Codecs não destrutivos para Texto

1st Emanuel de Sousa Roque
Departamento de Eng. Informática
Universidade de Coimbra
Coimbra
uc2020227336@student.uc.pt

2nd Ricardo Rafael Ferreira Guegan
Departamento de Eng. Informática
Universidade de Coimbra
Coimbra
uc2020211358@student.uc.pt

3rdDuarte Fonte-Santa
Departamento de Eng. Informática
Universidade de Coimbra
Coimbra
uc2018274628@student.uc.pt

***Index Terms*—codificar, descodificar, lossless, compressão de texto**

I. Introdução

Neste documento expomos e explicamos diversos métodos de compressão lossless de texto, descrevendo sucintamente o seu funcionamento e a sua eficácia aplicada a diferentes tipos de texto(texto literários, texto de codificação, caracteres aleatórios...).Esperamos conseguir aprofundar o nosso conhecimento sobre o método de funcionamento de codecs não destrutivos para texto bem como decidir quais os codecs que iremos utilizar.

II. Algoritmos de compressão não destrutivos para texto

A. Transformadas

Estes algoritmos não são métodos de compressão por eles próprios, mas servem para evidenciar as redundâncias/padrões presentes numa fonte de informação e assim conseguir baixar a entropia. Em determinados casos a eficácia resultante de um método de pré-processamento seguido de um método de compressão justifica o tempo e custos computacionais utilizados pelo método de pré-processamento. Estes métodos não revelam grande eficácia para textos de caracteres aleatórios.

1) Transformada de Burrows-wheeler: O algoritmo consiste em pegar num bloco(cadeia de símbolos) e formar todas as combinações possíveis usando um shift cíclico, posteriormente ordenam-se estas combinações lexicograficamente pela primeira letra de cada string, a transformada é a cadeia gerada pela última coluna destas combinações.
O método é tão mais eficaz quanto maior for o tamanho dos blocos e quanto maior for a probabilidade de ocorrência de determinados símbolos(língua inglesa a probabilidade de ocorrência da letra ”e” é bastante superior à probabilidade de ocorrência da letra ”z”). No entanto, quanto maior for a dimensão do bloco maiores serão os custos computacionais e o tempo necessário para a execução do algoritmo. [1]

2) Move-to-Front: O algoritmo tem uma lista com todos os símbolos do abecedário da fonte de informação e ao ler um caractere da fonte de informação codifica-o com o index no qual ele aparece na lista, posteriormente coloca-o na primeira posição da lista(index=0) e os restantes movem-se uma posição para a direita. Repete este processo até ao final da string.
Este algoritmo é bastante vantajoso para situações em que conjuntos de caracteres aparecem frequentemente juntos (exemplo “th” na língua inglesa). Útil para códigos de programação pois estes utilizam regularmente as mesmas keywords(“if,for”). [4]

B. Técnicas de compressão baseadas em dicionário

Estas técnicas são baseadas no princípio de que cadeias repetidas anteriormente terão grande probabilidade de se repetirem novamente no futuro. Geralmente estes métodos atingem taxas de compressão mais elevadas do que os métodos de codificação entrópicos pois a utilização do contexto permite reduzir a entropia. [8]

1) LZ77: Utiliza um sistema de janela móvel, esta é constituída por uma look-ahead window que terá os caracteres a codificar e uma search-window que corresponde aos n caracteres transmitidos mais recentemente na qual se irá procurar a maior sequência idêntica à sequência a ser transmitida(”greedy parsing”). O código devolvido segue o formato (número de bits a recuar, número de bits a copiar, próximo símbolo). [2]
Este algoritmo não é tão eficiente como outros algoritmos(lzw), quando ainda não foram transmitidos caracteres é extremamente ineficiente, o tempo de codificação aumenta imenso com o tamanho da search window, o que faz com que se tenha de trabalhar com uma pequena sample size o que também irá limitar a eficácia do método. No entanto é um método simples e eficaz para textos nos quais se repete constantemente as mesmas cadeias de símbolos(”finance”). [8]

2) LZW: Para codificar uma mensagem faz uma comparação símbolo a símbolo e vê se no dicionário existe alguma entrada na qual exista este símbolo, se sim repete este processo até à maior entrada do dicionário possível, se não codifica os caracteres presentes na maior

entrada do dicionário com o index e adiciona uma nova entrada no dicionário na qual tem estes caracteres mais o carácter seguinte.(o dicionário começa pré-preenchido com todos os elementos do alfabeto)
É um algoritmo bastante eficiente para compressão de documentos em que se repetem as mesmas palavras("jquery") contudo para documentos de caracteres aleatórios irá ter uma baixa eficiência. [1]

C. Técnicas de compressão entrópicas

São técnicas de compressão que procuram atingir a entropia mínima considerando a independência total entre os símbolos do alfabeto.

1) Huffman: Começa por ordenar os símbolos por ordem decrescente de probabilidade de ocorrência, depois agrupam-se os 2 simbolos com menor probabilidade (o de maior probabilidade é codificado com 1 e o outro com 0), criando um novo símbolo cuja probabilidade é a soma das probabilidades dos dois. Este novo símbolo é colocado no seu local de acordo com a ordem decrescente de probabilidades e o processo repete-se até que a soma de probabilidades seja 1, originando a árvore de Huffman. [1]
Este algoritmo possui uma elevada eficiência e gera um código ótimo ($H(S) \leq L < H(S)+1$) contudo existem algoritmos que permitem alcançar melhores taxas de compressão tais como LZ77 ou LZ78 [8]
Existem diversos algoritmos complexos que utilizam codificação huffman juntamente com outros algoritmos para obterem melhores taxas de compressão.

2) Aritméticos: Para cada símbolo do alfabeto atribui-se um número de 0 até n de acordo com a ordem que eles aparecem substitui os números pelos seus códigos, e posteriormente converte esses números num número binário de vírgula fixa com tamanho suficiente para preservar a precisão. [1]
Este algoritmo é dos que obtêm melhores resultados ralativamente a tecnicas de codificação entrópica($L < h(x)+2/n$) contudo tem uma elevada complexidade em relação aos outros métodos de compressão entrópica o que se irá traduzir num maior gasto de recursos computacional. [8]

D. outros

1) RLE(run-length-encoding): Este algoritmo baseia-se em reduzir o tamanho de cadeias de caracteres repetidos(também chamadas de "run"), codificando-a em 2 bytes o primeiro representa o número total de caracteres na "run"("run count"), e o segundo é o carácter original. Apenas é feita substituição quando existem 2 ou mais caracteres idênticos consecutivos.
Este algoritmo é de simples e rápida execução podendo tornar-se uma boa alternativa em relação a algoritmos demasiado complexos contudo não atinge os mesmos rácios de compressão e está limitado pela existência de repetições de símbolos consecutivos(o que não é comum em qualquer tipo de texto). Para o utilizar com eficácia seria necessário utilizar uma transformada previamente. [1]

E. Complexos

Os algoritmos complexos resultam da junção de vários algoritmos de compressão numa determinada ordem o que poderá resultar num melhor valor de compressão quando comparado com a utilização de apenas um algoritmo. Ao utilizar um algoritmo complexo é necessário ter em conta os gastos computacionais e a complexidade da junção de vários algoritmos.

1) Deflate (utilizado em arquivos zip): Este algoritmo divide a informação a ser codificada em blocos e cada bloco pode ser codificado de uma maneira diferente(sem qualquer compressão, compressão com LZ77 e depois comprimido com Huffman fixo ou compressão com LZ77 e depois comprimido com Huffman dinâmico)(a codificação de Huffman é realizada em várias etapas). [3]
Tem um menor tempo de execução quando comparado com bzip2 ou LZMA contudo não permite alcançar taxas de compressão tão elevadas como estes dois. [7]

2) bzip2: Bzip 2 utiliza diversos algoritmos seguidos uns dos outros. Começa por fazer Run-length encoding posteriormente utiliza transformada de Burrows-wheeler, move-to-front e volta a usar RLE e por último utiliza codificação huffman. [5]
Tem uma eficácia superior quando comparado com algoritmo Deflate mas a sua execução é consideravelmente mais demorada. [7]

3) LZMA (Lempel Ziv Markov Algorithm) : Baseia-se num melhoramento do algoritmo LZ77 que utiliza um "history buffer" maior, optimal parsing(utiliza look-ahead para considerar a codificação de uma cadeia menor do que a disponível para codificar, desde que depois seja possível codificar uma cadeia maior, fazendo com que a soma do comprimento destes códigos seja inferior ao resultante por greedy parsing), códigos menores para cadeias de símbolos repetidas recentemente,literal exclusion after matches" e codificação aritmética. [6]
Apesar de atingir um dos melhores valores de compressão o tempo de execução é consideravelmente superior a outros algoritmos (bzip,defalte,...). [7]

III. ALGORITMOS DE COMPRESSÃO NÃO DESTRUTIVOS DESENVOLVIDOS

A. Variante Compress

Inspirado no LZW, a "variante compress" tenta simular um dicionário real que guardará todas as palavras separadas por um símbolo. Posteriormente o dicionário será ordenado de acordo com o número decrescente de ocorrências de cada palavra(palavras mais frequentes ficam codificadas com

indexs menores) e finalmente substituiremos o texto pelo index no dicionário.
Neste novo algoritmo o número de indexs é limitado pelo número de palavras diferentes pelo que o tamanho do dicionário não irá crescer para valores muito elevados.

B. Myhuffman

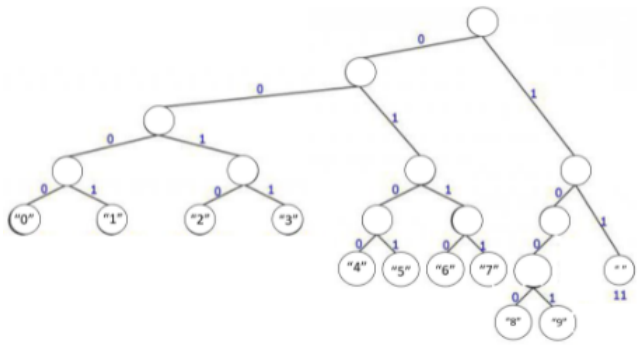

Fig. 1. Árvore fixa de Myhuffman

O algoritmo de myhuffam é basicamente a codificação de um texto no qual só existem números (0-9) usando uma árvore de huffman fixa. Na qual cada número é representado por 4 bits e o símbolo de separação(num texto normal é o espaço) representado por 11.

IV. Algoritmos selicionados para a compressão dos diferentes ficheiros

A. Random.txt

Como o nome indica, o ficheiro random.txt é uma sequência de caracteres aleatória.
Por conseguinte, não existe qualquer padrão, o que faz com que a aplicação de métodos de codificação por dicionário seja ineficaz. Nestas situações os melhores métodos de codificação são os entrópicos. Tendo estas características em mente decidimos codificar o ficheiro Random.txt usando apenas o algoritmo de huffman.
Neste caso optamos por não utilizar qualquer transformada uma vez que a transformada "Move-to.Front"(MTF) não iria evidenciar quaisquer resultados(devido à aleatoriedade dos caracteres) e a utilização da transformada de Burrows Wheeler(TBW) iria apenas colocar em evidência as redundâncias o que para codificação huffman não iria trazer qualquer aumento de eficácia.

B. Jquery-3.6.0..js

O ficheiro Jquery-3.6.0.js é um ficheiro de código, pelo que partilha das redundâncias de qualquer texto de código, sejam essas, a repetição de keywords como "for" "if", o que faz com que métodos de dicionários sejam os mais apropriados.
Para codificar este ficheiro optamos pela utilização de um algoritmo de dicionário (LZW) e para codificar o resultado deste algoritmo utilizamos Huffman à semelhança do algoritmo de Deflate que também comprime o resultado gerado por um algoritmo de dicionário(LZ77) utilizando Huffman.
No nosso caso optamos por LZW ao em vez de LZ77 pois os algoritmos de LZ77 são mais exigentes a nível computacional e têm tempos de execução consideravelmente superiores quando comparados com o algoritmo de LZW.
Não utilizamos qualquer transformada, pois no caso do jquery não é de esperar a existência de elevados números de repetição de caracteres pelo que a TBW não iria apresentar grande eficácia.
No caso do MTF o carácter extra para sinalizar onde começa e acaba o número resultante da codificação não compensa os ganhos que se possam vir a obter utilizando este algoritmo.

C. Bible.txt

O ficheiro "Bible.txt" é um texto em inglês, pelo que partilha das redundâncias de qualquer texto coerente, sejam essas, a existência de caracteres e de certos conjuntos de caracteres/palavras que possuem uma elevada taxa de ocorrência ("e" ," ","and" e "then") pelo que algoritmos de dicionários irão obter um rácio de compressão mais elevado do que os métodos entrópicos.
Partindo do princípio que todas as palavras são separadas por um espaço utilizamos o "variante Compress" que permitirá reduzir(na maioria das palavras, especialmente nas mais frequentes) o número de símbolos para representar uma palavra.
No ficheiro bible utilizamos este novo algoritmo que origina um dicionário e o texto codificado que posteriormente serão codificados utilizando um algoritmo de Huffman para cada um.
Optamos por fazer árvores de huffman diferentes para o dicionário e para o texto codificado pois para textos de dimensões consideráveis(como o "finance e o bible")o número de bits poupado excede o número de bits necessário para guardar a nova árvore binária.
Como este nosso algoritmo de dicionário foi criado especificamente para codificar palavras, ele irá ter uma maior eficiência na compressão de documentos de texto quando comparado com o algoritmo de LZW(algo que não se verifica no jquery uma vez que nem sempre são usados os mesmo conjuntos de caracteres separados por espaços, pois num texto de código antes de existir um espaço podem estar vários símbolos (" if(" é diferente de "if")).
Não utilizamos qualquer transformada antes do nosso algoritmo de dicionário pois o objetivo do nosso algoritmo é encontrar o início e o fim das palavras e se utilizássemos qualquer tipo de transformada iríamos deixar de ter esta informação.

D. *finance.csv*

O Finance.csv é um conjunto de dados financeiros, e todas as linhas obedecem ao mesmo formato, o ano da transação, o nível, o montante, o nome da companhia e o código da companhia. Este é um ficheiro com elevada redundância, principalmente na parte do ano de transação no nome da companhia e no código da companhia. Pelo que os métodos que beneficiam de padrões devem ser os que têm uma melhor performance.(algoritmos de dicionários).
À semelhança do ficheiro de bible.txt existem conjuntos de caracteres que se repetem contudo para o Finance reparamos que conjuntos de caracteres separados por vírgulas se repetem com uma maior frequência comparativamente com conjuntos de caracteres separados por espaços pelo que utilizamos o mesmo algoritmo que foi utilizado no bible(”variante compress e posteriormente huffman”), contudo em vez de separar as “palavras” pelos espaços, separamos pelas vírgulas(”,”).

V. Resultados

Para a obtenção dos seguintes resultados todos os ficheiros foram convertidos para ficheiros de texto(.txt)

A. *random*

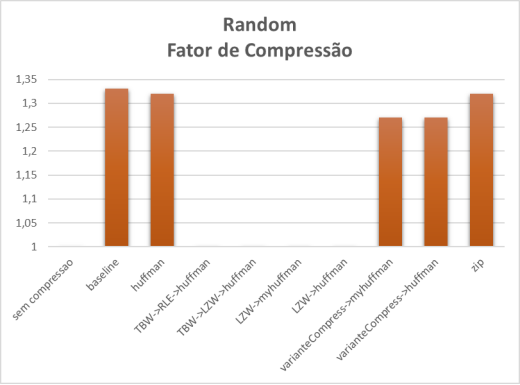

Fig. 2. Fatores de Compressão para o ficheiro random

Como seria expectável no ficheiro random não é possível alcançar fatores de compressão muito elevados. Contudo, tanto o algoritmo de huffman(escolhido para codificar este ficheiro) como os ficheiros comprimidos gerados pelo zip conseguem obter um valor ligeiramente inferior ao valor obtido pelo baseline.
A aplicação de TBW com RLE seguido de huffman não possibilita a compressão uma vez que não existe redundâncias suficientes de caracteres para que o RLE seja eficaz, estando neste caso a aumentar o número de símbolos necessários para representar os caracteres.
Qualquer algoritmo de LZW não possibilita a compressão tal como seria de prever, pois não existem padrões de caracteres em número significativo para que um algoritmo de dicionário seja eficiente.
O nosso algoritmo de dicionário (varianteCompress) possibilita a compressão pois ao separar os conjuntos de caracteres pelos espaços estamos a gerar um dicionário basicamente idêntico ao ficheiro original pelo que na prática é como se estivessemos a aplicar meramente um algoritmo de huffman ao ficheiro. O fator de compressão é inferior ao de huffman pois no varianteCompress é gerado para além do dicionário outro ficheiro que contém os indexs do dicionário(possibilita a reconstrução do ficheiro original).
Com os resultados obtidos podemos concluir que dos algoritmos desenvolvidos, o que escolhemos para comprimir o ficheiro random(huffman) é o que obtém melhor fator de compressão.

B. *Jquery*

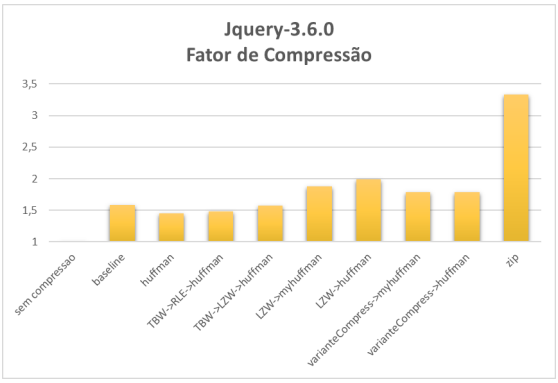

Fig. 3. Fatores de Compressão para o ficheiro jquery

Como expectável no ficheiro jquery os algoritmos de dicionário obtêm melhores fatores de compressão.(inclusivé melhor que o baseline)
Os algoritmos que utilizam LZW são mais eficazes neste tipo de texto devido às limitações do algoritmo varianteCompress (os mesmos conjuntos de caracteres têm de ser separados pelo mesmo símbolo, o que nem sempre se verifica num ficheiro de código).
Pelos resultados obtidos verificamos que ao comprimir o ficheiro com zip obtemos um fator de compressão muito superior, o que significa que este algoritmo(deflate) explora melhor o contexto no qual aparecem os caracteres.(neste tipo de texto um algoritmo de LZ77 obtêm valores significativamente superiores a um algoritmo de LZW)
Com os resultados obtidos podemos concluir que dos algoritmos desenvolvidos, o que escolhemos para comprimir o ficheiro jquery-3.6.0(LZW-huffman) é o que obtém melhor fator de compressão.

C. *Bible*

Como expectável no ficheiro bible os algoritmos de dicionário obtêm melhores fatores de compressão.(inclusivé melhor que o baseline)
Os algoritmos que utilizam a varianteCompress têm uma

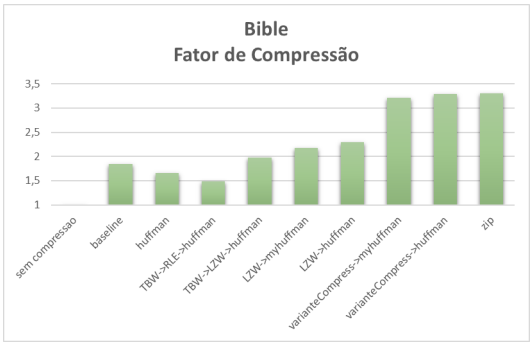

Fig. 4. Fatores de Compressão para o ficheiro bible

maior eficiência na compressão de documentos de texto quando comparado com o algoritmo de LZW uma vez que este algoritmo (varianteCompress) foi especificamente criado para codificar palavras.
Pelos resultados obtidos verificamos que tanto o zip como a varianteCompress conseguem explorar de forma eficaz a repetição de palavras em textos literários.
Com os resultados obtidos podemos concluir que dos algoritmos desenvolvidos, o que escolhemos para comprimir o ficheiro bible(varianteCompress-huffman) é o que obtém melhor fator de compressão.

D. Finance

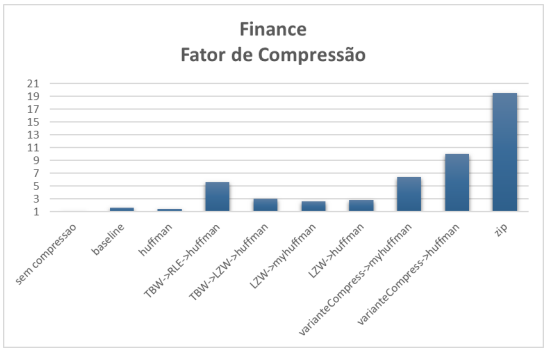

Fig. 5. Fatores de Compressão para o ficheiro bible

Como expectável no ficheiro finance é possível obter elevados fatores de compressão devido à elevada redundância na informação e os algoritmos de dicionário são os que obtêm melhores fatores de compressão.(inclusivé melhor que o baseline).
Os algoritmos que utilizam a varianteCompress que foi criada especificamente para codificar sequências separadas por um símbolo que se repete(neste caso “,”), têm um maior fator de compressão quando comparado com o algoritmo LZW.
A aplicação da TBW à fonte de informação evidencia as redundâncias dos caracteres regularmente repetidos, o que possibilitará a obtenção de melhores resultados comparativamente com os mesmos algoritmos sem esta transformada.Para além disso podemos observar que neste caso a utilização de um RLE após a TBW permite obter um fator de compressão mais elevado, ou seja, os caracteres repetem-se frequentemente.
No entanto, a utilização desta transformada antes do algoritmo varianteCompress não fará sentido uma vez que este foi criado especificamente para separar conjuntos de caracteres separados por determinados símbolos(TBW iria levar á perda desta informação).
Neste ficheiro grandes cadeias de caracteres aparecem frequentemente repetidas, próximas umas das outras pelo que o algoritmo LZ77 utilizado no zip permite obter valores consideravelmente superiores aos valores obtidos pelo varianteCompress.
Com os resultados obtidos podemos concluir que dos algoritmos desenvolvidos, o que escolhemos para comprimir o ficheiro finance(varianteCompress-¿huffman) é o que obtém melhor fator de compressão.

E. Observações gerais

O fator de compressão obtido pelo algoritmo de huffman é sempre inferior ao baseline devido H(S)(usado no cálculo do baseLine) $\leq$ L $<$H(S)+1 (para cada símbolo).
A utilização da codificação huffman em vez da myhuffman para a codificação dos resultados da codificação em dicionário permite obter melhores resultados pois a codificação huffman gera código ótimos e os algarismos que se repetem mais frequentemente são codificados usando um número menor de bits pelo que o número de bits aqui poupado é bastante superior ao número de bits necessário para codificar a árvore de huffman.
Na maioria dos ficheiros(todos excepto o random) comprimir primeiramente com um algoritmo de dicionário e posteriormente com huffman permite obter os melhores fatores de compressão.

VI. Trabalho Futuro

Gostaríamos então de evoluir os nossos métodos para se adaptarem a alguns dos problemas que conseguimos identificar. No que diz respeito à variantecompress esta depende do símbolo de separação usado, e para diferentes ficheiros o símbolo não é sempre o mesmo. Pelo que um dos pontos a otimizar seria adaptar o código de maneira a que explorasse melhor o contexto no qual surgem os conjuntos de caracteres.
Outro dos problemas a resolver seria arranjar uma forma de descodificar os ficheiros criados por este algoritmo sem ter a necessidade de criar ficheiros separados para os dicionários.

References

[1] Neha Sharma, Jasmeet Kaur, Navmeet Kaur “A Review on various Lossless Text Data Compression Techniques” pp. 58–63; 2014 Research Cell:An International Journal of Engineering Sciences
[2] http://multimedia.ufp.pt/codecs/compressao-sem-perdas/codificacao-baseada-em-dicionarios/lz77/
[3] https://www.cs.ucdavis.edu/ martel/122a/deflate.html
[4] https://www.geeksforgeeks.org/move-front-data-transform-algorithm/

[5] https://en.wikipedia.org/wiki/Bzip2
[6] http://mattmahoney.net/dc/dce.html?Section 523
[7] https://linuxreviews.org/Comparison of Compression Algorithms
[8] Robert Lourdusamy, Senthil Shanmugasundaram “A Comparative Study Of Text Compression Algorithms“ International Journal of Wisdom Based Computing, Vol. 1 (3), December 2011 pp. 68–76;