8
Huffman Randomizado

8.1
Introdução

Neste capítulo será apresentado o algoritmo RHUFF - Huffman Randomizado. Este algoritmo é uma variante do algoritmo de Huffman que define um procedimento de cripto-compressão que aleatoriza a saída. O objetivo é gerar textos cifrados aleatórios como saída para obscurecer as redundâncias do texto original (confusão). O algoritmo possui uma função de permutação inicial, que dissipa a redundância do texto original pelo texto cifrado (difusão).

Este trabalho foi publicado como relatório técnico *PUC-RioInf.MCC49/04* December, Brazil, 2004.

8.2
Descrição do Algoritmo

Esta seção tem por finalidade descrever o algoritmo de cripto-compressão. As alterações propostas são a inclusão de randomização aos dados comprimidos e a cifra por permutação inicial.

É proposta uma menor perda na taxa de compressão através de uma quebra em blocos do texto e a transformação dos blocos em textos de tamanho uma potência de 2 para tornar a codificação perfeita.

São empregados os códigos Huffman canônicos de homofônicos de comprimento variável para codificar o texto original.

Proposição 8.1 *Em uma árvore ótima, para uma distribuição diádica, todo nó de nível x tem probabilidade 2^{-x}.*

Prova. Por indução no número de folhas.

Da proposição 8.1, tem-se que os códigos canônicos de Huffman podem ser rapidamente construídos in tempo $O(n)$. Agora, é mostrado um resultado que generaliza um resultado anterior de Klein (8) sobre códigos de Huffman.

Proposição 8.2 *Codificação de prefixo ótima resulta em um fluxo aleatório de bits.*

Prova. A probabilidade de chegada de um novo bit corresponde a uma transição de um nó interno q de nível x da árvore para um de seus descendentes u ou v, ambos de nível $x+1$. Tem-se probabilidades condicionais de transição $P(u|q) = \frac{P(u \cap q)}{P(q)} = \frac{P(u)}{P(q)}$, dado que assume-se que a entrada é independente. Analogamente, $P(v|q) = \frac{P(v)}{P(q)}$. Da proposição 1, tem-se que $P(u|q) = \frac{2^{-(x+1)}}{2^{-x}} = 2^{-1} = \frac{1}{2}$ and $P(v|q) = \frac{1}{2}$.

Note que foi assumido que a entrada é independente, isto é, $P(u \cap q) = P(u)$, o que não é verdade para textos em linguagem natural. Então, alguma função de difusão deve ser aplicada. Neste trabalho, é aplicada uma permutação inicial para conseguir isso.

Teorema 8.3 *Codificação de Huffman canônica para uma distribuição diádica resulta em um fluxo aleatório de bits.*

Prova. Imediata da proposição 8.2.

A arquitetura do algoritmo é descrita a seguir conforme ilustração da figura 8.1.

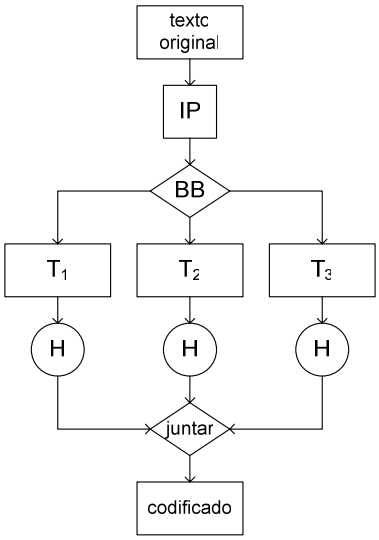

Figura 8.1: Arquitetura do algoritmo

8.2.1 Permutação Inicial (IP)

Uma varredura em palavras é realizada no texto original a fim de gerar a instância permutada, calcular o número total de símbolos N e criar o alfabeto.

É necessário eliminar a correlação que existe na linguagem natural do texto de forma a garantir a randomização de entrada necessária para usar a proposição 2. Isso é conseguido pela permutação do texto de entrada. O vetor de permutação $P = (p_1, p_2, ..., p_k)$ é empregado para permutar a

entrada em k blocos. Por exemplo, o texto original $T = bcbaadaacaadaaba$ permutado com $P = (4, 3, 1, 2)$ se transforma em $T = abbcaaaddacaabaa$. Essa permutação é simplesmente inversível e não causará nenhum efeito nas funções IRS (Information Retrieval Systems) tais como indexação ou busca.

8.2.2 Quebra em Blocos (BB)

A fim de evitar decomposição infinita, o tamanho do texto original é artificialmente aumentado para a próxima potência de dois, ou seja, é desejável que o tamanho do texto seja $N = 2^x$, onde x é algum inteiro positivo. Nesse caso, cada símbolo tem probabilidade $p_i = \frac{f_i}{2^x}$, onde f_i é a frequência do símbolo que leva a uma decomposição finita.

Para garantir que o texto tenha tamanho igual a alguma potência de dois, símbolos nulos são inseridos para completar o tamanho. Caso N não seja uma potência de dois, $2^{\lceil logN \rceil} - N$ símbolos nulos são inseridos. No entanto, esse procedimento pode até mesmo dobrar o tamanho original do texto, então como forma de contornar o problema, o texto original é quebrado em três blocos menores.

Os passos são os seguintes,

- quebre o texto original T em três blocos T_1, T_2 e T_3 de tamanhos $n_1 = 2^{\lfloor logN \rfloor}$, $n_2 = 2^{\lfloor log(N-n_1) \rfloor}$ e $n_3 = N - (n_1 + n_2)$;
- os textos T_1 e T_2 são de tamanho iguais a potências de dois, então complete apenas o bloco T_3 com $2^{\lceil log\, n_3 \rceil} - n_3$ símbolos nulos;

Com esse procedimento os símbolos nulos estarão apenas no bloco T_3 minimizando assim a expansão do texto.

Por exemplo, para a tabela 8.1, se $N = \mid T \mid = 23$ tem-se que $n_1 = 2^{\lfloor log23 \rfloor} = 16$, $n_2 = 2^{\lfloor log(23-16) \rfloor} = 4$ e $n_3 = 23-(16+4) = 3$. Então, $2^{\lceil log3 \rceil}-3 = 1$ símbolo nulo deverá ser inserido no bloco T_3.

O número de símbolos nulos inseridos (PAD - *padding*) é tal que

$$0 \leq PAD < 2.(N - 2^{\lfloor logN \rfloor} - 2^{\lfloor log(N-2^{\lfloor logN \rfloor}) \rfloor})$$

8.2.3 Função H

Cada bloco é então codificado com a função H, baseada em códigos de Huffman, que envolve as seguintes cinco operações.

Tabela 8.1: Frequências relativas a $T = bcbaabaacaabaababcaacac$.

Bloco	Tamanho	f_a	f_b	f_c	f_{nulo}
1	16	9	5	2	0
2	4	2	1	1	0
3	3+1	1	0	2	1

(i) *Varredura em blocos* - para cada bloco, realizar o *parse* novamente para recalcular as novas frequências relacionadas com os tamanhos dos textos n_1, n_2 e n_3.

(ii) *Geração da distribuição diádica* - para cada bloco, para cada símbolo no bloco, gerar a sua distribuição diádica pela decomposição de sua probabilidade em potências negativas de dois. Cada componente da distribuição diádica é um homofônico para o símbolo.

(iii) *Geração dos códigos de Huffman canônicos* - para cada bloco, para cada homofônico, gerar os códigos de Huffman canônicos. Usar o fato de que o comprimento do código é a potência absoluta de dois indices para cada símbolo de probabilidade.

Nesse momento existe um único dicionário, mas três possíveis frequências para cada símbolo. Um símbolo pode aparecer em um ou mais blocos tendo frequências relativas diferentes para cada bloco. Suponha $T = bcbaabaacaabaababcaacac$ com 23 simbolos, a Tabela 8.1 mostra um exemplo das frequências nos blocos.

(iv) *Codificação do texto original* - para cada símbolo escolha um dos seus homofônicos e retorne seu código de Huffman canônico para o texto codificado. Essa escolha deve ser aleatória. É necessário o emprego de algum gerador de números aleatórios para realizar esta operação.

(v) *Cifragem do modelo* - após codificação do texto original tem-se um texto comprimido e um modelo. Então, o modelo deve ser escondido por alguma cifra de bloco tal qual DES (29) ou AES (27).

8.2.4 Junção dos blocos

Ao final do processo, o algoritmo faz a junção dos três blocos em um arquivo final codificado.

A seguir são apresentados experimentos com a cadeia comprimida de bits usando a suite de testes estatísticos para geradores de números aleatórios e pseudo aleatórios para as aplicações criptográficas ("A statistical test suite

for random and pseudorandom number generators for cryptographic applications") do NIST (26).

Os ataques estatísticos são evitados com o uso de substituição homofônica (5, 11) e com a ambiguidade dos códigos de Huffman destacada em (28). Entretanto, outras estratégias de segurança podem ser empregadas para enrigecer os dados cifrados, como as descritas por Milidiú et al. em (14, 15, 13).

8.3 Experimentos

Esta seção apresenta os resultados de alguns experimentos com o algoritmo de cripto-compressão. O foco principal é avaliar as perdas pela compressão dos dados e a randomização do texto cifrado.

Nos experimentos foram utilizados textos extraidos do Projeto GUTENBERG (6). Os arquivos utilizados constituem um conjunto representativo de textos na língua inglesa. Os textos são escolhidos a partir dos 100 mais populares downloads da lista de livros eletrônicos clássicos: *Moby Dick*, Herman Melville [MEL]; *The Notebooks of Leonardo Da Vinci - Complete*, Leonardo da Vinci [LDV]; *Ulysses*, James Joyce [JOY]; *Don Quixote*, Miguel de Cervantes Saavedra [CER]; *The Count of Monte Cristo*, Alexandre Dumas [DUM]. Esses textos foram codificados com o algoritmo proposto e testados com *Statistical Test Suite for Random and Pseudorandom Number Generators v.1.7* (26). Os testes estatísticos aplicados foram:

- *Frequency Test*
- *Block Frequency Test*
- *Cumulative Sums Test*
- *Runs Test*
- *Longest Run of Ones Test*

Todos os textos cifrados passaram com sucesso em todos os testes do experimento.

A tabela 8.2 mostra as taxas de compressão de dados alcançadas, onde $|S|$ representa o tamanho do texto original e $|C|$ o tamanho do texto cifrado em bits. O tamanho do texto cifrado exclui o tamanho do modelo. A taxa média é de aproximadamente 30-40%, a qual é compatível com o algoritmo original de Huffman de compressão de dados por palavra (huffword). A tabela 8.3 mostra mais estatísticas detalhadas dos arquivos de texto cifrados: o número de palavras processadas e o tamanho de cada um dos três blocos codificados. A tabela 8.4 complementa a tabela 8.3 mostrando o número total de símbolos

Tabela 8.2: Arquivos texto da coleção de GUTENBERG.

Arquivos	$\|S\|$	$\|C\|$	Taxa $\frac{\|C\|}{\|S\|}$
MEL	1,227 KB	467 KB	38.06%
LDV	1,395 KB	459 KB	32.90%
JOY	1,526 KB	519 KB	34.01%
CER	2,293 KB	751 KB	32.75%
DUM	2,627 KB	1,024 KB	38.98%

(palavras) por bloco: w_1, w_2 e w_3. O número total de palavras e w_3 incluem os símbolos nulos inseridos. A tabela 8.5 compara o número de bits por símbolo (palavra) obtidos pelo algoritmo de cripto-compressão $\frac{|C|}{Palavras}$ e os obtidos pelo Huffman palavra (huffword). Note que Massey et. al (11) estimaram que para a substituição homofônica de comprimento variável, similar ao esquema utilizado aqui, se U é o dicionário de entrada de símbolos e V a substituição homofônica, então o limite de entropia obtido é $H(U) \leq H(V) < H(U) + 2$. Os experimentos mostram que o algoritmo de cripto-compressão adiciona 1 bit a mais no pior caso.

Finalmente, a tabela 8.6 sumariza o número total de símbolos nulos inseridos no bloco T_3, a taxa de expansão relativa ($\frac{PAD}{Palavras}$) devido à inserção desses símbolos nulos, e a expansão em bits/símbolo relacionada a huffword. Com a finalidade de comparar essa expansão total, foi utilizada nos experimentos a ferramenta *ZipNFind*, que foi implementada com a *C Library to search over compressed texts* desenvolvida em (34) para calcular os tamanhos dos arquivos codificados por um compressor Huffman de palavra. Pode-se perceber pela tabela que o algoritmo de cripto-compressão mostra pequenas perdas em taxa de compressão apenas nos casos em que o número de símbolos nulos inseridos (padding) é relativamente grande. A expansão PAD relacionada ao tamanho dos textos originais é $\frac{PAD}{N} < 2 - \frac{2^{\lfloor logN \rfloor} + 2^{\lfloor log(N - 2^{\lfloor logN \rfloor}) \rfloor})}{N}$. Nesse caso, os experimentos mostram uma taxa de expansão em bits/símbolo de aproximadamente 5 a 10%. Além disso, a expansão total de dados cifrados é quase zero nos experimentos.

Tabela 8.3: Arquivos texto em detalhe.

Arquivos	$\mid T_1 \mid$	$\mid T_2 \mid$	$\mid T_3 \mid$
MEL	240 KB	182 KB	47 KB
LDV	234 KB	116 KB	111 KB
JOY	511 KB	7 KB	2 KB
CER	466 KB	232 KB	54 KB
DUM	476 KB	359 KB	190 KB

Tabela 8.4: Número de símbolos/bloco.

Arquivos	Palavras	w_1	w_2	w_3
MEL	433,692	262,144	131,072	65,536
LDV	505,929	262,144	131,072	131,072
JOY	538,532	524,288	8,192	8,192
CER	845,850	524,288	262,144	65,536
DUM	948,404	524,288	262,144	262,144

Tabela 8.5: Comparando o desempenho bits/palavra.

Arquivos	Cripto-compressão	Huffword
MEL	8.82 bits	8.37 bits
LDV	7.43 bits	8.26 bits
JOY	7.89 bits	8.91 bits
CER	7.27 bits	7.61 bits
DUM	8.84 bits	8.02 bits

Tabela 8.6: Expansão de símbolos nulos.

Arquivos	PAD em T_3	$\frac{PAD}{Words}$	Expansão
MEL	25,060	%5,78	5,38%
LDV	18,359	%3,63	-10,05%
JOY	2,140	%0,40	-11,45%
CER	6,118	%0,72	-4,47%
DUM	100,172	%10,56	12,97%

8.4 Conclusões

Pelos resultados dos experimentos é possível concluir que o algoritmo de cripto-compressão alcança taxas de compressão médias de aproximadamente 30-40%, a expansão média fica em torno de 5 a 10% em relação a huffword, e que as saídas geradas são na verdade bits aleatórios. Resultados empíricos foram executados com as ferramentas do NIST (26), o que mostrou na prática os resultados teóricos do teorema 8.3, que o uso de distribuição diádica leva a bits aleatórios.