainda não foram processadas, guardando assim a ocorrência dos símbolos, de modo a encontrar igualdades);
- se um caracter for repetido, é guardado o ponteiro de referência para a localização da última ocorrência do símbolo no buffer e o número de símbolos que são idênticos.

A descodificação deste método começa com o mesmo dicionário, calculando o símbolo correspondente a cada palavra-código a partir dele.

F. Codificação de Elias

Este processo de codificação tem com base códigos que atuam em números inteiros positivos, onde cada palavra-código é construída baseando-se no número inteiro correspondente. Na construção destes códigos assume-se que a distribuição das probabilidades decresce à medida que os números aumentam.

□ Códigos Gamma e Gamma' de Elias

O código Gamma de Elias, $\gamma'(I)$, para um número inteiro positivo (I) pode ser criado procurando a representação binária ($\beta(R)$) de I, determinando o número total de bits (L) de $\beta(R)$ e formando a palavra-código $\gamma'(I)$ constituída por (L-1) zeros seguidos da sua representação binária.

□ Códigos Delta de Elias

O código Delta de Elias, $\delta(I)$, para um número inteiro positivo (I) pode ser criado procurando a representação binária ($\beta(R)$) de I, determinando o número total de bits (L) de $\beta(R)$ e formando uma palavra-código de L segundo o procedimento usado nos códigos Gamma. A palavra-código

δ(I) final obtém-se acrescentando à anterior os últimos (L-1) bits da representação binária de I excluindo o bit mais significativo.

□ Códigos Omega de Elias

O código Omega de Elias, ω(I), para um número inteiro positivo (I) pode ser criado ao definir R=I, ω(I)=[0], C= ω(I)[(1)]. Determina-se a representação binária (β(R)) de R, define-se ω(I)=[β(R)][C]] e descobre-se o número total de bits (L) de β(R). Caso L seja maior que 2, define-se R=L-1 e repete-se a partir de [(1)]; caso L seja igual a 2, o processo termina; caso L seja igual a 1, define-se ω(I)=[0] e o processo termina.

Os códigos de Elias são de implementação fácil, pois não necessitam de um dicionário para a compressão, e recorrem a números inteiros de baixo valor. No entanto, o rácio de compressão não é dos melhores, comparativamente a métodos como a codificação aritmética, e é pouco eficiente relativamente ao uso de recursos.

G. *Codificação Exponencial-Golomb*

Tal como na codificação de Elias, também a codificação Exponencial-Golomb tem com base códigos que atuam em números inteiros positivos, no entanto, também aceitam o elemento neutro.

O código Exponencial-Golomb de ordem k de um número inteiro (I) gera uma palavra-código binária da forma:

$EG_k(I)$ = [(L′-1) zeros][(most significant (L-k) bits of β(I))+1][last k bits of β(I)]

= [(L'-1) zeros][$\beta(1+I/2^k)$][Last k bits of β(I)],

onde, β(I) é a representação binária de I, L é comprimento de β(I) e L' é o comprimento de $\beta(1+I/2^k)$, que corresponde a retirar os primeiros (L-k) bits de β(I) e somar aritmeticamente 1.

Os símbolos mais comuns são consultados numa pequena tabela de Huffman, enquanto que os restantes são codificados segundo a fórmula. Uma das vantagens é precisamente isso, ser de computação rápida e sem necessitar de pesquisa total, como é no caso de Huffman.

H. *Codificação LZMA (Lempel-Ziv Markov Chain Algorithm)*

A codificação LZMA utiliza um esquema de compressor com dicionário, semelhante ao de LZ77. Este algoritmo fornece um rácio de compressão alto quando se tem uma bitstream desconhecida, ou seja, funciona melhor que o LZ77 para fontes que tenham menos repetição, ou mais aleatórias. Este modelo foi pioneiro no aspeto da sua versatilidade, dado que utiliza contextos específicos para a representação dos *bitfields* da fonte, sendo que em termos de complexidade é parecido com o modelo genérico baseado em bytes, contudo o LZMA fornece uma compressão melhor, dado que separa bits que não sejam do mesmo contexto.

O algoritmo consiste em usar filtros delta ao longo da fonte toda, em que se subtraem os símbolos adjacentes, ou seja, o bit final é a subtração entre o bit atual e o anterior, o output é comprimido com o método janela deslizante como no caso de LZ77. Por último os símbolos são codificados usando um codificador de faixa, com base na probabilidade de distribuição.

O método LZMA possui um processo de compressão relativamente lento e com uso de mémoria elevado, quando comparado com outros métodos como Bzip2, apesar de ser bastante eficiente em ficheiros binários e do processo de descompressão ser mais rápido.

I. *Codificação Deflate*

Neste processo de codificação o input começa por ser dividido numa série de blocos e utiliza-se o método LZ77 para encontrar sequências repetidas ao longo de cada bloco, inserindo ponteiros de referência para a primeira ocorrência. Os símbolos são depois trocados por palavras-código, criadas segundo a codificação Huffman.

A maior desvantagem do deflate é que ele corresponde apenas informação numa janela de 32 Kbytes. Portanto, faz um ótimo trabalho a encontrar e codificar redundância nessa janela, mas não fora dela.

J. *Codificação Bzip2*

A compressão Bzip2 é um processo avançado de compressão por camadas.

O seu algoritmo consiste em 6 passos principais:

• a fonte é percorrida desde o início, caracter a caracter;

• efetua-se o método RLE na informação lida;

• aplica-se a técnica de Burrows-Wheeler (BWT), que ordena a informação, agrupando caracteres com contextos idênticos, de modo a facilitar os passos seguintes;

• faz-se a transformação "move-to-front", onde os símbolos são colocados num array, trocados pelo seu índice e movidos para a frente do *array*, permitindo que símbolos imediatamente recorrentes sejam trocados por 0;

• realiza-se o método RLE, que vai permitir reduzir o tamanho das sequências;

• a informação obtida sofre a codificação Huffman:

- para construir a tabela, o comprimento em bits de cada palavra-código é armazenado de forma codificada como a diferença entre este e o comprimento em bits da palavra-código passada;

- é criado uma *bitmap* para saber os símbolos que são usados na tabela e os que devem ser incluídos na árvore.

Uma vantagem deste processo de compressão é que se consegue criar ficheiros com tamanho bastante reduzido. Apesar de ser mais rápido e eficiente que o LZMA, é ainda demorado e dispendioso em uso de CPU, comparativamente a outros métodos.

K. *Codificação PPM (Prediction by Partial Matching)*

A compressão por PPM é uma técnica estatística adaptativa de compressão de dados baseada na previsão e na modelagem de contexto. Este modelo utiliza os n símbolos anteriores para prever o símbolo seguinte, sendo basicamente um modelo de Markov de n-ésima ordem. Normalmente com esta técnica é também incorporada uma das técnicas de compressão que utilizam a entropia, como por exemplo a compressão aritmética ou Huffman.

O algoritmo do PPM consiste em:

• é assumido um modelo de Markov de n-ésima ordem;

• o ficheiro de entrada, F, é percorrido na totalidade do início ao fim, símbolo a símbolo;

• para o símbolo A_i, símbolo na posição i de A, os n símbolos anteriores são utilizados para saber a probabilidade da sequência B(A_{i-n}, A_{i-n-1}, …., A_{i-1},A_i);

• sendo B um símbolo do alfabeto alvo, todas as estruturas necessárias são atualizadas;

• se nenhum símbolo em B é representado, ou se faz o envio de A_i não sendo comprimido ou reduz-se B e adiciona-se ao alfabeto.

Apesar ter uma boa taxa de compressão, a principal desvantagem do PPM é que é um processo lento e com grande uso de memória, pois tem de guardar a estatística da ocorrência de símbolos.

IV. Algorítmos de compressão de imagens

A. *FELICS (Fast Efficient Lossless Image Compression System)*

FELICS é conhecido pela sua rapidez e perda minima de eficiência na compressão de dados. Este codificador utiliza a descorrelação da imagem e codifica com o uso da entropia. Utiliza uma varredura raster e, usando os dois pixeis mais próximos de um certo pixel, consegue obter diretamente uma probabilidade de distribuição para a intensidade desse pixel. Utiliza também uma técnica moderna para detetar e identificar valores que sejam errados, tendo um conjunto de modelos de erro, sendo que cada um corresponde para um simples código de prefixo. Quando tiver este código, a intensidade é finalmente comprimida.

Este algoritmo de compressão consegue executar a sua tarefa 5 vezes mais rápido que o JPEG original não destrutivo, e consegue um rácio de compressão parecido.

B. *CALIC (Context Based Adaptive Lossless Image Coder)*

CALIC representa uma das técnicas de uso geral e prático de codificação de imagens sem perdas com melhor desempenho. Para fins de modelagem e previsão de contexto, o processo de codificação usa uma vizinhança de valores de pixel retirados apenas das duas linhas anteriores da imagem. Consequentemente, os algoritmos de codificação e descodificação requerem um buffer que contém apenas duas linhas de pixeis que precedem imediatamente o pixel atual.

Opera em dois modos, modo binário e modo de tom contínuo, que permite a distinção entre os dois tipos de imagem, sendo importante devido às metodologias de compressão amplamente diferentes empregadas em cada modo. O primeiro usa codificação por previsão, enquanto o último codifica valores diretamente do pixel. CALIC seleciona um dos dois modos, dependendo se a vizinhança local do pixel atual tem ou não mais de dois valores de pixel distintos. No modo binário, um codificador aritmético ternário adaptativo context-based é usado para codificar três símbolos, incluindo um símbolo de escape. No modo de tom contínuo, o sistema tem quatro componentes principais integrados: previsão, seleção de contexto e quantização, cancelamento de polarização de erros de previsão baseado em contexto e codificação de entropia condicional de erros de previsão. O design de dois modos contribui para a universalidade e robustez do CALIC numa ampla gama de imagens.

C. *LOCO-I (Low Complexity Lossless Compression for Images)*

LOCO-I é o novo algoritmo padrão de ISO/ITU para compressão não destrutiva de imagens com um "tom contínuo". Este é contemplado como um algoritmo com um rácio de compressão muito semelhante ao da compressão aritmética, além de que este rácio é pouco inferior ao obtido nos melhores métodos disponíveis. Além disso, LOCO-I tem uma complexidade baixa, tanto a nível do código como de modelação, no paradigma do universal modeling, combinando a sua simplicidade com a compressão potencial de modelos de contexto.

Este método de compressão é baseado num contexto fixo simples, que tem a capacidade das técnicas mais avançadas para encontrar dependências de alta-demanda. Este modelo é modificado para uma melhoria na eficiência, em conjunto com alguns códigos-tipo da família Golomb e extensões alfabéticas incorporadas para a compressão de áreas de imagens com baixa entropia.

D. *PNG (Portable Network Graphics)*

O método PNG fornece um formato de imagem de alta resolução, mas isso significa que há muito espaço para melhorias na compressão. O processo de compressão do PNG, feito em duas etapas, Previsão e Compressão, é realizado sem perdas, o que significa que a partir da imagem comprimida podemos reconstruir a imagem original. O PNG é bom para imagens com grandes áreas da mesma cor ou poucas variações da cor, e que possuam texto ou pequenos pormenores. Apesar disso o rácio de compressão é pequeno.

O primeiro processo deste método é chamado de filtering (Previsão). Este utiliza a codificação delta para todas as linhas de pixeis, sendo que uma linha pode ser o canal vermelho de uma imagem, por exemplo, onde um certo pixel é codificado de acordo com os pixeis vizinhos (à sua esquerda, acima e na diagonal para cima à esquerda). De acordo com as características de cada linha de pixels, o PNG tem diferentes modos em que o filtering pode ser feito, de maneira a que se consiga o menor número de símbolos únicos possíveis. Após o filtering inicia-se a compressão utilizando o Deflate, que salvaguarda alguns casos. O tamanho das "matches" é limitado entre 3 e 258 símbolos, implicando que o rácio máximo atingível seja 1032:1; caso o tamanho da "match" seja menor que 3 símbolos, existirão alguns problemas de sobrecarga para representar o símbolo.

V. Conclusão

A classificação dos algoritmos de compressão depende dos fatores que se pretendam alcançar: qualidade, taxa de compressão ou rapidez. Concluímos que em termos de taxa de compressão, a compressão aritmética é dos melhores métodos, apesar das suas limitações. No entanto, os métodos mais equilibrados são o PPM e Bzip2, que possuem uma boa taxa de compressão num pequeno espaço de tempo. Já para imagens, o método LOCO-I é o mais eficiente, pois possui uma complexidade idêntica à do FELICS mas uma compressão maior. Apesar disso, para obter uma melhor qualidade de imagem deve ser usado o PNG.

References

[1] Ficheiros fornecidos pelo professor (SOTA 1, 2, 3)

[2] Wikipedia.com

[3] https://www.hpl.hp.com/loco/HPL-98-193R1.pdf

[4] http://www.ittc.ku.edu/~jsv/Papers/HoV94.progressive_FELICS.pdf

[5] https://medium.com/@duhroach/how-png-works-f1174e3cc7b7