SOBRE CÓDIGOS CORRETORES DE ERROS

Natália Pedroza de Souza

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientadores: Jayme Luiz Szwarcfiter
Paulo Eustáquio Duarte Pinto

Rio de Janeiro
Maio de 2018

SOBRE CÓDIGOS CORRETORES DE ERROS

Natália Pedroza de Souza

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Jayme Luiz Szwarcfiter, Ph.D.

Prof. Paulo Eustáquio Duarte Pinto, D.Sc.

Prof. Fábio Protti, D.Sc.

Prof. Franklin de Lima Marquezino, D.Sc.

Prof. Luerbio Faria, D.Sc.

Prof. Valmir Carneiro Barbosa, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
MAIO DE 2018

Souza, Natália Pedroza de

Sobre Códigos Corretores de Erros/Natália Pedroza de Souza. – Rio de Janeiro: UFRJ/COPPE, 2018.

IX, 69 p. 29, 7cm.

Orientadores: Jayme Luiz Szwarcfiter

Paulo Eustáquio Duarte Pinto

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2018.

Referências Bibliográficas: p. 66 – 69.

1. Códigos. 2. Correção de erros. 3. Hamming. I. Szwarcfiter, Jayme Luiz *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Agradecimentos

Gostaria de agradecer a minha família por todo apoio de sempre. Aos amigos vindos da UERJ, que também foram para a UFRJ, e tornaram esse novo mundo familiar, Israel, Evandro, Brunno e Pedro. Aos novos amigos do PESC, Danis, Hugo, Rebeca, Spencer e Renan, pelo interesse em discutir o problema da minha tese, juntamente com o amigo Alexandre no SPSchool. Ao amigo Alexsander por me ajudar inúmeras vezes com o latex e a todo o pessoal do laboratório de algoritmos. Ao meu maior amigo, Deni, por todas as revisões, traduções e por me acompanhar em cada passo dessa jornada.

Aos membros da banca, professores Valmir Carneiro, Luerbio Faria, Fábio Protti e Franklin Marquezino. Finalmente aos meus orientadores Jayme e Paulo. Ao Jayme, primeiramente por ter aceitado me orientar, por toda a paciência, por toda a liberdade para a escolha o tema da tese, por toda sua sabedoria, humildade e todo ensinamento. E ao professor Paulo Eustáquio o agradecimento mais especial. Tê-lo escolhido como orientador de mestrado foi a melhor decisão da minha vida acadêmica. Muito obrigada por ter estado comigo lá em 2011 e estar até hoje. São 7 anos convivendo com essa paixão pela ciência, essa dedicação, esse simples prazer de pesquisar, de descobrir, de programar. Foi maravilhoso e motivante cada vez que vibramos juntos por um programa rodar, por cada resultado que descobrimos, cada demonstração que finalizamos. Dá até uma pontinha de tristeza em pensar que tudo isso está acabando. Mas fica uma gratidão que não cabe em mim. O senhor é uma grande inspiração, eu lhe admiro muito. Muito, muito obrigada.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

SOBRE CÓDIGOS CORRETORES DE ERROS

Natália Pedroza de Souza

Maio/2018

Orientadores: Jayme Luiz Szwarcfiter
Paulo Eustáquio Duarte Pinto

Programa: Engenharia de Sistemas e Computação

Neste trabalho discutimos dois assuntos principais. Na primeira parte, desenvolvemos dois códigos corretores de erros, classificados como códigos de Hamming encurtados, $Gham(n)$ e $BP(n)$. Estes códigos são ótimos para distância Hamming 3, isto é, têm capacidade de corrigir 1 erro. Chamamos de códigos ótimos, os códigos que apresentam o maior número de palavras-código, dados um comprimento n das palavras-código e uma distância Hamming d. Apresentamos as construções recursivas dos códigos $Gham(n)$ e $BP(n)$ e seus algoritmos de codificação e decodificação com complexidade $O(n)$.

Na segunda parte, discutimos a construção de Códigos Corretores de Erro de Comprimento Variável (VLECC) e mostramos que seu custo pode ser menor do que o dos correspondentes de comprimento fixo, mesmo quando a distribuição de frequência dos símbolos a serem codificados é uniforme.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

ON ERROR-CORRECTING CODES

Natália Pedroza de Souza

May/2018

Advisors: Jayme Luiz Szwarcfiter
Paulo Eustáquio Duarte Pinto

Department: Systems Engineering and Computer Science

In this work we discuss two main issues. In the first part, we developed two error-correcting codes, classified as shortened Hamming codes, $Gham(n)$ and $BP(n)$. These codes are optimal for Hamming distance 3, that is, are able to correct 1 error. We call optimal codes, the codes that have the largest number of codewords, given a codeword length n and a distance Hamming d. We present the recursive constructions of the $Gham(n)$ and $BP(n)$ and their encoding and decoding algorithms with complexity $O(n)$.

In the second part, we discuss the construction of Variable Lenght Error Correcting Codes (VLECC) and show that their cost may be lower than the corresponding fixed length code, even when the frequency distribution of the symbols to be encoded is uniform.

Sumário

Lista de Tabelas

Capítulo 1

Introdução

Nesse capítulo, descremos a motivação para o trabalho, os principais resultados obtidos, incluindo as publicações, a organização da tese e os conceitos básicos de códigos corretores de erros.

1.1 Motivação

Imagine que uma pessoa recebe um e-mail de seu colega de trabalho com a seguinte mensagem:

Reunião na quaeta-feira às 10h?

Esta pessoa imediatamente supõe que a reunião é na quarta-feira. Em português e em qualquer outra língua, as palavras são *strings* formadas por letras tiradas de um alfabeto finito. Mas nem todas as *strings* de letras formam uma palavra com algum significado: "quaeta" não corresponde a nenhuma palavra, por isto detectamos o erro. Ao trocarmos a letra "e" pela letra "r", temos uma palavra em português. Trocar a letra "e" por qualquer outra letra, não nos daria significado algum à palavra. Assim, concluímos que a palavra é "quarta". É claro que o remetente da mensagem, poderia ter pensado em escrever "quinta", e ter cometido dois erros ao digitar, mas isso seria muito mais improvável.

O envio de mensagens por de meio eletrônico é realizado através de uma codificação. Assim, além do erro de digitação, podem ocorrer outros erros na transmissão de uma mensagem, chamados de *ruídos*, causados por exemplo, por interferências electromagnéticas. Vamos pensar num exemplo clássico. Se quiséssemos enviar apenas duas mensagens: SIM e NÃO. Poderíamos, por exemplo, codificar o SIM com a palavra-código 1 e o NÃO com a palavra-código 0. Mas neste caso, se houvesse algum erro na transmissão de uma mensagem, o SIM seria transformado em NÃO e vice-versa e o erro não seria detectado. Uma solução para esse problema seria

uma codificação diferente: codificar o SIM com 11 e o NÃO com 00. Imagine que a mensagem SIM é enviada, porém ocorre um erro na transmissão e a mensagem recebida é 10. O receptor detecta que houve um erro, pois 10 não corresponde a nenhuma palavra-código, mas não é capaz de corrigi-la. Por fim, uma nova codificação poderia ser feita: codificar o SIM com 111 e o NÃO com 000. Neste caso, se ocorresse um erro na transmissão do SIM, e o receptor recebesse o vetor 110, não só o erro seria detectado, como seria corrigido. Supondo que o canal raramente introduz erros, o vetor 110 seria associado a palavra-código 111, pois assim, apenas um erro teria ocorrido (mais provável do que dois erros terem ocorrido) e portanto, a mensagem seria decodificada como SIM.

A teoria de códigos corretores de erros apresenta as características necessárias a um código para que tenha tal capacidade de detecção e correção de erros. Apresenta também as relações entre as quantidades de símbolos a serem codificados e o comprimento das palavras-códigos. E ainda, processos de construção dos códigos, de codificação e decodificação. O parâmetro que define a capacidade de correção de erro é a *distância Hamming*. A primeira parte deste trabalho descreve dois códigos corretores de erros para distância Hamming 3 com características específicas.

A teoria de códigos de comprimento variável estuda técnicas de compressão de dados que visam agilizar o processo de transmissão por meio da redução de redundância dos dados. Finalmente, a teoria dos códigos corretores de erros de comprimento variável (variable length error-correcting codes - VLECC) busca unificar estes dois objetivos: compactar os dados e introduzir redundância para que erros possam ser detectados e corrigidos.

Definimos um código VLEC *ótimo* quando a média do comprimento das palavras que o formam é a menor possível. O problema de construir tais códigos de forma eficiente ainda encontra-se em aberto. A segunda parte do estudo feito neste trabalho aborda este problema.

Buscamos, especificamente, construir um código binário que possa detectar e corrigir quantos erros se deseje e, ao mesmo tempo, considerar as probabilidades de ocorrência dos símbolos utilizados para a otimização dos comprimentos médios das codificações.

Restringimos o estudo a códigos binários, pois estes são os utilizados na prática. Consideramos todos os requisitos para se ter um bom código, que são: palavras com os menores comprimentos possíveis (menor média de comprimento das palavras), para uma transmissão rápida; a possibilidade de se utilizar o maior número de palavras possíveis, para que haja uma grande variedade de mensagens a serem transmitidas e, por fim, que o maior número de erros possíveis possa ser corrigido (e detectado).

1.2 Contribuições e conteúdo

Nessa seção apresentamos as principais contribuições resultantes do trabalho da tese que se concentrarão em códigos corretores de distância Hamming igual a 3.

- Desenvolvemos uma generalização para os códigos Hamming binários, $Gham(n)$, que são códigos Hamming encurtados. Apresentamos a criação de tais códigos de maneira recursiva. Demonstramos que os códigos são lineares e têm capacidade de corrigir um erro. E, finalmente, apresentamos algoritmos de codificação e decodificação com complexidade $O(n)$. Estas complexidades descritas constituem um avanço em relação aos resultados existentes. Em particular não encontramos na literatura nenhum algoritmo de decodificação com complexidade linear.

- Desenvolvemos também um segundo código Hamming encurtado denominado *código binário posicional*, denotado por $BP(n)$. Apresentamos inclusive uma criação recursiva do mesmo. E ainda algoritmos de codificação e decodificação com complexidade $O(n)$. Conforme mencionado, não encontramos na literatura algoritmos com tal complexidade.

- Desenvolvemos uma técnica adicional de criação gulosa dos códigos mencionados que foi estendida para a criação de códigos não lineares, alguns dos quais ótimos. Além disso, a técnica foi generalizada para códigos lineares de distâncias 5 e 7 até $n = 22$.

- Propusemos duas famílias de códigos corretores de erros de comprimentos variáveis. Estes códigos significaram uma melhoria em relação ao estado atual da arte. Um resultado surpreendente é que estes códigos apresentam um custo menor do que os códigos correspondentes de comprimento fixo.

As seguintes publicações resultaram diretamente do trabalho dessa tese:

- "*Códigos corretores de erros de tamanhos variáveis*", XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, 2015 [33].

- "*Error correcting codes and cliques of graphs*", Latin American Workshop on Cliques in Graphs, em 2016 [34].

- "*Uma Generalização dos códigos Hamming*", II Encontro de Teoria da Computação, 2017 [36].

- "*Variable length error correcting codes*", São Paulo School of Advanced Science on Algorithms, Combinatorics and Optimization, 2016 [35].

- "*Sobre códigos corretores de erros de distância Hamming 3*", Latin American Workshop on Cliques in Graphs, em 2018 (submetido).
- "*On correcting codes of Hamming distance 3*", IEEE Transactions on Information Theory, 2018 (manuscrito).
- "*Families of variable length error correcting codes*", IEEE Transactions on Communications, 2018 (manuscrito).

A seguir descrevemos a organização desta tese.

Nesta introdução, fazemos um estudo dos códigos corretores de erros de comprimento fixo já propostos. Principalmente os códigos lineares, que utilizam matrizes em suas construções, possibilitando assim uma forte estruturação matemática das propriedades de tais códigos.

No Capítulo 2, apresentamos as principais propriedade de códigos lineares, que são uma classe especial de códigos corretores de erro. Onde se enquadram os códigos desenvolvidos neste trabalho nos Capítulos 4 e 5.

No Capítulo 3, fazemos um estudo sobre a intratabilidade de problemas relacionados a códigos corretores de erros. E ainda, descrevemos o problema de encontrar um código corretor de erros ótimo como um problema equivalente ao de encontrar clique máxima em determinado grafo particular.

No Capítulo 4 apresentamos uma generalização para os códigos Hamming binários, $Gham(n)$, que são códigos Hamming encurtados.

No Capítulo 5 apresentamos um segundo código Hamming encurtado denominado *código binário posicional*, denotado por $BP(n)$.

No Capítulo 6, descrevemos três métodos já desenvolvidos para se construir VLECC's mais relevantes encontrados na literatura. Primeiramente apresentamos o trabalho de Victor Buttigieg [12] feito em sua tese na década de 90. Depois o de Ting-Yi Wu [42], que apresentou em 2011 um algoritmo que encontra um VLEC ótimo dada uma distância livre. Finalmente, em 2013, Richa Gupta [18] apresentou algoritmos de codificação de decodificação de VLECC, que são bem eficientes, porém os VLECC's encontrados não apresentam média de comprimento das palavras satisfatório.

No Capítulo 7 apresentamos duas famílias códigos corretores de comprimento variável acima comentadas.

No Capítulo 8 apresentamos as conclusões e discutimos trabalhos futuros.

Na próxima seção apresentamos as definições pertinentes relacionadas a códigos corretores de erros.

1.3 Códigos corretores de erros

Seja um conjunto $A = \{a_1, a_2, \cdots, a_M\}$ de **símbolos** a serem codificados. Considere o conjunto $F_q = \{0, 1, 2, \cdots, q\}$ denominado **alfabeto**. Um **código** $q-$ário é um conjunto $C = \{c_1, c_2, \cdots, c_M\}$, onde cada c_i, chamado de **palavra**, é formado por uma sequência de elementos – *string* – de F_q. A cada elemento $a_i \in A$ corresponde um, e somente um, elemento $c_i \in C$.

O **comprimento** de uma palavra, $|c_i|$, é definido como o número de elementos que formam da palavra. No caso dos códigos de comprimento fixo, como o nome sugere, todas as palavras têm o mesmo comprimento n, isto é, $c_i \in F_q^n$.

O **tamanho** de um código C é determinado pela quantidade M de palavras do código.

Iremos trabalhar apenas com o alfabeto $F_2 = \{0, 1\}$. Portanto, ao longo do texto, utilizaremos a palavra código para nos referir a códigos binários. Por simplicidade, vamos denotar o alfabeto F_2 por F e chamaremos cada elemento do conjunto F de *bit*.

Exemplo 1.1 *Código com $M = 4$ palavras de comprimento $n = 5$:*

$$C = \{00000, 01101, 10110, 11011\}.$$

A **distância Hamming** entre duas palavras c_1 e c_2, de mesmo comprimento, é definida pelo número de posições em que elas diferem entre si e denotada por $d(c_1, c_2)$.

Exemplo 1.2 *A distância Hamming entre as palavras* 01101 *e* 11011 *é 3, pois as palavras diferem na 1^a, 3^a e 4^a posição.*

A **distância Hamming d de um código** C é a menor das distâncias Hamming entre duas palavras quaisquer de C. Denotada por $d(C)$.

Exemplo 1.3 *Considere o código*

$$C = \begin{matrix} 00000 \\ 01101 \\ 10110 \\ 11011 \end{matrix}$$

Temos que $d(C) = 3$, pois $d(00000, 11011) = 4$, $d(01101, 10110) = 4$ e a distância entre quaisquer outras duas palavras do código é 3.

O conceito de distância Hamming é muito importante para a teoria de códigos, pois determina a capacidade de detecção e correção de erros do código, como mostra o teorema a seguir.

Teorema 1.4 *[20] Um código C com distância Hamming $d = 2t + 1$ ou $d = 2t + 2$ é capaz de:*

1. *Detectar até $d - 1$ erros;*

2. *Corrigir até t erros.*

Demonstração:

1. Suponha que uma palavra c é transmitida e até $d - 1$ erros ocorrem. O vetor recebido não pode ser uma palavra do código, assim os erros são detectados.

2. Suponha que uma palavra c é transmitida e o vetor y é recebido com até t erros, assim, $d(c, y) \leq t$. Se c' é outra palavra do código, então $d(c', y) \geq t+1$. Caso contrário, $d(c', y) \leq t$, o que implica, pela desigualdade triangular, que $d(c, c') \leq d(c, y) + d(c', y) \leq 2t$, contradizendo $d(C) = 2t+1$ ou $d(C) = 2t+2$.

□

O **peso** w de uma palavra é o número de bits não nulos da palavra. O peso de um código C é o menor dos pesos das palavras de C, exceto da palavra composta apenas por 0's.

Exemplo 1.5 *Considere o código $C = \{00000, 01101, 10110, 11011\}$.*
Temos $w(01101) = 3$, $w(10110) = 3$ e $w(11011) = 4$, portanto $w(C) = 3$.

Dados $x = x_1x_2 \cdots x_n$ e $y = y_1y_2 \cdots y_n$ em F^n, a operação **ou exclusivo** é definida como $x \oplus y = (x_1 + y_1, x_2 + y_2, \cdots, x_n + y_n)$, onde a soma é efetuada em módulo 2. E o **produto** é dado por $x \odot y = (x_1 \cdot y_1, x_2 \cdot y_2, \cdots, x_n \cdot y_n)$, onde as operações são feitas em módulo 2.

Exemplo 1.6 $01101 \oplus 10110 = 11011$.

Lema 1.7 *Se x e $y \in F^n$, então $d(x, y) = w(x \oplus y)$.*

É fácil ver que $x \oplus y$ tem 1 apenas nas posições onde x e y diferem.

Lema 1.8 *Se x e $y \in F^n$, então $d(x, y) = w(x) + w(y) - 2w(x \odot y)$.*

A distância entre x e y é dada pelo número de 1's de x mais o número de 1's de y menos o número de posições em que x e y são iguais a 1.

Teorema 1.9 *Seja d um número ímpar. Um código binário com M palavras de comprimento n com distância Hamming d existe se, e somente se, um código binário com M palavras de comprimento $n + 1$ com distância Hamming $d + 1$ existe.*

Demonstração: $\Rightarrow$) Suponha um código binário com M palavras de comprimento n e distância Hamming d, onde d é um número ímpar. O código $\hat{C}$ de comprimento $n+1$ é obtido de C adicionando-se um bit de paridade a cada palavra $c = c_1c_2\cdots c_n \in C$, isto é, $\hat{c} = c_1c_2\cdots c_nc_{n+1}$, onde $c_{n+1} = \sum_{i=1}^{n} c_i \mod 2$.

Como $w(\hat{c})$ é par para toda palavra $\hat{c} \in \hat{C}$, segue do Lema 1.8 que $d(\hat{x}, \hat{y})$ é par para todo $\hat{x}, \hat{y} \in \hat{C}$. Portanto, $d(\hat{C})$ é par. Como $d \leq d(\hat{C}) \leq d+1$ e d ímpar, concluímos que $d(\hat{C}) = d+1$.

$\Leftarrow$) Suponha um código D com M palavras de comprimento $n+1$ e distância Hamming $d+1$, onde d é um número ímpar. Tome $x, y \in D$ tais que $d(x, y) = d+1$. Escolha uma posição em que x e y diferem e apague esta posição de todas as palavras. O resultado é um código com M palavras de comprimento n e distância Hamming d. □

O problema principal dos códigos corretores de erros envolve os 3 parâmetros n, M e d. Para q fixo, um bom código corretor de erros tem os menores valores para n e os maiores valores para M e d possíveis. Dado dois desses parâmetros, como otimizar o terceiro? A maneira mais comum de se abordar este problema é determinar o valor máximo de M, denotado por $A_q(n, d)$, tal que existe um código de comprimento fixo q-ário com M palavras de n bits e distância Hamming d. A Tabela 1.1 apresenta alguns valores para $A_2(n, d)$. Muitos dos valores são apresentados como uma faixa contendo limites inferiores e superiores, pois os resultados exatos ainda encontram-se em aberto. Cada entrada dessa tabela é um problema em particular, [1, 6, 13, 15, 19, 23–26, 28, 30–32, 40], as referências atualizadas de cada um dos valores encontram-se em http://www.win.tue.nl/~aeb/codes/binary-1.html.

Note que basta trabalharmos com d par ou ímpar. Pois do Teorema 1.9, temos o corolário a seguir.

Corolário 1.10 *Se d é ímpar, então $A_2(n+1, d+1) = A_2(n, d)$.*

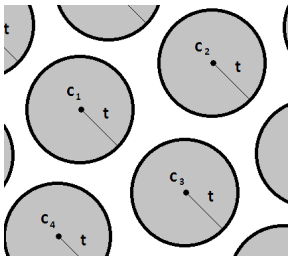
Para $u \in F^n$ e um inteiro $r \geq 0$, a **bola** fechada de raio r e centro u é dada por:

$$B[u, r] = \{v \in F^n | \ d(u, v) \leq r\}.$$

Para um código com distância $d \geq 2t+1$, as bolas de raio t centradas nas palavras de C são disjuntas. Assim, todo vetor y com distância de no máximo t a uma dada palavra c pertence a $B[c, t]$.

Tabela 1.1: Valores $A_2(n, d)$ dados n e d

n \ d	3	5	7	9	11	13	15
5	4	2	1	1	1	1	1
6	8	2	1	1	1	1	1
7	16	2	2	1	1	1	1
8	20	4	2	1	1	1	1
9	40	6	2	2	1	1	1
10	72	12	2	2	1	1	1
11	144	24	4	2	2	1	1
12	256	32	4	2	2	1	1
13	512	64	8	2	2	2	1
14	1024	128	16	4	2	2	1
15	2048	256	32	4	2	2	2
16	2816-3276	256-340	36	6	2	2	2
17	5632-6552	512-673	64-72	10	4	2	2
18	10496-13104	1024-1237	128-135	20	4	2	2
19	20480-26168	2048-2279	256	40	6	2	2
20	40960-43688	2560-4096	512	42-47	8	4	2
21	81920-87333	4096-6941	1024	64-84	12	4	2
22	147456-172361	8192-13674	2048	80-150	24	4	2
23	327680-344308	16384-24106	4096	136-268	48	6	4
24	2^{19}-599184	17920-47538	4096-5421	192-466	52-55	8	4
25	2^{20}-1198368	32768-84260	4104-9275	384-836	64-96	14	4
26	2^{21}-2396736	65536-157285	8192-17099	512-1585	128-169	28	6
27	2^{22}-4792950	131072-291269	16384-32151	1024-2817	178-288	56	8



Lema 1.11 *Uma bola de raio r em F^n contém exatamente*

$$\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{r}$$

vetores.

Demonstração: Seja u um vetor fixo de F^n. Considere os vetores v com distância exatamente m de u, onde $m \leq n$. As m posições em que v difere de u podem ser escolhidas de $\binom{n}{m}$ maneiras.

□

Teorema 1.12 *Um código com M palavras de comprimento n e distância Hamming $d = 2t + 1$ satisfaz:*

$$M\left[\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{t}\right] \leq 2^n.$$

Demonstração: Quaisquer duas bolas de raio t centrada em palavras distintas não têm vetores em comum. Portanto, o número total de vetores nas M bolas de raio t centradas nas M palavras é dado pelo lado direito da igualdade. Este número deve ser menor ou igual a 2^n que é o número total de vetores de F^n. □

Um código é dito **perfeito** se é um código onde as bolas de raio t centradas nas palavras do código preenchem exatamente todo o espaço.

No capítulo a seguir apresentamos conceitos básicos sobre códigos lineares [4, 27].

Capítulo 2

Códigos Lineares

Seja F_q um corpo de q elementos. Um código C q-ário é dito $[n, k, d]$-código linear (ou $[n, k]$-código), se é um subespaço vetorial de dimensão k de F_q^n. Isto é, C é um código linear se, e somente se:

1. $u + v \in C,\ \forall\ u$ e $v \in C$ e
2. $au \in C,\ \forall\ u \in C,\ a \in F_q$.

Em particular, para o caso de $q = 2$, um código é linear se, e somente se, a soma em módulo 2 de duas palavras quaisquer do código é também uma palavra do código.

Uma das propriedades mais úteis dos códigos lineares é de que sua distância Hamming é dada pelo menor dos pesos das palavras não-nulas do código.

Teorema 2.1 *[27] Se C é um código linear, então $d(C) = w(C)$.*

Demonstração: Existem palavras x e y de C tais que $d(C) = d(x, y)$. Note que $d(x, y) = w(x - y)$. Logo, $d(C) = w(x - y) \geq w(C)$, pois $x - y$ é uma palavra do código.

Por outro lado, para alguma palavra $x \in C$, $w(C) = w(x) = d(x, 0) \geq d(C)$. Note que 0 pertence a todo código linear. Como $d(C) \geq w(C)$ e $d(C) \leq w(C)$, concluímos que $d(C) = w(C)$.

□

Portanto, uma vantagem dos códigos lineares é que, para determinar a distância do código, precisamos examinar o peso de $M - 1$ palavras ao invés de examinar a distância da combinação dois a dois das M palavras do código.

Uma segunda vantagem é que para especificar um código linear $[n, k, d]$, ao invés de listar as M palavras, podemos simplesmente fornecer uma base de k palavras que gera o código.

Uma matriz $k \times n$ cujas linhas formam uma base de um $[n, k]$-código é chamada de **matriz geradora** do código.

Exemplo 2.2 *O código do Exemplo 1.3 é um* $[5, 2, 3]$*-código com matriz geradora.*

$$G = \begin{bmatrix} 01101 \\ 10110 \end{bmatrix}$$

Observação: Um código linear só está definido se q é uma potência de primo, porém possível construir códigos se q não é potência de primo a partir dos códigos lineares.

Codificação

Seja C um $[n, k]$-código q-ário sobre F_q com matriz geradora G. C contém q^k palavras, portanto podemos utilizá-lo para transmitir q^k mensagens distintas. Estas serão identificadas com as q^k k-uplas de F_q^k. Codificamos uma mensagem $u = u_1u_2 \cdots u_k$ multiplicando-a à direita por G. Assim, uG é uma palavra de C, sendo uma combinação linear das linhas da matriz geradora.

A codificação é ainda mais simples se escrevemos a matriz geradora G da forma $G = [I_k|A]$, onde A é uma matriz $k \times (n - k)$, chamada de **forma padrão**. Neste caso, os primeiros k dígitos da palavra são idênticos aos dígitos do vetor mensagem que a gerou, chamados de dígitos de informação. Os dígitos restantes são chamados de dígitos de paridade ou redundância.

Exemplo 2.3 *A forma padrão de uma matriz geradora do* $[7, 4]$*-código é dada por:*

$$G = \begin{bmatrix} 1000101 \\ 0100111 \\ 0010110 \\ 0001011 \end{bmatrix}$$

O vetor mensagem 1000 é codificado como 1000101.

Decodificação

Suponha que uma palavra $x = x_1x_2 \cdots x^n$ é enviada e o vetor $y = y_1y_2 \cdots y^n$ é recebido. Definimos o **vetor erro** como

$$e = y - x.$$

O decodificador deve decidir a partir de y que palavra x foi transmitida, ou de forma equivalente, que vetor erro e ocorreu. Uma esquema de decodificação de desenvolvido por Slepian [38] utiliza o fato de que um código linear é um subgrupo aditivo do grupo F_q^n. Vamos a algumas definições antes de mostrarmos o esquema.

Se C é um $[n,k]$-código linear e a um vetor em F_q^n. Então $a+C=\{a+x|x\in C\}$ é a classe lateral de C.

O teorema a seguir é um caso particular do teorema de Lagrange para subgrupos.

Teorema 2.4 (Lagrange) *Se C é um código linear, temos:*

- *todo vetor de F_q^n está em uma classe lateral de C;*
- *toda classe lateral contém exatamente q^k vetores;*
- *as classes laterais são disjuntas.*

Exemplo 2.5 *Seja o $[4,2]$-código $C=\{0000,1011,0101,1110\}$ com matriz geradora*

$$G=\begin{bmatrix}1011\\0101\end{bmatrix}$$

Então as classes laterais são:

$$\begin{aligned}0000+C&=C\\1000+C&=\{1000,0011,1101,0110\}\\0010+C&=\{0100,1111,1101,1010\}\\0001+C&=\{0010,1001,0111,1100\}\end{aligned}$$

O vetor de peso mínimo de uma classe lateral é chamado de **líder da classe lateral** (se houver mais de um vetor, escolhemos um para ser o líder). No Exemplo 2.5, 0001 é uma alternativa para líder da classe lateral $0001+C$.

Uma **tabela padrão** de um código C é uma matriz de todos dos vetores de F_q^n, onde a primeira linha é formada pelas palavras de C com o $\mathbf{0}$ como primeiro elemento. Para cada linha restante, escolhe-se um vetor a_i de menor peso não listado na tabela, este vetor é o tomado como o primeiro elemento da linha, e os elementos seguintes são gerados somando-se este elemento a_i com as palavras da primeira linha, ordenadamente.

Exemplo 2.6 *A matriz padrão do código do Exemplo 1.3 é dada por:*

$$\begin{array}{cccc}0000&1011&0101&1110\\1000&0011&1101&0110\\0010&1111&0001&1010\\0001&1001&0111&1100\end{array}$$

Na decodificação, quando um vetor y é recebido, ele deve ser encontrado na tabela padrão. E o vetor é decodificado como a palavra na coluna correspondente

a y. Este processo é chamado de decodificação por vizinhança máxima (maximum likelihood decoding).

Este é um exemplo de um esquema de decodificação de vizinhança mais próxima. Descrevemos este método apenas de maneira ilustrativa, pois na prática ele é muito lento e custoso. A seguir, vamos apresentar um outra maneira de decodificar um código.

Decodificação por síndrome

Antes de apresentar a decodificação por síndrome, precisamos de algumas definições.

Dado um código linear C, define-se o complementar $C^{\perp}$, dito **código dual** de C, como:

$$C^{\perp} = \{u \in F_q^n \mid \langle c, u\rangle = 0, \text{ para todo } c \in C\},$$

onde $\langle c, u\rangle$ representa o produto interno entre c e u.

Lema 2.7 *Se C é um $[n,k]$-código linear com matriz geradora G, então*

$$u \in C^{\perp} \Leftrightarrow uG^T = 0,$$

onde G^T é a matriz transposta de G.

Demonstração: A ida é imediata, pois as linhas de G são palavras de C.

Para a volta, suponha que as linhas de G são $r_1, \cdots, r_k$ e $\langle u, r_i\rangle = 0$ para todo i. Se u é uma palavra de C, então $u = \sum_{i=1}^{k} \lambda_i r_i$ para escalares λ_i e então:

$$\begin{aligned} \langle c, u\rangle &= \sum_{i=1}^{k} \lambda_i \langle u_i, r_i\rangle \\ &= \sum_{i=1}^{k} \lambda_i 0 = 0. \end{aligned}$$

Assim, concluímos que u é ortogonal a toda palavra de C e portanto, $u \in C^{\perp}$. □

Teorema 2.8 *Se C é um $[n,k]$-código linear, então o código dual $C^{\perp}$ é $[n, n-k]$-código linear.*

A demonstração desse teorema será omitida aqui, pode ser encontrada em [27].

Exemplo 2.9 *Se* $C = \left\{ \begin{array}{l} 000 \\ 110 \\ 011 \\ 101 \end{array} \right.$ *então* $C^{\perp} = \left\{ \begin{array}{l} 000 \\ 111 \end{array} \right.$.

Uma matriz H é dita **matriz de paridade** de C se é uma matriz geradora de $C^{\perp}$. Portanto, H é uma matriz $(n-k) \times n$ satisfazendo $GH^T = 0$. Portanto, segue que:

$$C = \{x \in F_q^n |\ xH^T = 0\}.$$

Assim, um código linear pode ser definido através de sua matriz de paridade.

O teorema a seguir mostra uma maneira simples de construir a matriz de paridade de um código, dada sua matriz geradora.

Teorema 2.10 *Se $G = [I_k|A]$ é uma matriz geradora na forma padrão de um $[n,k]$-código C, então a matriz de paridade de C é $H = [-A^T|I_{n-k}]$.*

Exemplo 2.11 *Se o código C tem matriz geradora na forma padrão*

$$G = \left[\begin{array}{c|c} & 101 \\ I_4 & 111 \\ & 110 \\ & 011 \end{array}\right]$$

Então a matriz de paridade de C é dada por

$$H = \left[\begin{array}{c|c} 1110 & \\ 0111 & I_3 \\ 1101 & \end{array}\right].$$

Uma matriz de paridade está na forma padrão se $H = [B|I_{n-k}]$.

Em seguida, descrevemos um teorema fundamental que estabelece uma relação entre distância mínima de um código linear e uma propriedade de independência linear das colunas de sua matriz de paridade.

Teorema 2.12 *A distância Hamming mínima d de um código linear C é o tamanho do menor conjunto linearmente dependente (l.d.) de colunas de H.*

Demonstração: Seja $c = c_1 \cdots c_n$ uma palavra de peso d – que existe, pois $d(C) = w(C)$. Temos que $cH^T = 0$ pode ser expandido em $c_1h_1 + \cdots + c_nh_n = 0$, onde h_i representa a i-ésima coluna de H. A equação tem apenas d elementos não nulos. Logo, o conjunto de colunas correspondentes $\{h_i : c_i \neq 0\}$ é l.d.

Para provar que não existe um conjunto menor l. d., considere qualquer conjunto de t colunas l. d. $\{h_{\alpha_1}, \cdots, h_{\alpha_t}\}$. Existem constantes, não todas nulas, tais que $k_{\alpha_1}h_{\alpha_1} + \cdots + k_{\alpha_t}h_{\alpha_t} = 0$, então a palavra x com k_{α_i} como α_1-ésima componente e o resto nulo é uma palavra com peso pelo menos t. Portanto, $t \geq d$.

□

A seguir, vamos descrever o processo de decodificação por síndrome.

Suponha H uma matriz de paridade de um $[n, k]$-código linear C. Para todo vetor $y \in F_q^n$, o vetor:

$$S(y) = yH^T$$

é chamado de **síndrome** de y.

Note que:

1. Se as linhas de H são $h_1, \cdots, h_{n-k}$, então $S(y) = (yh_1, \cdots, yh_{n-k})$ e
2. $S(y) = 0 \Leftrightarrow y \in C$.

Lema 2.13 *Dois vetores u e v são da mesma classe lateral de C se, e somente se, têm a mesma síndrome.*

Demonstração: Sejam u e v pertencentes a mesma classe lateral
$\Leftrightarrow \quad u + C = v + C$
$\Leftrightarrow \quad u - v \in C$
$\Leftrightarrow \quad (u - v)H^T = 0$
$\Leftrightarrow \quad uH^T = vH^T$
$\Leftrightarrow \quad S(u) = S(v).$

□

Corolário 2.14 *Existe uma bijeção entre as classes laterais e as síndromes.*

Para encontrar que classe lateral contém y no processo de decodificação, é feito o seguinte processo. Calcula-se a síndrome $S(e)$ para cada classe lateral e e estende-se a matriz listando as síndromes em uma coluna extra.

Exemplo 2.15 *Um código com matriz geradora*

$$G = \begin{bmatrix} 1011 \\ 0101 \end{bmatrix}$$

E portanto, matriz de paridade

$$H = \begin{bmatrix} 1010 \\ 1101 \end{bmatrix}.$$

Assim, as síndromes dos líderes das classes laterais são:

$$S(0000) = 00$$
$$S(1000) = 11$$
$$S(0100) = 01$$
$$S(0010) = 10$$

E a matriz padrão se torna:

$$\begin{array}{lllll} 0000 & 1011 & 0101 & 1110 & 00 \\ 1000 & 0011 & 1101 & 0110 & 11 \\ 0100 & 1111 & 0001 & 1010 & 01 \\ 0010 & 1001 & 0111 & 1100 & 10 \end{array}.$$

Para decodificação então, quando um vetor y é recebido, calcula-se $S(y) = yH^T$ e localiza-se $S(y)$ na coluna das síndromes. Localize y na linha correspondente e decodifique como a palavra no topo da coluna que contém y.

Assim, é necessário apenas armazenar duas colunas: a das síndromes e a dos líderes das classes laterais. O algoritmo para a decodificação por síndrome tem complexidade $O(n\ 2^{n-k})$, já que requer uma tabela com 2^{n-k} palavras de comprimento n [3].

Em seguida, apresentamos alguns limites inferiores e superiores para $A_2(n, d)$.

Teorema 2.16 (Hamming bound) *Seja $A_2(n, d)$ o maior tamanho de um código C de comprimento n e distância Hamming d. Temos:*

$$A_2(n, d) \leq \frac{2^n}{\sum_{k=0}^{t} \binom{n}{i}}$$

onde $t = \left\lfloor \frac{d-1}{2} \right\rfloor$.

Teorema 2.17 (Singleton bound)

$$A_2(n, d) \leq 2^{n-d+1}.$$

Teorema 2.18 *Se um código C com M palavras tem distância Hamming $d < \frac{n}{2}$, então:*

$$M \leq d \cdot 2^{n-2d+2}.$$

Temos também alguns limites inferiores encontrados na literatura.

Teorema 2.19 (Gilbert-Varshamov bound) *Para quaisquer n, d existe um código linear com esses parâmetros e tamanho*

$$M \geq \frac{2^n}{\sum_{i=0}^{d-1} \binom{n}{i}}.$$

Teorema 2.20 (Griesmer bound) *Seja $n * (k, d)$ o tamanho do menor código linear binário com dimensão k e distância mínima d. Então*

$$n * (k, d) \geq \sum_{i=0}^{k-1} \left\lceil \frac{d}{2^i} \right\rceil .$$

A seguir vamos definir uma classe em particular de códigos lineares, os códigos cíclicos.

Códigos cíclicos

Um código é dito **cíclico** se:

1. É linear e
2. Se $a_0 \cdots a_{n-2} a_{n-1}$ é uma palavra do código, então $a_{n-1} a_0 \cdots a_{n-2}$ também é.

Exemplo 2.21 *O código $C = \{0000, 1010, 0101, 1111\}$ é cíclico.*

Tome o polinômio $p(x) = x^n - 1$ e considere o anel $R_n = F[x]/(x^n - 1)$ de polinômios módulo $x^n - 1$.

Como $x^n \equiv 1 \mod x^n - 1$, podemos reduzir qualquer polinômio módulo $x^n - 1$ trocando x^n por 1, x^{n+1} por x, x^{n+2} por x^2 e assim por diante.

Vamos agora identificar uma palavra $a_0 \cdots a_{n-2} a_{n-1}$ com o polinômio $p(x) = a_0 + a_1 x \cdots + a_{n-1} x^{n-1}$ em R^n.

Note que multiplicando x pelo polinômio $p(x)$, temos:

$$\begin{aligned} x \cdot p(x) &= a_0 x + a_1 x^2 + \cdots + a_{n-1} x^n \\ &= a_{n-1} + a_0 x + \cdots + a_{n-2} x^{n-1} \end{aligned}$$

que corresponde a palavra $a_{n-1} a_0 \cdots a_{n-2}$. Ou seja, multiplicar por x corresponder a fazer um deslocamento cíclico.

Teorema 2.22 *Um código C em R_n é um código cíclico se, e somente se, C é um ideal de R_n, ou seja, se satisfaz:*

1. $a(x), b(x) \in C \Rightarrow a(x) + b(x) \in C$;
2. $a(x) \in C$ *e* $r(x) \in R_n \Rightarrow r(x)a(x) \in C$.

Demonstração: $\Rightarrow$) Suponha que C é um código cíclico em R_n. Então C é linear e o item 1 é satisfeito. Agora suponha que $a(x) \in C$ e $r(x) = r_0 + r_1 x + \cdots + r_{n-1} x^{n-1} \in$

R_n. Temos que $x \cdot a(x) \in C$, logo, $x \cdot (x \cdot a(x)) = x^2 a(x) \in C$ e assim por diante. Portanto $r(x)a(x) = r_0 a(x) + r_1 x a(x) + \cdots + r_{n-1} x^{n-1} a(x) \in C$.
$\Leftarrow$) Suponha 1 e 2. Tomando $r(x)$ como um escalar, as condições implicam que C é linear. Tomando $r(x) = x$, a condição 2 mostra que C é cíclico. □

Para construir um código cíclico, basta tomarmos um polinômio $p(x) \in R_n$ e o conjunto, denotado por $\langle p(x) \rangle$ de todos os múltiplos de $p(x)$ em R_n.

Pode-se checar facilmente que este conjunto forma um ideal de R_n, logo, é um código cíclico.

Exemplo 2.23 *Considere o código $C = \langle 1 + x^2 \rangle$ em R_3. Multiplicando $1 + x^2$ pelos 8 elementos de R_3 (reduzido módulo $x^3 - 1$), geramos as palavras correspondentes a $0, 1 + x, 1 + x^2$ w $x + x^2$. Logo, $C = \{000, 110, 101, 011\}$.*

Pode-se demonstrar ainda que todo código cíclico pode ser gerado por um polinômio.

Códigos Hamming

Seja r um inteiro positivo e H uma matriz $r \times (2^r - 1)$ cujas colunas são vetores não nulos de F^r. O código cuja matriz de paridade é H é chamado de código Hamming, $Ham(r)$.

Uma definição alternativa é dada por: o código $Ham(r)$ é o conjunto de todos os $[n, k]$-códigos lineares binários cujas matrizes de paridade H tem r linhas e n colunas, onde n é o maior número de colunas possível tais que todo par de colunas é linearmente independente.

Exemplo 2.24 $Ham(3)$

A matriz de paridade pode ser escrita tomando-se todos os números binários de 1 a 7 em ordem crescente.

$$H = \begin{bmatrix} 0001111 \\ 0110011 \\ 1010101 \end{bmatrix}.$$

Para obtermos a matriz de paridade na forma padrão, tomamos as colunas em uma ordem diferente.

$$H = \begin{bmatrix} 0111100 \\ 1011010 \\ 1101001 \end{bmatrix}.$$

A matriz geradora é então dada por

$$G = \begin{bmatrix} 1000011 \\ 0100101 \\ 0010110 \\ 0001111 \end{bmatrix}.$$

Teorema 2.25 *Um código Hamming, $Ham(r)$, para $r \geq 2$,*

1. *é um $[2^r - 1, 2^r - 1 - r]$-código;*
2. *tem distância mínima 3;*
3. *é um código perfeito.*

Decodificação

As classes laterais do código $Ham(r)$ são os 2^r vetores de F^n de peso 1. A síndrome do vetor $0 \cdots 010 \cdots 0$ (com 1 na j-ésima posição) é $(0 \cdots 010 \cdots 0)H^T$, que é a transposta da j-ésima coluna de H. Portanto, se as colunas de H são escritas em ordem crescente dos números binários, então temos o seguinte algoritmo de decodificação.

1. Quando o vetor y é recebido, calcula-se a síndrome $S(y) = yH^T$;
2. Se $S(y) = 0$, então assume-se y como a palavra enviada;
3. Se $S(y) \neq 0$, então assume-se que 1 erro ocorreu. $S(y)$ é a representação binária da posição do erro e assim, este pode ser corrigido.

Exemplo 2.26

$$H = \begin{bmatrix} 0001111 \\ 0110011 \\ 1010101 \end{bmatrix}.$$

Se $y = 1101011$, então $S(y) = 110$, assume-se que ocorreu um erro na sexta posição e decodifica-se y como 1101001.

Uma das propriedades mais importantes dos códigos Hamming é que estes são os únicos códigos lineares perfeitos, como mostra o teorema a seguir.

Teorema 2.27 *Os únicos códigos não triviais lineares perfeitos com distância Hamming 3 são os códigos Hamming.*

Demonstração: Seja um código linear C perfeito com distância Hamming $d = 3$ e M palavras. Então:

$$\begin{aligned} M &= \frac{2^n}{\sum_{i=0}^{1} \binom{n}{i}} = \frac{2^n}{1+n} \\ &\Rightarrow \; 2^n = M(1+n) \\ &\Rightarrow \; M = 2^l, \text{ com } 0 \leq l \leq n. \end{aligned}$$

Se $l = 0$, então o código tem apenas uma palavra e é portanto trivial. E se, $l = n$, então o código seria todo os espaço F^n e neste caso, teríamos $d = 1$. Portanto, temos,

$$M = 2^l, \text{ com } 0 < l < n.$$

Retornando a equação,

$$\begin{aligned} 2^n &= 2^l(1+n) \\ \Rightarrow n &= 2^{n-l} - 1. \end{aligned}$$

Temos então que $dim(C) = l$ e então $dim(C^{\perp}) = n - l$. Assim, toda matriz de paridade de C tem $n - l$ linhas e n colunas com $n = 2^{n-l} - 1$. Este é precisamente o número máximo de colunas de tamanho $n - l$ tais que todo par de colunas é linearmente independente. Portanto, C é um código Hamming.

□

Códigos Hamming Estendidos

O código Hamming estendido Hâm(r) é um código obtido do $Ham(r)$ adicionando-se um bit de paridade.

A distância do código resultante aumenta de 3 para 4. E o código é também linear, isto é, o código Hâm(r) é um $[2^r, 2^r - 1 - r, 4]$-código.

Seja H a matriz de paridade do código $Ham(r)$. A matriz de paridade $\bar{H}$ do código estendido pode ser obtida de H fazendo

$$H \Rightarrow \bar{H} = \begin{bmatrix} & & 0 \\ & \text{H} & \vdots \\ & & 0 \\ 1 & \cdots \; 1 & 1 \end{bmatrix}.$$

A última linha fornece a equação de paridade, $x_1 + x_2 + \cdots + x_{n+1} = 0$.

Se H é tomada com colunas em ordem crescente, então existe um algoritmo de

decodificação que corrige um erro e detecta dois.

Códigos Hamming Encurtados

Tome todas as palavras de um código C com o mesmo símbolo na i-ésima coordenada. Excluindo a i-ésima coordenada de todas essas palavras, formamos um novo código C' de comprimento $n-1$ e mesma distância Hamming de C. C' é um *código encurtado* de C.

Se C é um $[n,k,d]$-código linear e o símbolo apagado é o 0, então o código encurtado C' é um $[n-1,k-1,d']$-código linear, com $d' \geq d$. Se H é a matriz de paridade de C, então a matriz de paridade de C' é obtida apagando a coluna correspondente de H.

Os códigos Hamming encurtados são códigos lineares ótimos para distância 3 [4]. Na verdade, a distância Hamming $d \geq 3$ exata destes códigos é desconhecida. Em [16] é apresentada a distância para alguns valores de n.

2.1 Códigos Lineares Ótimos

Vamos considerar a seguir o problema de encontrar o maior valor M de palavras, denotado por $B_2(n,d)$, tal que existe um $[n,k,d]$-código linear.

Este problema pode ser descrito de duas maneiras:

Versão 1: Para um dado comprimento n, encontre a dimensão máxima k tal que existe um $[n,k,d]$-código linear. (Neste caso, temos que para este k, $B_2(n,d) = 2^k$.)

Versão 2: Para uma dada redundância r, encontre o comprimento máximo de n tal que existe um $[n,n-r,d]$-código linear.

Mostraremos a equivalência dos problemas mais adiante.

Um (n,s)**-conjunto** em F^r é um conjunto de n vetores tais que quaisquer s vetores são linearmente independentes.

Denota-se por $max_s(r)$ o maior valor de n para o qual existe um (n,s)-conjunto em F^r. Um (n,s)-conjunto em F^r que tem $n = max_s(r)$ é chamado de ótimo. O problema de empacotamento para F^r é o que determina os valores para $max_s(r)$ e os (n,s)-conjuntos ótimos. O problema foi considerado pela primeira vez em [8] e posteriormente aplicado em teoria de códigos, como mostra o teorema a seguir.

Teorema 2.28 *Existe um $[n,n-r,d]$-código em F^r se, e somente se, existe um $(n,d-1)$-conjunto em F^r.*

Demonstração: Suponha C um $[n, n-r, d]$-código com matriz de paridade H. Pelo Teorema 2.12, as colunas de H formam um $(n, d-1)$-conjunto em F^r. Por outro lado, suponha K um $(n, d-1)$-conjunto em F^r. Se formarmos uma matriz H $r \times n$ com os vetores de K como colunas, então, novamente pelo Teorema 2.12, H é a matriz de paridade de um $[n, n-r, d]$-código cuja distância mínima é pelo menos d. □

Corolário 2.29 *Para dados valores de q, d e r, o maior valor de n para o qual existe um $[n, n-r, d]$-código sobre F é $max_{d-1}(r)$.*

Então a versão 2 do problema principal de teoria de códigos é o mesmo problema que o de encontrar $max_{d-1}(r)$. Agora, o teorema a seguir mostra que os valores de $B(n, d)$ também são dados pelas soluções desse problema.

Teorema 2.30 *Suponha $max_{d-1}(r-1) < n \leq max_{d-1}(r)$. Então $B(n, d) = 2^{n-r}$.*

Demonstração: Como $n \leq max_{d-1}(r)$, existe um $[n, n-r, d]$-código sobre F, e portanto, $B(n, d) \geq 2^{n-r}$. Se $B(n, d)$ fosse estritamente maior do que 2^{n-r}, então existiria um $[n, n-r+1, d]$-código, implicando em $n \leq max_{d-1}(r-1)$, contrariando a hipótese. □

O problema para o caso $d = 3$ está resolvido como mostrado a seguir.

Teorema 2.31 *Para uma dada redundância r, o comprimento máximo n de um $[n, n-r, 3]$-código sobre F é $2^r - 1$, isto é, $max_2(r) = 2^r - 1$.*

Demonstração: Pelo Corolário 2.29, o valor de n é $max_2(r)$, o maior tamanho de um $(n, 2)$-conjunto em F_r. Um conjunto S de vetores em F_r é um $(n, 2)$-conjunto se, e somente se, nenhum vetor em S é um múltiplo escalar de nenhum outro vetor em S. Os $2^r - 1$ vetores não nulos de F_r não são múltiplos escalares um do outro. Portanto, um $(n, 2)$-conjunto em F_r de tamanho máximo é um conjunto de $2^r - 1$ vetores. □

Os $[n, n-r, 3]$-códigos com $n = 2^r - 1$ são somente os códigos Hamming $Ham(r)$. A solução do problema principal para códigos lineares segue imediatamente dos dois Teoremas anteriores.

Teorema 2.32 *$B(n, 3) = 2^{n-r}$, onde r é o único inteiro que satisfaz $2^{r-1} - 1 < n \leq 2^r - 1$.*

Os códigos lineares ótimos para distância 3 são exatamente os códigos Hamming e os códigos Hamming encurtados [27]. Os algoritmos de decodificação podem variar bastante em termos de tempo e memória utilizada. Nos capítulos seguintes iremos

descrever maneiras alternativas de construir tais códigos e apresentaremos processos de codificação e decodificação eficientes.

Para outras distâncias, ainda existem casos em aberto. A Tabela 2.1 mostra a dimensão máxima para comprimentos $5 \leq n \leq 27$ e distância Hamming $3 \leq d \leq 15$, para d ímpar.

Tabela 2.1: Dimensão de $B(n, d)$

n \ d	3	5	7	9	11	13	15
5	2	1	–	–	–	–	–
6	3	1	–	–	–	–	–
7	4	1	1	–	–	–	–
8	4	2	1	–	–	–	–
9	5	2	1	1	–	–	–
10	6	3	1	1	–	–	–
11	7	4	2	1	1	–	–
12	8	4	2	1	1	–	–
13	9	5	3	1	1	1	–
14	10	6	4	2	1	1	–
15	11	7	5	2	1	1	1
16	11	8	5	2	1	1	1
17	12	9	6	3	2	1	1
18	13	9	7	3	2	1	1
19	14	10	8	4	2	1	1
20	15	11	9	5	3	2	1
21	16	12	10	5	3	2	1
22	17	13	11	6	4	2	1
23	18	14	12	7	5	2	2
24	19	14	12	7	5	3	2
25	20	15	12	8	6	3	2
26	21	16	13	8	7	4	2
27	22	17	14	10	7	5	3

Uma versão alternativa da Tabela 2.1 é dada na Tabela 2.2 que, para dados n e d ímpar, apresenta o valor de $B(n, d)$.

Apresentamos no capítulo a seguir as complexidades de problemas relacionados a códigos corretores.

Tabela 2.2: $B(n, d)$

n \ d	3	5	7	9	11	13	15
5	4	2	1	1	1	1	1
6	8	2	1	1	1	1	1
7	16	2	2	1	1	1	1
8	16	4	2	1	1	1	1
9	32	4	2	2	1	1	1
10	64	8	2	2	1	1	1
11	128	16	4	2	2	1	1
12	256	16	4	2	2	1	1
13	512	32	8	2	2	2	1
14	1.024	64	16	4	2	2	1
15	2.048	128	32	4	2	2	2
16	2.048	256	32	4	2	2	2
17	4.096	512	64	8	4	2	2
18	8.192	512	128	8	4	2	2
19	16.384	1.024	256	16	4	2	2
20	32.768	2.048	512	32	8	4	2
21	65.536	4.096	1.024	32	8	4	2
22	131.072	8.192	2.048	64	16	4	2
23	262.144	16.384	4.096	128	32	4	4
24	524.288	16.384	4.096	128	32	8	4
25	1.048.576	32.768	8.192	256	64	8	4
26	2.097.152	65.536	16.384	256	128	16	4
27	4.194.304	131.072	32.768	512	128	32	8

Capítulo 3

Intratabilidade de códigos

Problemas como o de encontrar algoritmos de codificação e decodificação para códigos corretores de erros de tamanhos encontram-se hoje ainda em aberto. São então feitos estudos sobre a complexidade de diversos problemas particulares relacionados a códigos corretores de erros. Iremos citar e descrever como foram feitas as demonstrações de alguns deles.

Em Berlekamp et al. [5] é demonstrado que são NP-completo os dois problemas: o de **decodificação para códigos lineares** e o de **encontrar pesos para os códigos lineares**. Estes problemas são, respectivamente, equivalentes aos problemas:

A - Classe lateral de pesos
Entrada: Uma matriz binária A, um vetor binário y e $w \in \mathbb{Z}_+$.
Propriedade: Existe um vetor x com peso menor ou igual a w tal que $xA = y$?

B - Subespaço de pesos
Entrada: Uma matriz binária A e $w \in \mathbb{Z}_+$.
Propriedade: Existe um vetor x de peso w tal que $xA = 0$?

Como vimos anteriormente, no processo de decodificação por vizinhança máxima, quando um vetor y é recebido, deve-se encontrar a solução de peso mínimo da equação $Hx = s$, onde $s = Hy$ e H é a matriz de paridade do código. Um algoritmo polinomial para esse problema implica em um algoritmo polinomial para o problema **A**. O problema **B** corresponde exatamente ao problema de decidir quando um código linear tem uma palavra de um dado peso w.

As demonstrações da NP-completude dos problemas **A** e **B** foram feitas utilizando-se o problema a seguir:

C - Emparelhamento tridimensional
Entrada: Um conjunto $U \subset T \times T \times T$, onde T é um conjunto finito.

Propriedade: Existe um subconjunto $W \subset U$ tal que $|W| = |T|$ e não existem dois elementos de W iguais em nenhuma coordenada?

Podemos descrever U com uma matriz de incidência binária $|U| \times 3|T|$ onde cada linha corresponde a uma das triplas. Assim, uma solução para C é a existência de $|T|$ linhas tais que a soma de cada linha $\mod 2$ é 1.

Reduções:

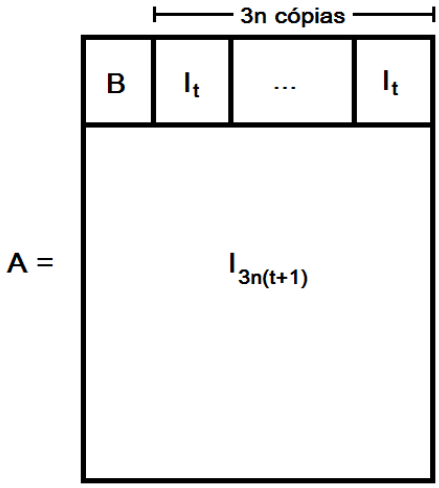
Emparelhamento tridimensional → Classe lateral de pesos

Suponha que temos um algoritmo polinomial para A. Dada uma entrada $U \subset T \times T \times T$ para o problema C, tome M a matriz de incidência $|U| \times 3|T|$. Então, rodando o suposto algoritmo para A com entradas $M, y = (111 \cdots 111), w = |T|$, temos um algoritmo polinomial para C sempre que o emparelhamento existe.

Logo, o problema A é NP-completo.

Emparelhamento tridimensional → Subespaço de pesos

Vamos construir uma matriz A para o algoritmo B a partir da matriz de incidência B de dimensão $t \times 3n$.



A é uma matriz $(3nt+3n+t) \times (3nt+3n)$ com o topo contendo t linhas consistindo da matriz B seguida de $3n$ cópias da matriz identidade $t \times t$, e na parte inferior uma matriz identidade $(3nt + 3n) \times (3nt + 3n)$.

Assuma que existe um algoritmo polinomial para o problema B. Se aplicamos este algoritmo na matriz A, e tomarmos $w = 3n^2 + 4n$, teremos um algoritmo polinomial sempre que o conjunto original tem triplas com emparelhamento.

Por hipótese, x é um vetor com $xA = 0$. Sejam x_0 o vetor formado pelas t primeiras componentes de x e x_1 formado pelas últimas $3n(t + 1)$ componentes. E seja ainda $y = x_0 B$.

Então, $|x_1| = |y| + 3n|x_0|$, onde $|x|$ denota o peso de x. Adicionando $|x_0|$ aos dois lados da equação,

$$|x| = |y| + (3n+1)|x_0|.$$

Desde que $0 \leq |y| \leq 3n$, $|x_0|$ e $|y|$ pode ser unicamente determinado a partir de $|x|$: eles são o resto e o quociente quando $|x|$ é dividido por $3n+1$. Em particular, se $|x| = 3n^2 + 4n$, temos $|x_0| = n$ e $|y| = 3n$. Então o código com a matriz de paridade A tem uma palavra de peso $3n^2 + 4n$ se, e somente se, o conjunto de triplas admite um emparelhamento.

Logo, o problema B é NP-completo.

Em [29], é demonstrado que o problema de encontrar uma palavra de peso mínimo que não é múltiplo de $k \geq 2$ em um código linear é NP-difícil. O problema é dado de maneira formal a seguir.

Peso mínimo não múltiplo de k
Entrada: Uma matriz binária A e $w \in \mathbb{Z}_+^*$, $k \in \mathbb{Z}$.
Propriedade: Existe um vetor x tal que o peso de w de x não é múltiplo de k, $0 < |x| \leq w$ e $xA = 0 \mod 2$?

O artigo demonstra que esse problema é NP-completo reduzindo do seguinte problema NP-completo:

Decodificação de código linear
Entrada: Uma matriz binária A, um vetor binário y e $w \in \mathbb{Z}_+^*$.
Propriedade: O sistema de equações lineares $xA = y \mod 2$ tem uma solução x_0 tal que $|x_0| \leq w$?

Em [2], é demonstrado que o problema de **decidir quando um código binário tem um vetor de peso** $n/2$ é NP-completo. E que são polinomiais os problemas de **decodificar códigos cíclicos** e **encontrar uma base de peso mínimo total para códigos cíclicos**.

Em [14], é demonstrado que o problema de **determinar se um grafo tem um código perfeito que corrige 1 erro** é NP-completo, reduzido de 3-SAT.

Um conjunto de palavras de um código corretor de erros pode ser visto como um subconjunto de vértices de um hipercubo. Um código corretor de erro *perfeito* é um código onde não existem duas palavras adjacentes e toda "não-palavra" é adjacente a exatamente uma palavra.

A seguir descrevemos o problema de encontrar um código corretor de erros como um problema de encontrar clique em grafos.

3.1 Problema Principal

Primeiramente, precisamos de algumas definições. Seja $d(u, v)$ a **distância** entre os vértices u e v em G, isto é, o tamanho do caminho mínimo entre u e v. O **grafo potência** d de um grafo $G = (V, E)$ é o grafo $G^d = (V, E^d)$ tal que $uv \in E^d$ se, e somente se, $d(u, v) \leq d$ [7]. Para k-cubo, a distância entre dois vértices é igual a distância Hamming entre as palavras associadas a esses vértices.

O problema de determinar o número máximo de palavras $A_2(n, d)$ de um código C com distância Hamming d e palavras de comprimento n pode ser visto como o problema de encontrar a clique máxima do grafo G. Onde os vértices de G representam todos os 2^n vetores binários de tamanho n e dois vértices de G possuem aresta entre si se, e somente se, a distância Hamming entre estes vértices é pelo menos d. Podemos descrever G como um grafo completo de 2^n vértices, onde retiramos as arestas correspondentes às distâncias $1, 2, \cdots, d-1$, estas arestas por sua vez, correspondem às arestas da $(d-1)$-ésima potência do n-cubo. De maneira formal, temos:

Problema: **Máximo de palavras**
Entrada: Um grafo $G = K_{2^n} - E[Q_n^{d-1}]$.
Propriedade: O conjunto de vértices C forma uma clique máxima?

Note que este grafo é regular. O grau de cada vértice v de G é dado por:

$$d(v) = 2^n - 1 - \sum_{i=1}^{d-1} \binom{n}{i},$$

onde $\binom{n}{i}$ representa a combinação de n i a i.

Note que a entrada do problema é exponencial em n, portanto, mesmo um algoritmo polinomial não seria eficiente.

Apresentamos nos capítulos a seguir construções recursivas para códigos lineares com distância Hamming 3.

Capítulo 4

Uma generalização dos Códigos Hamming

Um código linear é dito *ótimo* se, fixados uma distância Hamming d e um comprimento n, possui o maior número de palavras M possível. Ou, de maneira equivalente, fixados n e M, d é máximo. Vimos que o problema de decodificação de um código linear geral é NP-completo [9]. Vimos ainda, que os códigos lineares ótimos para distância 3 são exatamente os códigos Hamming e os códigos Hamming encurtados [27]. A dimensão destes códigos é dada por $B(n,3) = 2^{n-\lceil \log_2(n+1) \rceil}$ [22].

Embora esses resultados sejam antigos, não identificamos na literatura algoritmos lineares para decodificação de tais códigos. O que se encontra, em geral, são códigos que procuram maximizar o valor de d para um dado n. Porém, seus algoritmos de decodificação não são adequados para valores grandes de n. Inclusive encontramos na literatura somente algoritmos de codificação $O(n^2)$ para os códigos Hamming [10], mesmo sendo possível obter algoritmos de ordem $O(n)$. Por exemplo, os melhores algoritmos de decodificação para os códigos Reed-Solomon têm complexidade $O(n^2)$ e $O(n^3)$ [17], para os códigos Goppa, $O(n^3)$ [37]. O algoritmo básico de decodificação de um código linear, a decodificação por síndrome, tem complexidade $O(n\ 2^{n-k})$, onde k é a dimensão do código [3]. Assim, em certas circunstâncias, pode ser mais vantajoso abrir mão de uma maior detecção de erros em nome de uma decodificação mais rápida, principalmente em sistemas de transmissão digital em que a relação sinal-ruído é baixa.

Neste trabalho, caracterizaremos duas famílias de códigos Hamming encurtados, denotadas por $Gham(n)$ e $BP(n)$, onde n indica o comprimento das palavras dos códigos.

Este capítulo será dedicado a $Gham(n)$, uma família de códigos lineares binários com n bits para todo $n \in \mathbb{N}, n \geq 3$, com distância 3, que generaliza os códigos de Hamming. Essa generalização é baseada em recursão, o que permite um manuseio eficiente desses códigos. Tanto a codificação quanto a decodificação nessa família

tem complexidade $O(n)$. Em particular, para n da forma $2^r - 1$ temos exatamente os códigos Hamming binários. Este código utiliza $r = \lceil \log_2(n+1) \rceil$ bits de paridade, tendo dimensão $k = n - \lceil \log_2(n+1) \rceil$ e 2^k palavras.

4.1 Construção do código $Gham(n)$

Vamos construir um código de distância 3, que utiliza n bits, de maneira recursiva, a partir do código $Gham(3) = \{000, 111\}$. Observe que, para este código base, temos $n = 3, r = 2, k = 1$ e distância 3. Uma matriz geradora do mesmo é a matriz $G_{1\times 3} = [111]$.

Dado o código $Gham(n-1) = \{c_1^{n-1}, c_2^{n-1}, \cdots, c_M^{n-1}\}$ de comprimento $n-1$ com $M = 2^{n-1-\lceil \log_2(n) \rceil}$ palavras e distância Hamming 3, criamos o código $Gham(n) = \{c_1^n, c_2^n, \cdots, c_{M'}^n\}$ da seguinte forma:

1. Se $n \neq 2^m$, para $m \in \mathbb{N}$, então as palavras de $Gham(n)$ são dadas por:

$$c_i^n = \begin{cases} 0||c_i^{n-1}, \text{ para } 1 \leq i \leq M \\ 1||[c_{i-M}^{n-1} \oplus bin(n, n-1)], \text{ para } M+1 \leq i \leq 2M \end{cases},$$

   onde $bin(n, n-1)$ denota a representação binária do inteiro n utilizando $n-1$ bits e $a||c_i$ representa a concatenação do bit a com a palavra c_i.

2. Se $n = 2^m$, para $m \in \mathbb{N}$, então as palavras de $Gham(n)$ são dadas por:

$$c_i^n = c_{i_1}^{n-1} \cdots c_{i_k}^{n-1} || 0 || c_{i_{k+1}}^{n-1} \cdots c_{i_{n-1}}^{n-1}, \text{ para } 1 \leq i \leq M.$$

   onde $c_{i_j}^{n-1}$ representa o bit j da palavra c_i^{n-1}.

   Observe que, neste caso, tomamos o código $Gham(n-1)$ e acrescentamos 0 na posição $k+1$. Observe também que a segunda metade das palavras ainda pode ser escrita como $1||[c_i^{n-1} \oplus bin(n, n-1)]$, para $\frac{M}{2} \leq i \leq M$.

Dada uma palavra c_i^n, denotaremos por u_i^k a string formada pelos primeiros k bits de c_i^n e, por $p(u_i^k)$ a string formada pelos r bits finais.

A seguir exemplificamos a criação recursiva da família de códigos $Gham(n)$. Cada código é apresentado em duas colunas, com a coluna da esquerda correspondendo às palavras que começam com 0 e as da direita, com 1. Na Tabela 4.1 mostramos, além do código base, os códigos para n de 4 a 8. Os códigos para n de 5 a 7 correspondem à primeira forma de criação recursiva, com 3 bits de paridade para todos. Os códigos para $n = 4$ e $n = 8$ correspondem à segunda forma. Observe que o código para $n = 8$ passa a ter 4 bits de paridade.

Tabela 4.1: $Gham(n)$ para n de 3 a 8

n = 3		n = 4		n = 5	
000	111	0000	1011	00000	10101
				01011	11110
n = 6		**n = 7**		**n = 8**	
000000	100110	0000000	1000111	00000000	10000111
001011	101101	0001011	1001100	00010011	10010100
010101	110011	0010101	1010010	00100101	10100010
011110	111000	0011110	1011001	00110110	10110001
		0100110	1100001	01000110	11000001
		0101101	1101010	01010101	11010010
		0110011	1110100	01100011	11100100
		0111000	1111111	01110000	11110111

O Algoritmo 1 ilustra a criação do código $Gham(n)$. Nesse algoritmo é utilizada a função $str(c, f, s)$, que obtém o substring que vai das posições c até f do string s. A operação indicada por $||$ representa a concatenação de strings. Se as operações de obtenção de uma palavra puderem ser implementadas em tempo constante, então a complexidade do algoritmo é igual ao número de palavras do código, ou seja, $O(n\ 2^k)$. Observe que não há a necessidade de usar recursão para obter o código.

Algoritmo 1: Criação do código $Gham(n)$:

dados: inteiro: n
início
  $r \leftarrow \lceil \log_2(n+1) \rceil$; $k \leftarrow n - r$;
  $q \leftarrow 4$; $s \leftarrow 0$;
  para i **de** 3 **a** n **:**
    se $i = q$ **então** $q \leftarrow 2q$;
    senão $s \leftarrow s + 1$; $A[s] \leftarrow i$;
  $G[1] \leftarrow bin(n, 0)$;
  para i **de** 1 **a** k **:**
    para j **de** 1 **a** 2^{i-1} **:**
      $G[2^{i-1} + j] \leftarrow$
        $bin(k - i + 1, 1)||str(k - i + 2, k, G[i])||p(G[i]) \oplus bin(r, A[i])$
fim

4.2 Propriedades dos códigos $Gham(n)$

Nesta seção apresentaremos algumas propriedades dos códigos $Gham(n)$. Inicialmente, mostramos que o código $Gham(n)$ constitui-se num conjunto ordenado.

Propriedade 4.1 *O código Gham(n) gerado pelo Algoritmo 1 é ordenado e os strings u_i^k correspondem à representação binária, usando k bits, de todos os inteiros*

do intervalo 0 a 2^{k-1}.

Demonstração: Observe que todos os vetores u^{k+1} de F^{k+1} são gerados ora concatenando-se 0 no início de todos os vetores $u^k \in F^k$, ora concatenando-se 1. Isto é,

$$u^{k+1} = 0||u^k \text{ ou } u^{k+1} = 1||u^k, \text{ para algum } u^k \in F.$$

O código base, $Gham(3) = \{000, 111\}$, tem os strings $u_1^1 = 0$ e $u_2^1 = 1$ ordenados e correspondendo a todo o espaço F^2. Como os códigos são construídos de maneira recursiva, temos que as palavras formadas concatenando-se 0 no início já estarão ordenadas e, estas por sua vez, serão menores do que todas as palavras geradas concatenando-se 1, que por sua vez também já estão ordenadas. Quando temos n potência de 2, é acrescido 0 na posição $k+1$, isto é, nos bits de paridade, não alterando assim os bits de informação u_i^k.

□

Em seguida, vamos provar que os códigos $Gham(n)$ são códigos lineares com distância Hamming 3 e apresentaremos suas matrizes geradoras e de paridade.

Propriedade 4.2 *Os códigos* $Gham(n)$ *são códigos lineares, cujas matrizes geradoras podem ser escritas na forma* $G = [I_{k\times k}|A_{k\times n-k}]$*, onde* $I_{k\times k}$ *representa a matriz identidade de ordem* $k = n - \lceil \log_2(n+1) \rceil$ *e as linhas de* $A_{k\times n-k}$ *são formadas pelas representações binárias dos números de 3 a n que não são potências de 2, em ordem decrescente, usando r bits.*

Demonstração: Vamos fazer a prova por Indução Matemática.

Temos dois casos base, que são $n = 3$ e $n = 4$. É fácil ver que $Gham(3)$ é gerado por $G_3 = [111]$, $k = 3 - \lceil \log_2(3+1) \rceil = 1$ e a única linha de A é o número 3 representado em $3 - 1 = 2$ bits. Já para $Gham(4)$ temos a matriz $G_4 = [1011]$, com $k = 4 - \lceil \log_2(4+1) \rceil = 1$ e a única linha de A é o número 3 representado em $4 - 1 = 3$ bits.

Para o restante da prova consideraremos dois casos conforme n seja potência de 2 ou não. Lembre que todos os vetores u_i^{k+1} de F^{k+1} são da forma $0u_j^k$ ou $1u_j^k$, para $u_j^k \in F^k$. Suponha que $G_n = [I_{k\times k}|A_{k\times n-k}]$ é uma matriz geradora do código $Gham(n)$. Isto é, as palavras de $Gham(n)$ são formadas fazendo-se $c_n = u_i^k \cdot G_n$, para todo $u_i \in F^k$.

1. Se $n + 1 \neq 2^m$, para $m \in \mathbb{N}$, então,

Temos que a matriz G_{n+1} é da forma:

$$G_{n+1} = \left[\begin{array}{c|c} 1 & \text{bin(n+1,n)} \\ \hline 0 & \\ \vdots & G_n \\ 0 & \end{array}\right].$$

onde $bin(n+1, n)$ é escrita em n colunas de G_{n+1}.

(a) se $u_i^{k+1} = 0||u_i^k$, então temos que $(0||u_i^k) \cdot G_{n+1} = 0||c_i^n$, pois ao multiplicarmos $0||u_i^k$ pela primeira coluna de G_{n+1}, geramos o primeiro bit 0 no resultado final; e como o primeiro bit de $0||u_i^k$ é 0, multiplicar $0||u_i^k$ pelas n últimas colunas de G_{n+1} é o mesmo que fazer $u_i^k \cdot G_n = c_i^n$.

(b) se $u_i^{k+1} = 1||u_i^k$, então $(1||u_i^k) \cdot G_{n+1} = 1||[c_i^n \oplus bin(n+1, n)]$, pois ao multiplicarmos $1||u_i^k$ pela primeira coluna de G_{n+1}, geramos o primeiro bit 1 no resultado final; e como o primeiro bit de $1||u_i^k$ é 1, multiplicar $1||u_i^k$ pelas n últimas colunas de G_{n+1} é o mesmo que fazer $u_i^k \cdot G_n = c_i^n$ e somar em cada posição, os bits da primeira linha, ou seja, fazer $c_i^n \oplus bin(n+1, n)$.

Assim, todas as palavras de $Gham(n+1)$ são geradas.

2. Se $n + 1 = 2^m$, para $m \in \mathbb{N}$, então,

   Neste caso, a matriz geradora G_{n+1} possui uma coluna nula na posição $k + 1$.

$$G_{n+1} = \left[\begin{array}{c|c|c} & 0 & \\ I_k & \vdots & A_{k\times n-k} \\ & 0 & \end{array}\right].$$

   Os vetores de informação pertencem a F^k. Logo, temos que $u_i^k \cdot G_{n+1} = c_{i_1}^n \cdots c_{i_k}^n 0 c_{i_{k+1}}^n \cdots c_{i_n}^n = c_i^{n+1}$ (é acrescentado um bit 0 na posição $k + 1$ em todas as palavras c_i^n. Isto é, todas as palavras de $Gham(n + 1)$ são geradas.

□

Na Tabela 4.2 apresentamos a parte não trivial das matrizes geradoras dos códigos $Gham(n)$ para n de 3 a 11.

A seguir apresentamos um algoritmo que encontra uma codificação qualquer de um código $Gham(n)$.

Codificação

Seja uma palavra c_i^n, com $0 \leq i \leq 2^k - 1$, do código $Gham(n)$. Note que pelo processo de construção dos códigos $Gham(n)$ se uma palavra começa com bit 1,

Tabela 4.2: $A_{k\times n-k}$ dos códigos $Gham(n)$

n	3	4	5	6	7	8	9	10	11
	11	011	101	110	111	0111	1001	1010	1011
			011	101	110	0110	0111	1001	1010
				011	101	0101	0110	0111	1001
$A_{k\times n-k}$					011	0011	0101	0110	0111
							0011	0101	0110
								0011	0101
									0011

então ela foi gerada fazendo-se um ou exclusivo com $bin(n, r)$, ou $bin(n-1, r)$ se n é potência de 2. Cada c_i^n, para $1 \leq i \leq 2^k$, tem os k primeiros bits correspondendo aos números binários $0 \leq i \leq 2^k - 1$ ordenadamente, isto é, $u_i^k = u_{i_k} u_{i_{k-1}} \cdots u_{i_1} = bin(i-1, k)$. Para calcular $p(u_i^k)$, fazemos a operação de ou exclusivo de $bin(0, r)$ com $A(j)$ para todo j tal que $u_{i_j} = 1$, $1 \leq j \leq k$, onde $A(j)$ representa o j-ésimo número que não é potência de 2, como mostra o Algoritmo 2.

Vamos exemplificar o processo de encontrar a palavra c_i^n, com $0 \leq i \leq 2^k - 1$, de um código $Gham(n)$.

Exemplo 4.3 *Para encontrar c_{14}^8, tomamos $u_{13}^4 = 1101$. Os bits da direita para a esquerda que são iguais a 1 são o $1^o, 3^o$ e 4^o, logo, devemos fazer a operação de ou exclusivo de 0000 com 3, 6 e 7 em binário utilizando 4 bits. Isto é, fazemos*

$$\begin{array}{rll} & 0000 & \\ \oplus & 0011 & \leftarrow A[1] = 3 \\ \hline & 0011 & \\ \oplus & 0110 & \leftarrow A[3] = 6 \\ \hline & 0101 & \\ \oplus & 0111 & \leftarrow A[4] = 7 \\ \hline & 0010 & \end{array}$$

Temos então $p(u_{13}^4) = 0010$ e portanto, $c_{14}^8 = 11010010$.

Teorema 4.4 *A complexidade do Algoritmo 2 é $O(n)$.*

Considerando o vetor A como um inteiro, temos que sua geração é feita em $O(n)$. Com a tecnologia atual, para que A seja representado como um inteiro, devemos ter $n \leq 2^{64}$, o que é bastante plausível. O comando **para** é executado $k = n - \lceil \log_2(n+1) \rceil$ vezes. Assim, a complexidade do Algoritmo 2 é $O(n)$.

Propriedade 4.5 *A distância Hamming de $Gham(n)$ é 3.*

Algoritmo 2: Obtenção de c_i^n:

dados: inteiros: i, k, n
início
$A \leftarrow [3, 5, 6, 7, \ldots, n]$;
$u_i \leftarrow bin(i-1, k)$;
$p(u_i) \leftarrow \mathbf{0}$;
para j **de** $k, k-1, \cdots, 1$ **:**
se $u_{i_j} = 1$ **então** $p(u_i) \leftarrow p(u_i) \oplus A[k-j+1]$;
$c_i^n \leftarrow u_i p(u_i)$;
retorna c_i^n;
fim

Demonstração: Pelo Teorema 2.1, basta provarmos que $w(Gham(n)) = 3$.

Dada uma palavra $c_i^n \in Gham(n)$, considere $w(u_i^k) = d$.

1. Se $d = 1$, então $p(u_i^k)$ foi gerado fazendo-se $p(u_i^k) = \mathbf{0} \oplus bin(n_1, r)$ para algum n_1 tal que $w(n_1) \geq 2$, pois n_1 não é uma potência de 2. Logo, $w(p(u_i^k)) \geq 2$, de onde concluímos que $w(c_i^n) \geq 3$.

2. Se $d = 2$, então $p(u_i^k)$ foi gerado fazendo-se $p(u_i^k) = \mathbf{0} \oplus bin(n_1, r) \oplus bin(n_2, r)$, para algum $n_1 \neq n_2$, ambos não potência de 2.

   Como $n_1 \neq n_2$, temos que $bin(n_1, r) \oplus bin(n_2, r) \neq \mathbf{0}$. Logo, $w(p(u_i^k)) \geq 1$, de onde concluímos que $w(c_i^n) \geq 3$.

3. Se $d \geq 3$, então $w(c_i^n) \geq 3$.

□

Na sequência, vamos discutir a decodificação com o código $Gham(n)$.

Decodificação

Vamos apresentar o algoritmo de decodificação do código $Gham(n)$ e sua prova de corretude.

Teorema 4.6 *Considere que um vetor y de comprimento n é recebido. Vamos denotar os $k = n - \lceil \log_2(n+1) \rceil$ bits inicias de y por u e os $r = \lceil \log_2(n+1) \rceil$ bits finais por $p'(u)$. O processo de decodificação consiste em calcular $p(u)$ e fazer a operação $p(u) \oplus p'(u)$. Temos que:*

1. *$p(u) \oplus p'(u) = \mathbf{0}$ se, e somente se, não ocorreu erro em y;*

2. *$p(u) \oplus p'(u)$ tem apenas um bit 1, digamos na posição l se, e somente se, ocorreu um erro na posição l de $p'(u)$;*

Algoritmo 3: Decodificação de $y = u||p'(u)$:

dados: inteiros: y, n
início
  $k \leftarrow n - \lceil \log_2(n+1) \rceil$;
  $p(u) \leftarrow$ `CalculaParidade(`y`)`;
  $t \leftarrow p(u) \oplus p'(u)$;
  se $w(t) \leq 1$ **então**
    retorna u;
  senão se $t = A[l]$ *para algum* $1 \leq l \leq k$ **então**
    $u' \leftarrow u \oplus bin(2^{l-1}, k)$;
    retorna u';
  senão
    retorna erro;
fim

3. *$p(u) \oplus p'(u)$ é o l-ésimo número em binário que não é potência de 2, com $1 \leq l \leq k$ se, e somente se, ocorreu um erro na posição l (da direita para a esquerda) de u;*

4. *Se $p(u) \oplus p'(u)$ não satisfaz a nenhuma condição anterior, então ocorreu mais de um erro e y não é decodificado.*

Demonstração:

1. $p(u) \oplus p'(u) = \mathbf{0} \Leftrightarrow p(u) = p'(u) \Leftrightarrow y = u||p(u) \in Gham(n)$.

   Note que, para todo $u \in F^k$, o vetor $u||p(u)$ de comprimento n é uma palavra do código $Gham(n)$.

2. Se $p(u) \oplus p'(u)$ tem apenas um bit 1, digamos na posição l, então $d(y, u||p(u)) = 1$. Note que, o fato de um código C ter distância Hamming 3, garante que um vetor qualquer de F^k tem distância Hamming 1 para, no máximo, uma palavra de C. Neste caso, y é decodificado como $u||p(u)$, isto é, ocorreu um erro na posição l de $p'(u)$.

   A volta é imediata.

3. Considere o vetor u' obtido a partir de u trocando-se apenas o bit na posição l, para algum $1 \leq l \leq k$. Temos que $p(u') = p(u) \oplus A[l]$ (ou $p(u) \oplus p(u') = A[l]$).

   Se $p(u) \oplus p'(u) = A[l]$, então $p(u') = p'(u)$. Portanto, $d(y, u'||p(u')) = 1$. E o vetor y é decodificado como $u'||p(u')$, tendo ocorrido então um erro na posição l de u.

   Por outro lado, se ocorreu um erro na posição l de u, então $p(u) = p(u') \oplus A[l]$. Como não ocorreu erro na parte de paridade, temos que $p(u') = p'(u)$. Logo, concluímos que $p(u) \oplus p'(u) = A[l]$.

4. Se $p(u) \oplus p'(u)$ não satisfaz nenhuma das condições anteriores, considerando as demonstrações dos 3 itens, podemos concluir que houve mais de 1 erro o que impossibilita a correção.

□

Teorema 4.7 *A complexidade de decodificação do $Gham(n)$ é $O(n)$.*

O método de decodificação calcula $p(u)$ como no Algoritmo 2. Além disso, executa uma operação extra de ou exclusivo com $p(u')$. Assim, sua complexidade é $O(n)$.

Exemplo 4.8 *Considere o código $Gham(5) = \{00000, 01011, 10101, 11110\}$. Temos que as bolas $B[c_i^5, 1]$ de raio 1 centradas nas palavras c_i^5, para i de 1 a 4, são dadas na Tabela 4.3.*

Tabela 4.3: Bolas do código $Gham(5)$

00000	**01011**	**10101**	**11110**
00001	01010	10010	11111
00010	01001	10001	11100
00100	01111	10111	11010
01000	00011	11011	10110
10000	11011	00011	01110

Se o vetor 00001 *é recebido. Temos* $u = 00$ *e* $p'(00) = 001$. *Calculamos a paridade de* u, $p(00) = 000$. *Em seguida fazemos* $p(00) \oplus p'(00) = 001$, *que tem peso 1. Logo, concluímos que houve um erro na posição 1 de* $p'(00)$. *E portanto,* 00001 *é corrigido para* 00000 *e decodificado como* 00.

Se o vetor 11011 *é recebido. Temos* $u = 11$ *e* $p'(11) = 011$. *Calculamos a paridade de* u, $p(11) = A[1] \oplus A[2] = 011 \oplus 101 = 110$. *Em seguida fazemos* $p(11) \oplus p'(11) = 101 = bin(5) = A[2]$ *(segundo número que não é potência de 2). Logo, concluímos que houve um erro na posição 2 da direita para a esquerda de* $u = 11$. *E portanto,* 11011 *é corrigido para* 01011 *e decodificado como* 01.

Note que, os vetores $\{00110, 00111, 01100, 01101, 10010, 10011, 11000, 11001\}$ *não estão à distância 1 de nenhuma palavra do código. Portanto, se um desses vetores é recebido, este não é decodificado.*

A dimensão dos códigos lineares ótimos de distância Hamming 3 já é conhecida, porém, apresentamos no Teorema 4.12 uma prova alternativa para este resultado. A seguir, apresentamos alguns lemas para a prova deste Teorema.

Lema 4.9 *A matriz de paridade H do código $Gham(n)$ é formada pelas representações binárias dos números de 1 a n escritos em coluna utilizando $n-k$ bits.*

Demonstração: A matriz geradora do código $Gham(n)$ é dada por $G = [I_k|A_{k\times n-k}]$, onde I_k representa a matriz identidade de ordem $k = n - \lceil \log_2(n+1) \rceil$ e as linhas de $A_{n-k\times k}$ são formadas pelas representações binárias dos números de 3 a n que não são potências de 2, em ordem decrescente, usando $n-k$ bits.

Temos então que a matriz de paridade H do código $Gham(n)$ é dada por $H = [A^T_{n-k\times k}|I_{n-k}]$, onde as colunas de $A^T_{n-k\times k}$ são formadas pelas representações binárias dos números de 3 a n que não são potências de 2, em ordem decrescente, usando $n-k$ bits. E note que as colunas de I_{n-k} são formadas pelas representações binárias dos números de 1 a n que são potências de 2, em ordem decrescente, usando $n-k$ bits. □

Teorema 4.10 *Os códigos de Hamming $Ham(r)$ são um caso particular dos códigos $Gham(n)$.*

Demonstração: Temos que o código $Ham(r)$ tem matriz de paridade H de ordem $r \times (2^r - 1)$ cujas colunas são formadas pelos vetores não nulos de F^r que são exatamente as representações binárias dos números de 1 a n utilizando $r = n - k$ bits. Assim, a matriz de paridade do código $Ham(r)$ é equivalente a matriz de paridade do código $Gham(n)$, onde $n = 2^r - 1$. E portanto, $Ham(r)$ é equivalente a $Gham(n)$.

□

A seguir apresentamos dois lemas que serão utilizados para demonstrar que os códigos $Gham(n)$ são códigos lineares ótimos. Onde utilizamos o termo ótimo para um código linear com o maior número de palavras possível de comprimento n.

Lema 4.11 *Seja $2^l \leq n < 2^{l+1}$. Temos que a representação binária de n utiliza $l+1$ dígitos.*

Demonstração: Vamos considerar primeiramente o caso em que $n = 2^l$ e utilizar o Princípio de Indução de Matemática em l.

Para $l = 0$, temos que $2^0 = 1$, utiliza 1 dígito em sua representação binária.

Agora, suponha que a representação binária 2^l utiliza $l+1$ dígitos. Temos que $2^{l+1} = 2 \cdot 2^l$. Digamos que a forma binária de 2^l é dada por $d_1 d_2 \cdots d_{l+1}$. Ao multiplicamos este número por 2 cuja forma binária é 10, pela definição de multiplicação binária, temos

$$\begin{array}{rr} & d_1 d_2 \cdots\ d_{l+1} \\ \times & 1\ 0 \\ \hline & 0 \cdots\ 0\ 0 \\ + & d_1 d_2 \cdots d_{l+1} \\ \hline & d_1 d_2 \cdots d_{l+1}\ 0 \end{array}$$

Logo, o resultado é representado com $l+2$ dígitos.

Consideremos agora o caso onde $2^l < n < 2^{l+1}$. Podemos escrever $n = 2^l + r$, onde $0 < r < 2^l$. Pelo caso anterior, sabemos que 2^l utiliza $l+1$ dígitos se escrito na base 2. Como $r < 2^l$, sabemos que r utiliza no máximo $l+1$ dígitos se escrito na base 2. Pela propriedade de soma de números binários, temos que $n = 2^l + r$ utiliza $l+1$ dígitos se escrito na base 2. □

Note que se, $n = 2^l + r$, onde $0 \leq r < 2^l$, então $\lceil \log_2(n+1) \rceil = l+1$.

Teorema 4.12 *Um código linear ótimo de comprimento n e distância Hamming 3 tem dimensão* $k = n - \lceil \log_2(n+1) \rceil$.

Demonstração: Dado $n \geq 3$, considere C o código linear ótimo com M palavras de comprimento n cuja distância Hamming é 3. Seja $G = [I_k | A_{k \times n-k}]$ a matriz geradora de C na forma padrão. Podemos escrever $n = k + r$, onde k é dimensão de C. Temos que $M = 2^k$ é máximo, portanto, k deve ser máximo. Suponha, por absurdo, que $k \geq n - \lceil \log_2(n+1) \rceil + 1$. Temos que as linhas da matriz $A_{k \times n-k}$ têm que corresponder às representações binárias de números distintos, não nulos e que não são potências de 2 (caso contrário, G teria linhas com peso 2). Lembramos que existem exatamente $k = n - \lceil \log_2(n+1) \rceil$ números satisfazendo estas condições. Temos então, que uma linha de $A_{k \times n-k}$ será formada pela representação binária no número $n+1$, se $n+1$ não é potência de 2, ou $n+2$, caso contrário. Note que, sob essas condições, a representação binária de $n+1$ é escrita com $\lceil \log_2(n+1) \rceil$ dígitos e a de $n+2$ com $\lceil \log_2(n+1) \rceil + 1$. Como $k \geq n - \lceil \log_2(n+1) \rceil + 1$, temos que $n - k \leq \lceil \log_2(n+1) \rceil - 1$, isto é, $n+1$ ou $n+2$ é escrito com no máximo $\lceil \log_2(n+1) \rceil - 1$ dígitos, absurdo.

Como existe um código linear de dimensão $k = n - \lceil \log_2(n+1) \rceil$, temos que este é o k máximo e portanto a dimensão do código linear ótimo de comprimento n. □

Corolário 4.13 *Os códigos Gham(n) são códigos lineares ótimos.*

4.3 Obtenção gulosa de $Gham(n)$

Mostraremos a seguir, o Algoritmo 6 que é um algoritmo alternativo para obtenção do código $Gham(n)$. Nesse algoritmo usamos a função $Dist(str1, str2)$, que obtém a distância Hamming entre os strings $str1$ e $str2$.

Algoritmo 4: Criação gulosa do código $Gham(n)$:

dados: inteiro: n
início
  $r \leftarrow \lceil \log_2(n+1) \rceil$; $k \leftarrow n - r$;
  $G[1] \leftarrow bin(n, 0)$;
  para i *de* 1 *a* 2^{k-1} **:**
    $b \leftarrow bin(k, i)$;
    para j *de* 0 *a* 2^{r-1} **:**
      $x \leftarrow 1$;
      para t *de* 1 *a* $i-1$ **:**
        se $Dist(G[t], b.bin(r, j)) < 3$ **então**
          $x \leftarrow 0$; parar;
      se *(x=1)* **então**
        parar;
    $G[i] \leftarrow b||bin(r, j)$;
fim

A ideia do Algoritmo 6 é inicialmente colocar $0 \in F^k$ no código e, em seguida, para cada $u \in (F^k \setminus \{0 \in F^k\})$, concatenar a menor paridade $p \in F^r$ possível, tal que a palavra resultante tenha distância 3 para as palavras anteriores já definidas. O algoritmo obedece à Propriedade 4.14, ainda a ser provada, para todos os códigos conhecidos.

Propriedade 4.14 *O Algoritmo 6 obtém o código* $Gham(n)$.

O interesse no Algoritmo 6 surgiu do fato de que algumas experimentações sugerem uma conexão entre códigos $Gham(n)$, que são códigos lineares, com alguns códigos não lineares, um tema ainda pouco explorado na literatura. Com pequenas alterações no algoritmo, conseguimos obter os códigos ótimos para n de 8 a 11, que, como pode ser visto na Tabela 1.1, são códigos não lineares. Devem ser feitas duas pequenas mudanças no algoritmo. A primeira é que ao invés de se concatenar sempre uma única paridade $v \in F^r$ a cada elemento $u \in F^k \setminus \mathbf{0}$ pode-se ter nenhuma ou mais de uma concatenação, desde que o resultado seja coerente com as escolhas já feitas. A segunda alteração é acrescentar ao código, no início de tudo, além de $\mathbf{0} \in \mathbf{F^k}$ algumas palavras $v \in F^n$. Para os valores de n de 8 a 11, verificamos, experimentalmente, que é necessário o acréscimo inicial de 4 palavras, além de $\mathbf{0}$, para obter-se o código desejado. Essas palavras estão listadas na Tabela 4.4.

Um desafio interessante, que enfrentaremos na sequência do trabalho, é o de generalizar essa construção para os demais códigos não lineares e obter todos esses os códigos não lineares ótimos.

Tabela 4.4: Palavras fixadas para a obtenção dos códigos não lineares.

n	8	9	10	11
	00000000	000000000	0000000000	00000000000
	00011111	000011111	0000011111	00000011111
	00100101	000100101	0000100101	00000100101
	00101010	001000111	0001000111	00000101011
	01001101	011000001	0011000001	01010000010

Capítulo 5

Códigos Binário Posicional

Neste capítulo mostramos uma segunda maneira de construir códigos lineares binários com n bits para todo $n \in \mathbb{N},\ n \geq 3$ e distância Hamming 3. Denominamos Códigos Binário Posicional e para cada n denotamos por $BP(n)$.

5.1 Construção recursiva do código

Vamos construir novamente um código de distância 3, que utiliza n bits, de maneira recursiva, a partir do código $BP(3) = \{000, 111\}$. A matriz geradora do mesmo é a matriz $G_{1\times 3} = [111]$. Antes de apresentar a construção dos códigos, apresentamos a definição a seguir.

Definição 5.1 *Definimos o binário posicional de um número n como o número $b(n)$ que contém 1 (ou 0) na posição 2^i se, e somente se, $bin(n)$ contém 1 (ou 0) na posição i.*

Exemplo 5.2 $b(6) = 1010$ *tem 1 nas posições* $2 = 2^1$ *e* $4 = 2^2$*, pois* $bin(6) = 110$ *tem 1 nas posições 1 e 2.*

Observação: $b(n)$ inicia na posição 1.

Dado o código $BP(n-1) = \{c_1^{n-1}, c_2^{n-1}, \cdots, c_M^{n-1}\}$ de comprimento $n-1$ com $M = 2^{n-1-\lceil \log_2(n) \rceil}$ palavras e distância Hamming 3, criamos o código $BP(n) = \{c_1^n, c_2^n, \cdots, c_{M'}^n\}$ da seguinte forma:

1. Se $n \neq 2^m$, para $m \in \mathbb{N}$, então as palavras de $BP(n)$ são dadas por:

$$c_i^n = \begin{cases} 0||c_i^{n-1}, \text{ para } 1 \leq i \leq M \\ c_{i-M}^{n-1} \oplus p(n), \text{ para } M+1 \leq i \leq 2M, \end{cases}$$

onde $p(n) = bin(2^{n-1}) \oplus b(n)$.

Exemplo 5.3 $p(6) = 2^6 \oplus b(6) = 101010$.

A Tabela 5.1 apresenta as palavras $p(n)$ para $3 \leq n \leq 27$, onde n não é potência de 2.

Tabela 5.1: $p(n)$ para n de 3 a 27

n	$p(n)$	n	$p(n)$
3	111	17	11000000000000001
5	11001	18	101000000000000010
6	101010	19	1001000000000000011
7	1001011	20	10001000000000001000
9	110000001	21	10000100000000001001
10	1010000010	22	10000010000000001010
11	10010000011	23	10000001000000001011
12	100010001000	24	10000000100000010000000
13	1000010001001	25	10000000010000010000001
14	10000010001010	26	10000000001000010000010
15	100000010001011	27	10000000000100010000011

2. Se $n = 2^m$, para $m \in \mathbb{N}$, então as palavras de $BP(n)$ são dadas por:

$$c_i^n = 0||c_i^{n-1}, \text{ para } 1 \leq i \leq M.$$

A seguir exemplificamos a criação recursiva dos códigos $BP(n)$. Na Tabela 5.2 mostramos, além do código base, os códigos para n de 4 a 8. Os códigos para n de 5 a 7 correspondem à primeira forma de criação recursiva. Os códigos para $n = 4$ e $n = 8$ correspondem à segunda forma.

Tabela 5.2: $BP(n)$ para n de 3 a 8

n = 3		n = 4		n = 5	
000	111	0000	0111	00000	11001
				00111	11110
n = 6		**n = 7**		**n = 8**	
000000	101010	0000000	1001011	00000000	01001011
000111	101101	0000111	1001100	00000111	01001100
011001	110011	0011001	1010010	00011001	01010010
011110	110100	0011110	1010101	00011110	01010101
		0101010	1100001	00101010	01100001
		0101101	1100110	00101101	01100110
		0110011	1111000	00110011	01111000
		0110100	1111111	00110100	01111111

Observe que, pela formação do código, se tomamos os bits nas posições (de 1 a n da direita para a esquerda) que não são potência de 2 da i-ésima palavra do

código $BP(n)$, temos exatamente $bin(i-1)$. Por exemplo, em $c_6^6 = \underline{101}\underline{1}01$, temos $bin(5) = 101$.

5.2 Propriedades dos códigos $BP(n)$

Nesta seção apresentaremos algumas propriedades dos códigos $BP(n)$. Vamos mostrar que os códigos $BP(n)$ são códigos lineares com distância Hamming 3 e apresentaremos suas matrizes geradoras e de paridade.

Propriedade 5.4 *Os códigos $BP(n)$ são códigos lineares.*

Demonstração: Para mostrarmos que o código binário é linear, basta mostrarmos que a soma de duas palavras do código é também uma palavra do código. Faremos isto utilizando o Princípio de Indução Matemática em n.

É fácil ver que o código $BP(3) = \{000, 111\}$ é um código linear. Considere que o código $BP(n-1)$ é linear.

Se $n = 2^k$, para algum k inteiro, a prova é trivial. Para $n \neq 2^k$, temos:

1. $[0||c_i^{n-1}] \oplus [0||c_j^{n-1}] \in BP(n)$, pois, por hipótese de Indução, $c_i^{n-1} \oplus c_j^{n-1} \in BP(n-1)$.

2. $$\begin{aligned} [0||c_i^{n-1}] \oplus [c_j^{n-1} \oplus p(n)] &= [0||c_i^{n-1} \oplus c_j^{n-1}] \oplus p(n) \\ &= [0||c_k^{n-1}] \oplus p(n), \text{ para algum } 1 \leq k \leq n-1 \\ &= c_k^{n-1} \oplus p(n) \in BP(n). \end{aligned}$$

3. $[c_i^{n-1} \oplus p(n)] \oplus [c_j^{n-1} \oplus p(n)] = c_i^{n-1} \oplus c_j^{n-1} \in BP(n)$.

□

Propriedade 5.5 *As palavras $p(i)$, para todo i não potência de 2, no intervalo $3 \leq i \leq n$ formam uma base para o código $BP(n)$.*

Demonstração: Podemos ver que os vetores $p(i)$ são linearmente independentes, pois existe um bit 1 na posição i, não potência de 2, no intervalo $3 \leq i \leq n$ apenas no vetor $p(i)$. Ainda, temos $n - \lceil \log_2(n+1) \rceil$ números não potência de 2, no intervalo $3 \leq i \leq n$, portanto, estes vetores formam uma base para $BP(n)$.

□

Codificação e Decodificação

Observe que todas as palavras do código $BP(n)$ são obtidas através da operação de ou exclusivo de palavras $p(j)$, para alguns valores de j. Utiliza-se $p(j)$ em todas as palavras da segunda metade do código $BP(i)$. Antes de discutir os algoritmos de codificação e decodificação do código $BP(n)$ vamos apresentar o Algoritmo 5 para a geração dos vetores p e b. O vetor p guarda os valores $p(j)$ e o vetor b, os valores $b(j)$, para $1 \leq j \leq n$. Será mostrado que o algoritmo tem complexidade $O(n)$.

Algoritmo 5: Geração dos vetores p e b:

```
dados: inteiro: n
Gera (t, m, sp, s)
início
    para q de t a m :
        sa ← s + p2[p2[m − q] − 1];
        spa ← sp + p2[m − q];
        se q < m então
            Gera (q + 1, m, spa, sa);
        p[np] ← sa + p2[spa − 1];
        b[np] ← sa;
        np ← np − 1;
fim
l ← ⌈log2 n⌉;
np ← 2^l − 1;
Gera (1, l, 0, 0);
```

O Algoritmo 5 é uma variação do algoritmo clássico de *backtracking* para gerar todos os subconjuntos de um conjunto S. Na presente versão, o conjunto S corresponde às $\lceil \log_2 n \rceil$ potências de 2 menores ou iguais a n e cada subconjunto gerado forma um dos números binários posicionais e também um valor $p(j)$, necessários aos processos de codificação e decodificação. O algoritmo usa uma variável global np e vetores globais $p2$ (potências de 2), p e b. O número de subconjuntos gerados é $2^{\lceil \log_2 n \rceil} - 1$, uma vez que o subconjunto vazio não é considerado. A variável np é utilizada como um índice para o armazenamento dos subconjuntos gerados e é inicializada apontando para as posições finais dos vetores p e b. O algoritmo começa com um conjunto vazio. A cada chamada do procedimento *Gera* uma nova potência de dois é adicionada ao subconjunto anterior, formando um novo subconjunto. Observe que essas potências são tomadas em ordem decrescente e que os vetores p e b são preenchidos apenas após a chamada recursiva do algoritmo. Isso garante que os números binários posicionais sejam criados em ordem decrescente. O primeiro valor armazenado corresponde ao conjunto contendo todas as potências de dois e seu armazenamento é na última posição de b, uma vez que ele é o maior número considerado. O último número salvo corresponde ao número 1 e ele é colocado na primeira posição de b. O vetor p é criado através de um processo simultâneo análogo. A variável $m = \lceil \log_2 n \rceil$ não se modifica ao longo da execução e as variáveis

s, sa, sp, spa guardam as somas de potências de dois.

O teorema seguinte dá a complexidade do algoritmo descrito.

Teorema 5.6 *A complexidade do Algoritmo 5 é $O(n)$.*

Demonstração: Cada vez que o comando **para** é executado um subconjunto diferente de potências de dois é criado. O número total de execuções desse *loop* é, portanto, $np = 2^{\lceil \log_2 n \rceil} - 1 < 2^{\log_2 n + 1} = 2n$. As demais instruções são todas executadas em $O(1)$. Então, a complexidade final é $O(n)$. □

A seguir, consideramos a codificação da i-ésima palavra do código $BP(n)$. O Algoritmo 6 obtém essa i-ésima palavra.

Algoritmo 6: Obtenção de c_i^n:

dados: inteiros: i, n
início
  $k \leftarrow n - \lceil \log_2(n+1) \rceil$; $A \leftarrow [3, 5, 6, 7, \cdots, n]$;
  `Gera` $p(j), b(j),\ j \in \{1, 2, 3, \cdots, n\}$;
  $u \leftarrow bin(i-1, k) = u_k u_{k-1} \cdots u_1$; $c_i^n \leftarrow \mathbf{0}$;
  para j **de** *1* **a** k **:**
    se $u_j = 1$ **então**
      $c_i^n \leftarrow c_i^n \oplus p(A[j])$
  retorna c_i^n;
fim

Teorema 5.7 *A complexidade do Algoritmo 6 é $O(n)$.*

Os parâmetro $p(i)$ são gerados em tempo $O(n)$. O comando **para** é executado $k = n - \lceil \log_2(n+1) \rceil$ vezes. Assim, a complexidade do algoritmo é $O(n)$.

Exemplo 5.8 *Vamos obter a palavra c_{12}^7.*

$$u \leftarrow bin(11) = 1011 \quad \Rightarrow$$

$$\begin{array}{rr} & 0000000 \\ \oplus & 0000111 \\ \hline & 0000111 \end{array} \leftarrow p(A[1]) = p(3)$$

$$\begin{array}{rr} & 0000111 \\ \oplus & 0011001 \\ \hline & 0011110 \end{array} \leftarrow p(A[2]) = p(5)$$

$$\begin{array}{rr} & 0011110 \\ \oplus & 1001011 \\ \hline & \underbrace{1010101} \\ & c_{12}^7 \end{array} \leftarrow p(A[4]) = p(7)$$

Algoritmo 7: Decodificação de y:

dados: inteiro: n**; vetor** y
início
  Gera $p(j),\ b(j),\ j \in \{1, 2, 3, \cdots, n\}$;
  $v \leftarrow y = y_n y_{n-1} \cdots y_1$; $q \leftarrow 4$;
  para i **de** *3* **a** n **:**
    se $i \neq q$ **então**
      se $y[i] = 1$ **então**
        $v \leftarrow v \oplus p(i)$
    senão $q \leftarrow 2q$;
  se $v = 0$ **ou** $v = b(l)$, *para algum* $1 \leq l \leq n$ **então**
    se $v \neq 0$ **então**
      $y \leftarrow y \oplus bin(2^{l-1})$;
    $q \leftarrow 4$; $j \leftarrow 0$;
    para i **de** *3* **a** n **:**
      se $i \neq q$ **então**
        $j \leftarrow j + 1$; $u_j \leftarrow y_i$;
      senão $q \leftarrow 2q$;
    retorna u;
  senão
    retorna erro;
fim

Apresentamos o algoritmo de decodificação do código e sua prova de corretude.

Teorema 5.9 *Considere que um vetor y de comprimento n é recebido. O processo de decodificação consiste em fazer, sucessivamente, a operação ou exclusivo de $y = y_n y_{n-1} \cdots y_1$ com $p(i)$ se $y_i = 1$, para todo i não potência de 2. O resultado da operação é:*

1. **0** *se, e somente se, não ocorreu erro em y;*

2. *$b(l)$, para algum $1 \leq l \leq n$, se, e somente se, ocorreu um erro na posição l (da direita para a esquerda) de y;*

3. *Diferente de $b(l)$, então ocorreu mais de um erro e y não é decodificado.*

Após detectar e corrigir o erro de y, se for o caso, o vetor é decodificado tomando os bits nas posições de y que não são potência de 2.

Demonstração: Considere que y difere de alguma palavra $c \in BP(n)$ na posição l, para algum $1 \leq l \leq n$. Podemos escrever, $y = c \oplus bin(2^{l-1})$.

Observe que a única palavra da base de $BP(n)$ que possui 1 na posição i, para i não potência de 2, é a palavra $p(i)$.

1. Se l é potência de 2, então as palavras $p(i)$ que são somadas no processo de decodificação são exatamente as mesmas que formam c, pois y tem 1 nas mesmas posições não potências de 2 que c. Assim, no processo de decodificação, é somada a palavra c ao vetor y. Temos que

$$y \oplus c = bin(2^{l-1}) = b(l).$$

2. Se l não é potência de 2, então vamos analisar dois casos.

   (a) Se c tem 0 na posição l, então y tem 1 na posição l. No processo de decodificação, são somadas as mesmas palavras $p(i)$ que formam c, e além delas, é somada a palavra $p(l)$.

   (b) Se c tem 1 na posição l, então y tem 0 na posição l. No processo de decodificação, são somadas as mesmas palavras $p(i)$ que formam u, exceto a palavra $p(l)$. Note que isto é equivalente a fazer a operação $y \oplus c \oplus p(l)$.

   Sendo assim, nos dois casos, fazemos:

$$y \oplus c \oplus p(l) = bin(2^{l-1}) \oplus p(l) = b(l).$$

□

Teorema 5.10 *A complexidade do Algoritmo 7 é $O(n)$.*

Os parâmetros $p(j)$ e $b(j)$ são criados em tempo $O(n)$. Os comandos **para** são executados $n-3$ vezes, cada. Cada operação leva tempo $O(1)$. Assim, o Algoritmo 7 tem complexidade $O(n)$.

Exemplo 5.11 *Considere que o vetor* 1011100 *é recebido. Fazemos*

$$\begin{array}{rll} & 1011100 & \\ \oplus & 1001011 & \leftarrow p(7) \\ \hline & 0010111 & \\ \oplus & 0011001 & \leftarrow p(5) \\ \hline & 0001110 & \\ \oplus & 0000111 & \leftarrow p(3) \\ \hline & 0001001 & \rightarrow \textit{erro no bit } 4+1=5 \end{array}$$

Logo, a palavra enviada foi na verdade 1001100*. A mensagem é então decodificada tomando-se os bits que estão nas posições que não são potência de 2:* 1001*, a* 10^a *palavra do código* $BP(7)$*.*

Exemplo 5.12 *Considere que o vetor* 10101111 *é recebido. Fazemos*

$$\begin{array}{rll} & 10101111 & \\ \oplus & 00000111 & \leftarrow p(3) \\ \hline & 10101000 & \\ \oplus & 00101010 & \leftarrow p(6) \\ \hline & 10000010 & \rightarrow \textit{ocorreu mais de 1 erro} \end{array}$$

O resultado 10000010 *corresponde a* $b(8+2) = b(10)$*, que está fora do intervalo de 1 a 8. Portanto o erro não pode ser corrigido.*

Os algoritmos apresentados até este ponto baseiam-se no fato de que certos valores possam ser representados por inteiros e que a operação básica utilizada, de ou exclusivo, possa ser realizada em $O(1)$. Essa restrição não tem relevância no caso do código $Gham(n)$. Nesse código, os inteiros envolvidos são i, o valor a ser codificado, n, o número de bits e o vetor $A[k]$, com $k = n - \lceil \log_2(n+1) \rceil$, contendo inteiros menores ou iguais a n. As operações de ou exclusivo são aplicadas com o vetor A que, considerando a tecnologia atual, pode representar inteiros $n \leq 2^{64}$, o que é adequado a qualquer aplicação prática. Observe que o valor i a ser codificado pode ser representado por um inteiro ou por um vetor de bits, sem afetar a complexidade do algoritmo descrito.

Já no caso do código $BP(n)$, pode ser inviável a representação, como inteiros, dos valores $p(j)$ e $b(j)$, $1 \leq j \leq n$, para $n > 64$, considerando a tecnologia atual. Vamos mostrar algoritmos alternativos, que não geram esses valores, mas simulam o efeito de sua utilização no vetor que está sendo codificado ou decodificado. O Algoritmo 8 mostra a codificação do vetor u. Foi acrescentado um loop adicional que simula o efeito de geração dos $p(j)$. Esse loop tem complexidade $O(\log n)$, pois $r = n - k = \lceil \log_2(n+1) \rceil$. Portanto, a complexidade da codificação passa para $O(n \log n)$.

O Algoritmo 9 mostra a decodificação do vetor y. O *loop* inicial (de 3 a n) verifica os bits 1 de y. Para cada bit 1 encontrado, foi acrescentado outro *loop* em r, de complexidade $O(\log n)$, que simula o efeito da geração dos $p(j)$. Portanto, a complexidade total do primeiro *loop* é $O(n \log n)$. Na continuação, temos outro *loop* em n para formação de um número binário posicional. Sua complexidade é $O(n)$. O terceiro *loop*, finalmente, gera o vetor u, de decodificação. Sua complexidade é $O(n)$. Portanto, a complexidade do algoritmo é $O(n \log n)$.

Algoritmo 8: Codificação de $u = u_k u_{k-1} \cdots u_1$:

dados: inteiro: n, **vetor** $u = u_k u_{k-1} \cdots u_1$
início
  $k \leftarrow n - \lceil \log_2(n+1) \rceil$;
  $A \leftarrow [3, 5, 6, 7, \ldots, n]$;
  $r \leftarrow n - k$;
  $v \leftarrow 0 = v_n v_{n-1} \cdots v_1$;
  para j **de** *1* **a** k **:**
    se $u_j = 1$ **então**
      $a \leftarrow A[j] = a_r a_{r-1} \cdots a_1$;
      para l **de** *1* **a** r **:**
        se $a_l = 1$ **então**
          $v[2^{l-1}] \leftarrow v[2^{l-1}] \oplus 1$
      $v[a] \leftarrow v[a] \oplus 1$;
  retorna v;
fim

Algoritmo 9: Decodificação de y:

dados: inteiro: n, **vetor:** y
início
$A \leftarrow [3, 5, 6, 7, \ldots, n]$;
$r \leftarrow \lceil \log_2(n+1) \rceil$;
$v \leftarrow y = y_n y_{n-1} \cdots y_1$; $q \leftarrow 4$; $j \leftarrow 0$;
para i **de** *3* **a** n **:**
se $i \neq q$ **então**
$j \leftarrow j + 1$;
se $y[i] = 1$ **então**
$a \leftarrow A[j] = a_r a_{r-1} \cdots a_1$;
para l *de 1 a* r **:**
se $a[l] = 1$ **então**
$v[2^{l-1}] \leftarrow v[2^{l-1}] \oplus 1$
$v[a] \leftarrow v[a] \oplus 1$;
senão $q \leftarrow 2q$;
$l \leftarrow 0$; $q = 1$;
para i **de** *1 a* n **:**
se $v[i] = 1$ **então**
se $i \neq q$ **então**
retorna erro;
senão
$l \leftarrow l + 2^{i-1}$;
se i=q **então**
$q \leftarrow 2q$;
se $l \neq 0$ **então**
$v[l] = v[l] \oplus 1$
$q \leftarrow 4$; $j \leftarrow 0$
para i *de 3 a* n **:**
se $i \neq q$ **então**
$j \leftarrow j + 1$; $u[j] \leftarrow v[i]$;
senão
$q \leftarrow 2q$
retorna u;
fim

Capítulo 6

Códigos Corretores de Erros de Comprimentos Variáveis

6.1 Definições

Considere um conjunto $A = \{a_1, a_2, \cdots, a_M\}$ de símbolos a serem codificados, onde a frequência em que cada um dos símbolos é transmitido não é uniforme. A cada símbolo a_i é associada uma probabilidade de ocorrência p_i, com $\sum_{i=1}^{M} p_i = 1$.

Um símbolo a_i de A é associado a uma palavra c_i do código $C = \{c_1, c_1, \cdots c_M\}$ de acordo com o comprimento de c_i e a probabilidade de a_i. A palavra de menor comprimento é associada ao símbolo de probabilidade mais alta e a palavra de maior comprimento é associada ao símbolo de probabilidade mais baixa. O **comprimento médio** das palavras do código é definido como $\overline{l} = \sum_{i=1}^{M} p_i |c_i|$.

Um código é dito **livre de prefixo** se nenhuma palavra é prefixo da outra. Analogamente, é dito **livre de sufixo** se nenhuma palavra é sufixo da outra

Exemplo 6.1 *O código* $C = \{0110, 011010, 11011\}$ *não é livre de prefixo, mas é livre de sufixo. O código* $C = \{0110, 100110, 11011\}$ *não é livre de sufixo, mas é livre de prefixo. O código* $C = \{0110, 101101, 11011\}$ *é livre de prefixo e de sufixo.*

A classe de códigos de comprimentos variáveis livres de prefixo são ditos códigos **prompt**, [21]. Um código é dito α-prompt se corrige α erros.

Como as palavras possuem comprimentos variáveis, é necessário expandir o conceito da distância. Sejam duas palavras c_i e c_j tais que $|c_i| \geq |c_j|$. A **distância divergente** entre duas palavras c_i e c_j, $d_d(c_i, c_j)$, é definida como sendo a distância Hamming entre o prefixo de c_i de comprimento $|c_j|$ e c_j. Considerando $c_i = c_{i_1} c_{i_2} \cdots c_{i_{l_i}}$ e $c_j = c_{j_1} c_{j_2} \cdots c_{j_{l_j}}$, onde $|c_i| = l_i$ e $|c_j| = l_j$ e considerando $l_i > l_j$,

podemos escrever:

$$d_d(c_i, c_j) = d(c_{i_1}c_{i_2}\cdots c_{i_{l_j}}, c_{j_1}c_{j_2}\cdots c_{j_{l_j}}).$$

A distância divergente de um código C é a menor das distâncias divergentes entre duas palavras quaisquer de C.

A **distância convergente** entre c_i e c_j, $d_c(c_i, c_j)$, é definida como sendo a distância Hamming entre o sufixo de c_i de comprimento $|c_j|$ e c_j. Podemos escrever:

$$d_c(c_i, c_j) = d(c_{i_{l_i-l_j+1}}c_{i_{l_i-l_j+2}}\cdots c_{i_{l_i}}, c_{j_1}c_{j_2}\cdots c_{j_{l_j}}).$$

A distância convergente de um código C é a menor das distâncias convergentes entre duas palavras quaisquer de C.

A **distância de bloco mínima**, b_k, associada a um comprimento L_k é definida como a distância Hamming entre todas as palavras de comprimento L_k. A distância de bloco mínima global b_{min} é o menor valor de todos os b_k.

O **código estendido** de C é um código de comprimento fixo n definido por:

$$F_n = \{f_i \text{ tal que, } |f_i| = n, f_i = c_{i_1}c_{i_2}\cdots c_{i_{\eta_j}}, c_{i_j} \in C, \ \forall\ j = 1, 2, \cdots \eta_i\}.$$

A **distância livre**, d_{free}, é a menor das distâncias Hamming de todos os códigos estendidos de C. Temos que $d_{free} \geq min(b_{min}, d_c + d_d)$.

A seguir vamos apresentar alguns algoritmos de codificação de decodificação encontrados na literatura.

6.2 Algoritmos de Codificação e Decodificação

Ao longo dos anos, foram desenvolvidos alguns algoritmos para construção de VLECC. Em 1995, Buttigieg [11] desenvolveu três algoritmos distintos. Todos tomando como partida códigos de comprimento fixo. O Code Anti-code Algorithm consiste em remover bits das palavras de um código de comprimento fixo. Já o Greedy Algorithm (G.A.) e Majority Voting Algorithm (M.V.A.) consistem em adicionar bits nas palavras. Iremos descrever brevemente o Code Anti-code adiante.

Em 2011, Ting-Yi Wu [42] desenvolveu um algoritmo que encontra um VLECC ótimo dada uma distância livre d_{free}. O algoritmo consiste em construir uma árvore de busca, onde podem ser encontrados todos os possíveis VLECC's com um número fixo de elementos. Posteriormente, fixando-se um número máximo para a distância livre, encontra-se o código ótimo. Porém, no artigo, não foi feito um estudo sobre a complexidade do algoritmo. Além disso, no processo de decodificação chamado

MAP (Maximum a posterior) assume-se que o destinatário conhece o número de palavras transmitidas.

Por fim, em 2013, foram propostos algoritmos de codificação e decodificação por Richa Gupta, [18], que são bastante eficientes, porém não apresentam resultados muito relevantes. Iremos descrever este código com mais detalhes, pois dentre os citados, é o único que utiliza o parâmetro da distância divergente e portanto é comparável com os códigos que nós encontramos.

Anti-code

Para construir um código com M palavras e $d_{free} = d$, parte-se de um $[n, k, d]$-código C com o menor valor de k tal que $2^k \geq M$ e matriz geradora G.

Toma-se a distância divergente

$$d_d = \left\lfloor \frac{d_{free} + 1}{2} \right\rfloor.$$

Deve-se rearranjar a colunas de C de modo que as m colunas mais a direita formem um código A $[m, k, \sigma]$ com o maior valor de m tal que $\sigma = d_{free} - d_d$.

Para isso, utiliza-se operações elementares nas linhas e colunas de G de modo que a matriz geradora de A tenha o maior número consecutivo de 0's no topo a partir da direita.

Depois deve-se apagar as m colunas mais a direita das primeiras s_1 palavras de C, onde s_1 é o número de palavras com as m colunas mais a direita idênticas. A seguir, deve-se repetir este processo para $m-1, m-2, \cdots$ até não existirem mais colunas para serem apagadas.

Richa Gupta

A ideia é gerar VLEC, chamado Alpha Promt, a partir de códigos lineares corretores de erro de comprimento fixo $[n_1, k_1, d_1]$, $[n_2, k_2, d_2], \cdots, [n_\sigma, k_\sigma, d_\sigma]$.

O código é gerado tomando-se todas as palavras do código $[n_1, k_1, d_1]$ de comprimento n_1, exceto a última, esta é concatenada com todas as palavras do código $[n_2, k_2, d_2]$ de comprimento n_2. A esta última concatenação são concatenadas todas as palavras do código $[n_3, k_3, d_3]$ de comprimento n_3, e assim por diante até $[n_\sigma, k_\sigma, d_\sigma]$.

O número de palavras desse código é dado por:

$$M = \sum_{i=1}^{\sigma} 2^{k_i} - \sigma + 1.$$

O custo do código construído é dado por:

$$c(C) = (2^{k_1} - 1)n_1 + (2^{k_2} - 1)(n_1 + n_2) + \cdots + (2^{k_\sigma} - 1)(n_1 + n_2 + \cdots + n_\sigma).$$

O algoritmo de codificação utiliza uma tabela G, formada com as matrizes geradoras dos códigos $[n_1, k_1, d_1]$, $[n_2, k_2, d_2], \cdots, [n_\sigma, k_\sigma, d_\sigma]$ postas uma abaixo da outra. Assim, G possui k_i linhas de tamanho n_i, para $1 \leq i \leq \sigma$.

Os bits de informação $u_1, \cdots, u_M$ de cada palavra são construídos através do mesmo processo descrito para a construção das palavras.

Cada palavra c_i é formada através da concatenação do resultado da multiplicação dos primeiros k_1 bits de u_i por G_1 com o resultado da multiplicação dos próximos k_2 bits de u_i por G_2 (se existir), e assim por diante até G_σ (se existir).

Note que o código tem capacidade de correção por segmentos. A palavra tem capacidade de correção de e_1 erros nos primeiros n_1 bits, e_2 erros nos próximos n_2 bits e assim por diante.

Exemplo 6.2 *Para construir um VLEC com 12 palavras e que corrige 1 erro, pode se tomar os códigos* $[6, 3, 3], [5, 2, 3]$ *e* $[3, 1, 3]$ *cujas matrizes geradoras são:*

$$G_1 = \begin{bmatrix} 101101 \\ 101010 \\ 011011 \end{bmatrix}, \; G_2 = \begin{bmatrix} 11100 \\ 00111 \end{bmatrix} \; e \; G_3 = \begin{bmatrix} 111 \end{bmatrix}$$

Os bits de informação e as palavras geradas pelo algoritmo são dadas na Tabela 6.1.

Tabela 6.1: VLEC construído a partir dos códigos $[6, 3, 3], [5, 2, 3]$ e $[3, 1, 3]$.

u_i	c_i
0 0 0	0 0 0 0 0 0
0 0 1	0 1 1 0 1 1
0 1 0	1 0 1 0 1 0
0 1 1	1 1 0 0 0 1
1 0 0	1 0 1 1 0 1
1 0 1	1 1 0 1 1 0
1 1 0	0 0 0 1 1 1
1 1 1 0 0	0 1 1 1 0 0 0 0 0 0 0
1 1 1 0 1	0 1 1 1 0 0 0 0 1 1 1
1 1 1 1 0	0 1 1 1 0 0 1 1 1 0 0
1 1 1 1 1 0	0 1 1 1 0 0 1 1 0 1 1 0 0 0
1 1 1 1 1 1	0 1 1 1 0 0 1 1 0 1 1 1 1 1

A Tabela 6.2 apresenta os resultados obtidos em [18] dos códigos gerados pelos algoritmos de Buttigieg e Gupta para um alfabeto de 26 letras que corrige um erro (no caso de Gupta código com distância divergente 3).

Gupta ressalta em seu trabalho as vantagens da velocidade com que seu algoritmo gera os códigos VLEC's. Era de se esperar tal velocidade, já que o que seu algoritmo faz é concatenar códigos já gerados. A média de comprimento das palavras dos códigos gerados por ela, deixa bastante a desejar. Apresentaremos alternativas melhores no capítulo seguinte. E ainda, algo que não parece ter ficado muito claro em sua tese é como escolher os códigos de comprimento fixo para a construção do VLEC. Existem muitas maneiras e não há critérios para escolher a melhor. Também no capítulo seguinte, mostramos a maneira ótima de se concatenar dois códigos de comprimento fixo.

Tabela 6.2: VLEC's com correção de 1 erro.

Símb.	G.A.	G.A.	M.V.A.	M.V.A.	Anticode	Alpha Promt
L_{min}	5	6	5	6	7	7
E	00000	000000	00000	000000	0000000	0000000
T	00111	000111	00111	000111	1001100	0011100
A	11001	011001	11001	011001	0100110	1110000
O	11110	011110	11110	011110	1101010	1101100
N	010100	0010100	010100	0010100	0010011	1010101
R	011010	0011010	011010	0011010	1011111	1001001
I	100110	0100110	100110	0100110	0110101	0100101
S	101000	0101000	101000	0101000	1111001	0111001
H	0101110	1000010	0101110	1000010	00010001	0101010
D	0110010	1001100	0110010	1001100	10001000	0110110
L	1001010	1110000	1001010	1110000	01011101	1011010
F	1010110	1111110	1010110	1111110	11000101	1000110
C	001011100	1010111	001011100	1010111	00110111	1111111
M	001100100	1100101	001100100	1100101	10111111	1100011
U	010010100	1101011	010010100	1101011	01111011	0001111
G	010101100	00110010	010101100	00110010	11100011	0010011000000
Y	00101111100	10010011	00101111100	10010011	000000101	0010011011011
P	00110011100	101000100	00110011100	101000100	100110101	0010011101010
W	01001011110	101100010	01001011110	101100010	010011101	0010011110001
B	01010111100	101101101	01010111100	101101101	110101101	0010011101101
V	0010111111100	101110111	0010111111100	101110111	001001001	0010011110110
K	0011001111100	1010001110	0011001111100	1010001110	101111101	0010011000111
X	0100101110100	1011000010	0100101110100	1011000010	011010001	001001101110000000
J	0101011111100	1011011100	0101011111100	1011011100	111100001	001001101110000111
Q	01011111111100	1011101000	001011111111100	1011101000	000100100	001001101110011100
Z	00110011111110	1111001001	001100111111100	1111001001	100010100	001001101111001101

Capítulo 7

Famílias Especiais de VLECC

Em um VLECC temos que considerar as frequências dos símbolos de um alfabeto e uma quantidade t de erros que ele é capaz de corrigir. De uma forma simplificada, podemos dizer que códigos ótimos seriam aqueles capazes de produzir as menores mensagens codificadas possíveis. Na literatura pesquisada esse problema de otimização encontra-se em aberto. A maioria dos enfoques considera apenas reduções do problema ou a construção de heurísticas para a obtenção de bons códigos. Um exemplo disso é a construção de VLECC através de concatenação de códigos fixos descrita em [41].

Neste trabalho apresentamos uma possibilidade do uso de VLECC's ainda não comentada pelos diversos autores pesquisados, que é o fato de permitirem economia no comprimento da codificação de mensagens em relação aos códigos de comprimento fixo, mesmo para o caso de frequências constantes. Descrevemos dois processos distintos para obtenção de VLECC's com capacidade de correção de 1 erro e com a característica apontada.

A seguir apresentamos uma família infinita de VLECC's com custo inferior ao de códigos de comprimento fixo, mesmo para distribuição uniforme de frequências.

7.1 Uma família VLECC's com custo inferior ao dos correspondentes códigos corretores de comprimento fixo

Tome um código ótimo C_M com $M = A_2(n, 3)$ palavras. Podemos criar um código C_{M+1} com $M + 1$ palavras, a partir de C_M, da seguinte forma: duplicamos uma palavra qualquer de C_M e, a cada uma destas palavras iguais, adicionamos uma sequência de 3 bits com distância Hamming 3 entre si. Note que C_{M+1} possui uma palavra a mais que C_M e a distância divergente 3 é mantida. De forma mais

geral, podemos tomar p palavras de um código C_M para duplicar e, desta forma, construirmos um código com $M + p$ palavras.

Um código ótimo com $M = A_2(n, 3)$ palavras utiliza no máximo $A_2(n, 3) \cdot n$ bits. Portanto, o custo do código construído através do processo descrito é no máximo:

$$A_2(n, 3) \cdot n + pn + 2p \cdot 3,$$

onde o primeiro termo representa o custo de C_M, o segundo, o número de bits das p palavras duplicadas e terceiro, os 3 bits acrescidos em cada uma das p palavras escolhidas e suas duplicações. Note que este custo pode ser ainda menor, se o código de comprimento variável C_{M-p} é melhor do que o código de comprimento fixo para este valor.

É válido aplicarmos este processo apenas nos casos em que o custo do código resultante é menor do que o do código de comprimento fixo, ou seja, quando

$$\begin{aligned} A_2(n, 3) \cdot n + pn + 2p \cdot 3 &< (A_2(n, 3) + p)(n + 1) \\ \Rightarrow\ p &< \frac{A_2(n, 3)}{5}. \end{aligned}$$

Observe que este é um processo de concatenação de códigos. O que se faz aqui é escolher p' palavras para serem prefixos no código resultante C_{M+p}. Cada prefixo dá origem a p_i palavras de modo que $\sum_{i=1}^{p'} p_i = p$. Deste modo, um prefixo contribui em 0 com a distância, logo, os bits adicionais devem por si só formar um código com distância divergente de ao menos 3.

A quantidade de bits adicionados é

$$pn + \sum_{i=1}^{p'} c(C_{p_i}).$$

onde $c(C_{p_i})$ representa o número de bits utilizados pelo código C_{p_i}. Como o código que utiliza o menor número de bits por palavra é o código C_2, concluímos que

$$p \cdot c(C_2) = 6p \leq \sum_{i=1}^{p'} c(C_{p_i}).$$

Logo, a alternativa que adiciona o menor número de bits possível é a utilizada no processo descrito: transformar cada um dos p prefixos em duas palavras.

Descrevemos um método para criar C_M a partir de C_{M-p}, com $M - p = A_2(n, 3)$, se $p < \frac{A_2(n, 3)}{5}$. Porém, observe que se $A_2(n + 1, 3) < A_2(n, 3) + p$, não podemos criar um código de comprimento fixo com $A_2(n, 3) + p$ palavras utilizando $n + 1$ bits.

Portanto, se $A_2(n+1,3) < A_2(n,3) + p$, vamos aplicar o mesmo método e comparar com o código de comprimento fixo que utiliza $n+2$ bits. O método será vantajoso em relação ao código que utiliza $n+2$ bits quando:

$$\begin{aligned} & A_2(n,3)\cdot n + pn + 2p\cdot 3 && < (A_2(n,3)+p)(n+2) \\ \Longrightarrow \quad & (n+6-n-2)p && < -A_2(n,3)\cdot n + (A_2(n,3))(n+2) \\ \Longrightarrow \quad & p && < \frac{A_2(n,3)}{2}. \end{aligned}$$

Logo, o método pode ser aplicado também para os valores

$$A_2(n+1,3) - A_2(n,3) < p < \frac{A_2(n,3)}{2}.$$

Estes valores estão descritos na Tabela 7.1.

Tabela 7.1: Média de bits comparados aos códigos de comprimento fixo $n+2$

M	21	22	23
Fixo	9	9	9
Variável	8,43	8,64	8,83

Observe que existem valores de M para os quais o método é vantajoso nos dois casos. Mas podemos notar analisando as Tabelas 7.5 e 7.1 que o primeiro caso é mais vantajoso. Comprovaremos este resultado algebricamente.

Vamos verificar quando o custo do código criado, C_M, é menor quando é tomado como base o código de comprimento fixo que utiliza $n-1$ bits comparado ao código de comprimento fixo que utiliza n bits (onde n é o maior valor tal que $A_2(n,3) < M$).

Devemos ter:

$$M = A_2(n-1,3) + p' = A_2(n,3) + p$$

Logo, podemos escrever:

$$p' = p + A_2(n,3) - A_2(n-1,3)$$

Assim, o custo do código C_M é menor quando é tomado como base o código de comprimento fixo que utiliza $n-1$ bits quando:

$$\begin{aligned} & [A_2(n-1,3)+p'](n-1) + 6p' && < [A_2(n,3)+p]n + 6p \\ \Rightarrow \quad & [p + A_2(n,3)](n-1) + 6[p + A_2(n,3) - A_2(n-1,3)] && < [A_2(n,3)+p]n + 6p \\ \Rightarrow \quad & A_2(n,3)\cdot(n-1) + 6[A_2(n,3) - A_2(n-1,3)] - A_2(n,3)\cdot n && < (n+6-n+1-6)p \\ \Rightarrow \quad & 5A_2(n,3) - 6A_2(n-1,3) && < p \end{aligned}$$

Listamos na Tabela 7.2 os valores mínimos de p que satisfazem a equação 7.1,

para os valores $5 \leq n-1 \leq 11$.

Tabela 7.2: Valores mínimos de p

$n-1$	5	6	7	8	9	10	11
p	21	41	9	101	153	361	529

Analisando a Tabela 7.2, notamos apenas um valor plausível para p, no caso em que $n-1=7$. Porém, para frequências de símbolos constantes, os códigos gerados a partir de bases com 7 bits são mais vantajosos do que os gerados a partir de bases com 8 bits quando ambos são mais custosos do que os códigos de comprimento fixo que utilizam 9 bits. Ou seja, podemos concluir que quando a frequência dos símbolos é uniforme, é sempre mais vantajoso tomar como base o código com $A_2(n,3)$, onde n é o maior valor tal que $M > A_2(n,3)$.

Já no caso de frequências de símbolos distintas, os códigos construídos através deste método podem apresentar vantagens.

Tabela 7.3: VLEC's com $d=3$ para $M=26$.

Símb.	Alpha Promt	C_{26} Proposto
$\bar{l}$	7,2570	7,6569
E	0000000	0000000
T	0011100	1111111
A	1110000	1000101
O	1101100	1100010
N	1010101	0110001
R	1001001	1011000
I	0100101	0101100 000
S	0111001	0101100 111
H	0101010	0010110 000
D	0110110	0010110 111
L	1011010	0001011 000
F	1000110	0001011 111
C	1111111	0111010 000
M	1100011	0111010 111
U	0001111	0011101 000
G	0010011000000	0011101 111
Y	0010011011011	1001110 000
P	0010011101010	1001110 111
W	0010011110001	0100111 000
B	0010011101101	0100111 111
V	0010011110110	1010011 000
K	0010011000111	1010011 111
X	0010011011100000000	1101001 000
J	0010011011100000111	1101001 111
Q	0010011011100011100	1110100 000
Z	0010011011110011101	1110100 111

Para a distribuição de frequência de uso das 26 letras do alfabeto na língua inglesa feita por Tsai [39], temos que o comprimento médio das palavras do código

proposto C_{26} é $\bar{l} = 7,2570$, apresentado na Tabela 7.3. Este valor apresenta uma boa vantagem em relação ao melhor código fixo que se poderia utilizar, que nos daria uma média de $\bar{l} = 9$. E vantagem ainda sobre o código encontrado por Richa Gupta, que tem média de $\bar{l} = 7,6569$.

Tabela 7.4: Média de bits comparados aos códigos de comprimento fixo $n + 1$

M	9	17	18	19	21	22	23	41	42	43
Fixo	7	8	8	8	9	9	9	10	10	10
Variável	6,66	7,35	7,67	7,95	8,29	8,55	8,78	9,15	9,29	9,42
M	44	45	46	47	73	74	75	76	77	$\cdots$
Fixo	10	10	10	10	11	11	11	11	11	
Variável	9,55	9,67	9,78	9,89	10,08	10,16	10,24	10,32	10,39	

Na próxima seção apresentamos alguns VLECC's especiais, fora da família apresentada e que também são vantajosos em relação aos códigos de comprimento fixo.

7.2 Casos especiais

Para os valores de M iguais a 3, 5, 6 e 10, o método ilustrado na Seção 7.1 não se aplica. Porém, através de uma busca exaustiva encontramos VLECC's com custo menor que os códigos de comprimento fixo. E, ainda, com esta busca, para $M = 9$, melhoramos o custo do VLECC. A técnica exaustiva utilizada é descrita a seguir.

Para formar um código com M palavras, construímos uma árvore com M folhas. Às arestas serão associadas sequências de bits. Assim, cada palavra será formada concatenando-se os bits associados às arestas do caminho percorrido da raiz até uma folha.

A construção da árvore é feita nível a nível. A cada nível precisamos formar ao menos uma palavra que deve ter $d_d \geq 3$ para todos os prefixos formados até então. Analisamos todas as possibilidades de escolhas de k bits para associar às arestas de cada nível. O processo limita-se a construir árvores onde o total dos comprimentos das M palavras seja inferior ao do correspondente código de comprimento fixo.

Note que, para o primeiro nível, $k \geq 3$, pois com menos do que isso, não é possível criar uma palavra com $d_d \geq 3$ para os outros prefixos.

Vamos exemplificar o método para $M = 3$. Consideremos a primeira escolha de bits para associar a uma aresta sendo 000. Neste caso, só existe um segmento com 3 bits e $d_d \geq 3$ para 000, o segmento 111, que é então associado a uma segunda aresta neste primeiro nível. Assim, 000 forma uma palavra do código e 111 é o prefixo das outras duas palavras. No segundo nível, devemos criar dois filhos para o nó associado à aresta 111. O número mínimo de bits que precisamos associar a cada

aresta deste nível é 3, já que as palavras possuirão o mesmo prefixo, 111, e devem ter $d_3 = 3$. Isto pode ser feito associando-se os segmentos 000 e 111. Assim criamos o código $C = \{000, 111000, 111111\}$. Observe que este é o mesmo código criado para $M = 3$ do caso anterior. Se com esta mesma técnica iniciamos o processo com a primeira escolha sendo 0000 ao invés de 000, obtemos um código melhor.

Alguns códigos encontrados através deste processo são apresentados a seguir.

C_3 =	C_5 =	C_6 =	C_{10} =
0000	0000	00000	000000
01110	011100	11011	011110
10111	101101	011011	101011
	110110	011100	110101
	111011	101010	0100111
		101101	0110010
			1001101
			1011000
			1100100
			1110001

7.3 Conclusão

Mostramos que, para certas quantidades M de símbolos a serem codificados (algumas ilustradas na Tabela 7.5) é mais vantajosa a utilização de codificações com comprimentos variáveis do que comprimentos fixos para frequências constantes. Salientamos que, para frequências distintas dos símbolos, a utilização de códigos de comprimentos variáveis vem a ser ainda melhor do que de códigos de comprimento fixo. Fizemos o estudo para códigos com distância divergente 3, porém a técnica proposta pode ser estendida para qualquer d_d.

Tabela 7.5: Média de bits para códigos de comprimento fixo e variável e $d = 3$

M	3	5	6	9	10	11	12	17	18	19	21	22	23
Fixo	5	6	6	7	7	7	7	8	8	8	9	9	9
Variável	4,67	5,6	5,66	6,44	6,6	6,63	6,66	7,35	7,67	7,95	8,29	8,55	8,78

Capítulo 8

Conclusões

Neste trabalho, inicialmente, apresentamos conceitos básicos sobre códigos corretores de erros. Principalmente sobre códigos lineares e especificamente os códigos de Hamming. Discutimos a complexidade computacional da codificação e decodificação de códigos lineares.

Em seguida, desenvolvemos dois códigos lineares equivalentes a códigos Hamming encurtados. O primeiro denominado, $Gham(n)$, é também uma generalização para os códigos de Hamming. E o segundo denominado, Código Binário Posicional, $BP(n)$. Tais códigos são códigos lineares ótimos de distância Hamming 3, isto é, têm capacidade de corrigir um erro. Descrevemos todas as suas estruturas e características. Apresentamos algoritmos de construção recursiva, algoritmos de codificação e decodificação com complexidade $O(n)$. Acreditamos que esses códigos sejam uma contribuição relevante para a teoria dos códigos corretores de erros, pois são códigos facilmente implementáveis, estão definidos para todo inteiro $n \geq 3$.

Na segunda parte do trabalho, apresentamos duas famílias de códigos corretores de comprimento variável. Tais famílias representam uma economia de custo significativa em relação aos códigos de comprimento fixo e ainda em relação aos códigos de comprimento variável encontrados na literatura. Mostramos a maneira ótima se utilizar a técnica de concatenação de códigos de comprimento fixo para se criar um código de comprimento variável.

Os trabalhos futuros que podem ser desenvolvidos a partir desta tese são:

- Estender os códigos lineares ótimos para distâncias Hamming $d > 3$. Observe que conseguimos estender a construção gulosa para distâncias 5 e 7 restritas a $n \leq 22$.

- Utilizar as técnicas de construção de códigos descritas neste trabalho para códigos não lineares. Conseguimos estender parcialmente as técnicas utilizadas para obter alguns códigos baseados na proibição de certas configurações,

obtendo códigos ótimos para $8 \leq n \leq 11$ e códigos subótimos para outros valores. Permanece como desafio conseguir melhorias na Tabela 1.1.

- Encontrar códigos VLEC's ótimos para outras quantidades M de palavras ainda não cobertas pela família infinita de códigos desenvolvida.
- Determinar as complexidades de codificação e decodificação dos códigos VLEC's desenvolvidos neste trabalho.
- Permanece como desafio determinar também a complexidade do problema de otimização ligado à construção dos códigos VLEC's.

Referências Bibliográficas

[1] Agrell, E., Vardy, A., Zeger, K., 2001, "A table of upper bounds for binary codes", *IEEE Transactions on Information Theory*, v. 47, n. 7, pp. 3004–3006.

[2] Barg, A., 1998, "Complexity issues in coding theory", *Handbook of Coding theory*, v. 1, pp. 649–754.

[3] Barthelmy, J., Cohen, G., Lobstein, A., 1997, *Algorithmic Complexity and Telecommunication Problems*. CRC Press.

[4] Baylis, D. J., 1997, *Error Correcting Codes: A Mathematical Introduction*, v. 15. CRC Press.

[5] Berlekamp, E. R., McEliece, R. J., Van Tilborg, H. C., 1978, "On the inherent intractability of certain coding problems", *IEEE Transactions on Information Theory*, v. 24, n. 3, pp. 384–386.

[6] Best, M., 1980, "Binary codes with a minimum distance of four (Corresp.)", *IEEE Transactions on Information Theory*, v. 26, n. 6, pp. 738–742.

[7] Bondy, J. A., Murty, U. S. R., 1976, *Graph Theory with Applications*, v. 290. Macmillan London.

[8] Bose, R. C., 1947, "Mathematical theory of the symmetrical factorial design", *Sankhyā: The Indian Journal of Statistics*, pp. 107–166.

[9] Bruck, J., Naor, M., 1990, "The hardness of decoding linear codes with preprocessing", *IEEE Transactions on Information Theory*, v. 36, n. 2, pp. 381–385.

[10] Buddha, S. K., 2011, "Hamming and Golay Codes", *Indiana State University, 2011.*

[11] Buttigieg, V., 1995, *Variable-Length Error-Correcting Codes*. Tese de Doutorado, Department of Electrical Engineering, University of Manchester, En-

gland. Available online: <http://staff.um.edu.mt/vbut1/research/PhDThesis.pdf>.

[12] Buttigieg, V., Farrell, P. G., 1994, "On variable-length error-correcting codes". In: *Information Theory, 1994. Proceedings., 1994 IEEE International Symposium on*, p. 507. IEEE.

[13] Cohen, G., Honkala, I., Litsyn, S., et al., 1997, *Covering Codes*, v. 54. Elsevier.

[14] Cull, P., Nelson, I., others, 1998, *Perfect codes, NP-completeness, and towers of Hanoi graphs*. Relatório técnico, Corvallis, OR: Oregon State University, Dept. of Computer Science.

[15] Elssel, K., Zimmermann, K.-H., 2005, "Two new nonlinear binary codes", *IEEE transactions on information theory*, v. 51, n. 3, pp. 1189–1190.

[16] Fujiwara, T., Kasami, T., Lin, S., 1989, "Error detecting capabilities of the shortened Hamming codes adopted for error detection in IEEE standard 802.3", *IEEE Transactions on Communications*, v. 37, n. 9, pp. 986–989.

[17] Garrammone, G., 2013, "On decoding complexity of Reed-Solomon codes on the packet erasure channel", *IEEE Communications Letters*, v. 17, n. 4, pp. 773–776.

[18] Gupta, R., 2013, *Variable Length Error Correcting Codes and Reversible Variable Length Codes: Analysis and Applications*. Tese de Doutorado, Jaypee Institue of Information Technology, Noida, Índia.

[19] Hamalainen, H., 1988, "Two new binary codes with minimum distance three", *IEEE transactions on information theory*, v. 34, n. 4, pp. 875.

[20] Hamming, R. W., 1950, "Error detecting and error correcting codes", *Bell System Technical Journal*, v. 29, n. 2, pp. 147–160.

[21] Hartnett, W. E., 2012, *Foundations of Coding Theory*, v. 1. Springer Science & Business Media.

[22] Hill, R., 1986, *A First Course in Coding Theory*. Oxford University Press.

[23] Honkala, I., 1987, "Bounds for binary constant weight and covering codes", *Licentiate thesis, Dept. Math., Univ. of Turku, Turku, Finland*.

[24] Kaikkonen, M., 1998, "Codes from affine permutation groups", *Designs, Codes and Cryptography*, v. 15, n. 2, pp. 183–186.

[25] Kaikkonen, M. K., 1989, "A new four-error-correcting code of length 20", *IEEE transactions on information theory*, v. 35, n. 6, pp. 1344.

[26] Laaksonen, A., Östergård, P. R., 2017, "Constructing error-correcting binary codes using transitive permutation groups", *Discrete Applied Mathematics*, v. 233, pp. 65–70.

[27] MacWilliams, F. J., Sloane, N. J. A., 1977, *The Theory of Error Correcting Codes*. Elsevier.

[28] Milshtein, M., 2015, "A new binary code of length 16 and minimum distance 3", *Information Processing Letters*, v. 115, n. 12, pp. 975–976.

[29] Ntafos, S. C., Hakimi, S. L., 1981, "On the complexity of some coding problems (corresp.)", *Information Theory, IEEE Transactions on*, v. 27, n. 6, pp. 794–796.

[30] Östergård, P. R., 2005, "Two new four-error-correcting binary codes", *Designs, Codes and Cryptography*, v. 36, n. 3, pp. 327–329.

[31] Ostergard, P. R., 2011, "On the size of optimal three-error-correcting binary codes of length 16", *IEEE Transactions on Information Theory*, v. 57, n. 10, pp. 6824–6826.

[32] Ostergard, P. R., Baicheva, T., Kolev, E., 1999, "Optimal binary one-error-correcting codes of length 10 have 72 codewords", *IEEE Transactions on Information Theory*, v. 45, n. 4, pp. 1229–1231.

[33] Pedroza, N., Pinto, P. E. D., Szwarcfiter, J. L., 2015, "Códigos corretores de erros de tamanhos variáveis", *XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*. Available online: <`http://sbseg2015.univali.br/anais/SBSegResumos/resumoEstendido09.pdf`>.

[34] Pedroza, N., Pinto, P. E. D., Szwarcfiter, J. L., 2016, "Error Correcting Codes and Cliques of Graphs", *Anais do Latin American Workshop on Cliques in Graphs*.

[35] Pedroza, N., Pinto, P. E. D., Szwarcfiter, J. L., 2016, "Variable Length Error Correcting Codes", *São Paulo School of Advanced Science on Algorithms, Combinatorics and Optimization*. Available online: <`http://www.ime.usp.br/~spschool2016/wp-content/uploads/2016/07/PedrozaNatalia.pdf`>.

[36] Pedroza, N., Pinto, P. E. D., Szwarcfiter, J. L., 2017, "Uma Generalização dos Códigos Hamming", *II Encontro de Teoria da Computação*.

[37] Porter, S. C., Shen, B.-Z., Pellikaan, R., 1992, "Decoding geometric Goppa codes using an extra place", *IEEE Transactions on Information Theory*, v. 38, n. 6, pp. 1663–1676.

[38] Slepian, D., 1960, "Some further theory of group codes", *Bell System technical journal*, v. 39, n. 5, pp. 1219–1252.

[39] Tsai, C.-W., Wu, J.-L., 2001, "On constructing the Huffman-code-based reversible variable-length codes", *Communications, IEEE Transactions on*, v. 49, n. 9, pp. 1506–1509.

[40] Van Pul, C., 1982, "On bounds on codes", *Master's thesis, Dept. of Mathematics and Computing Science, Eindhoven Univ. of Technology, Eindhoven, The Netherlands*.

[41] Wenisch, T., Swaszek, P. F., Uht, A. K., 2001, "Combined error correcting and compressing codes". In: *Information Theory, 2001. Proceedings. 2001 IEEE International Symposium on*, p. 238. IEEE.

[42] Wu, T.-Y., Chen, P.-N., Alajaji, F., et al., 2011, "On the construction and MAP decoding of optimal variable-length error-correcting codes". In: *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pp. 2223–2227. IEEE.