Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Comunicações

ALGORITMOS DE DECODIFICAÇÃO ABRUPTA PARA CÓDIGOS LDGM

Autor: Fernando Pujaico Rivera

Orientador: Prof. Dr. Jaime Portugheis

Tese de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: **Telecomunicações e Telemática.**

Banca Examinadora

Prof. Dr. Jaime Portugheis (presidente) — FEEC/UNICAMP
Prof. Dr. Marcelo Firer — IMECC/UNICAMP
Prof. Dr. Gustavo Fraidenraich — FEEC/UNICAMP

Campinas – SP
03/06/2011

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Pujaico Rivera, Fernando
P965a Algoritmos de Decodificação Abrupta
para Códigos LDGM
Fernando Pujaico Rivera. – Campinas, SP:
[s.n.], 2011.

Orientador: Jaime Portugheis.
Dissertação de Mestrado - Universidade Estadual de Campinas,
Faculdade de Engenharia Elétrica e de Computação.

1. Teoria da codificação. 2. Códigos de controle de erros.
I. Portugheis, Jaime. II. Universidade Estadual de Campinas.
Faculdade de Engenharia Elétrica e de Computação. III.
Título

Título em Inglês:	Hard Decision Algorithms for LDGM Codes
Palavras-chave em Inglês:	Coding Theory
	Error Control Codes
Área de concentração:	Telecomunicações e Telemática
Titulação:	Mestre em Engenharia Elétrica
Banca Examinadora:	Marcelo Firer IMECC Unicamp
	Gustavo Fraidenraich FEEC Unicamp
Data da defesa:	03/06/2011
Programa de Pós Graduação:	Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Fernando Pujaico Rivera

Data da Defesa: 3 de junho de 2011

Título da Tese: "Algoritmos de Decodificação Abrupta para Códigos LDGM"

Prof. Dr. Jaime Portugheis (Presidente): _____

Prof. Dr. Marcelo Firer: _____

Prof. Dr. Gustavo Fraidenraich: _____

Resumo

Desde que Gallager introduziu o algoritmo de decodificação Bit-Flipping (BF) com decisão abrupta para códigos Low Density Parity Check ($LDPC$), outras duas variantes foram propostas por Sipser e Spielman para os códigos conhecidos como "Expander Codes". Posteriormente, uma versão da decodificação BF por decisão suave conhecida como decodificação Modified Weighted BF (MWBF), foi investigada. Esta tese propõe versões modificadas dos algoritmos de Sipser e Spielman. Resultados de simulações para códigos Low Density Generator Matrix ($LDGM$) sistemáticos, com comprimento longo mostraram um melhor desempenho da versão proposta. Adicionalmente, para um comprimento médio dos códigos LDGM, resultados de simulações mostraram um desempenho similar à decodificação MWBF com a vantagem de não ser necessário o uso de operações em ponto flutuante.

Palavras-chave: Códigos Corretores de Erro, Códigos de Matrizes Geradoras de Baixa Densidade, Algoritmos de Decodificação, Decodificação por Decisão Abrupta.

Abstract

Since Gallager introduced Bit-Flipping (BF) decoding with hard-decision for Low-Density Parity-Check Codes ($LDPC$), other two variants were proposed by Sipser and Spielman for expander codes. Later, a soft-decision version of BF decoding, known as Modified Weighted BF ($MWBF$) decoding, was investigated. This thesis proposes modified versions of Sipser and Spielman algorithms. Simulation results for long systematic Low-Density Generator Matrix ($LDGM$) codes show a better performance of the proposed versions. Moreover, for moderate length systematic LDGM codes, simulation results show performance similar to that of MWBF decoding with the advantage of not requiring floating-point operations.

Keywords: Error correcting codes , Low density generator matrix codes, Decoding algorithms, Hard decision decoding.

Agradecimentos

Ao meu orientador Prof. Jaime Portugheis, sou grato pela orientação.

Aos demais colegas de pós-graduação, pelas críticas e sugestões.

A minha família pelo apoio durante esta jornada.

À CAPES, pelo apoio financeiro.

Aos meus pais, irmão, avó e tios

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

G - Matriz geradora
H - Matriz de verificação de paridade
U - Vetor de informação
V - Vetor de dados enviado pelo canal
R - Vetor de dados recebido pelo canal

Capítulo 1

Introdução

Gallager introduziu em [1] os códigos de baixa densidade de verificação de paridade também chamados códigos $LDPC$ (do inglês Low-Density Parity-Check). Gallager também propôs um algoritmo para decodificação por decisão abrupta conhecido como decodificação BF (do inglês Bit-Flipping). Sipser an Spielman estudaram em [2] uma sub-classe de códigos $LDPC$, chamados "expander codes", e propuseram duas novas formas de decodificação BF por decisão abrupta. Kou et al. propuseram em [3] uma versão por decisão suave da decodificação BF conhecida como decodificação WBF (do inglês Weighted BF). Posteriormente, Zhang e Fossorier em [4] modificaram a função de troca de bits da decodificação WBF acrescentando um fator de correção que depende de um parâmetro, α, a ser optimizado. Esta variante da decodificação BF é conhecida como decodificação $MWBF$ (do inglês Modified WBF). Os desempenhos de todas as versões de decisão suave da decodificação BF em [4] foram obtidos para códigos $LDPC$. Estes mostraram um melhor desempenho quando comparados à decodificação BF com decisão abrupta. A decodificação por decisão suave requer operações de ponto flutuante que são computacionalmente custosas. Isto implica que em algumas situações a decodificação por decisão abrupta de códigos longos pode ser preferível. Nesta dissertação se investiga o desempenho da decodificação BF com decisão abrupta para matrizes geradoras $LDGM$ (do inglês Low Density Generator Matrix) sistemáticas. Primeiro, propõe-se uma versão modificada do algoritmo de decisão abrupta de Sipser e Spielman. Esta modificação na sua versão sequencial será chamada de "serial hard BF" e a versão paralela de "parallel hard BF ". Os resultados das simulações para códigos longos mostram um melhor desempenho das versões modificadas propostas. Também se compara o desempenho da decodificação $MWBF$ com algoritmos propostos pela decodificação BF por decisão suave considerando um comprimento médio de código. Os resultados da simulação mostraram que não há melhora significativa do algoritmo $MWBF$ sobre os algoritmos propostos de decodificação BF por decisão abrupta.

1.1 Organização da dissertação

A tese está organizada como segue:

No capítulo 2, são apresentados os fundamentos da teoria das comunicações, o modelo do sistema de transmissão digital usado e a notação das variáveis empregadas. Também se apresenta um estudo da capacidade do canal e os modelos de canal usados e finalmente se faz uma introdução à codificação de dados.

No capítulo 3, se faz um estudo dos códigos corretores de erros usados nesta dissertação assim como um estudo das distintas formas de criar as matrizes geradoras e de verificação de paridade para aos códigos de bloco empregados.

No capítulo 4, mostra-se os algoritmos de decodificação usados para os códigos de bloco vistos no capitulo 3, além de um estudo sobre a taxa de erro de bit e os limites máximos possíveis a serem atingidos pelos algoritmos de decodificação. Para finalizar este capítulo apresenta-se as simulações para cada um dos algoritmos de decodificação.

No capítulo 5, são descritas as conclusões da dissertação de acordo com os resultados das simulações do capitulo 4 e se faz a análise sobre os resultados comparando as distintas versões dos algoritmos BF existentes com a versão modificada para decisão abrupta proposta nesta dissertação.

Para finalizar, no apêndice A pode ser encontrada uma pequena ajuda para o capítulo 3, onde se explica um algoritmo para a geração de um campo de Galois. No apêndice B se descreve os programas usados para a simulação assim como a forma de usar os mesmos.

1.2 Contribuições da dissertação

As principais contribuições desta dissertação são:

- Uma modificação do algoritmo de decodificação "bit-flipping" com decisão abrupta. Este algoritmo gerou resultados muito próximos aos obtidos com a versão de decisão suave, versão esta que possui maior custo computacional. Em alguns casos, para a versão paralela da modificação do algoritmo, os resultados mostraram uma melhora no desempenho da taxa de erro de bit.

- Um estudo do comportamento das distintas variantes do algoritmo de decodificação "bit-flipping" para matrizes $LDGM$.

- Estudo do desempenho da versão modificada do algoritmo "bit-flipping" para matrizes $LDGM$ sistemáticas e matrizes $LDPC$ regulares.

Capítulo 2

Fundamentos da Teoria das Comunicações

Numa transmissão de dados se deseja enviar informação entre dois pontos, o emissor e o receptor, como mostra a Figura 2.1. Para conseguir isto se faz uso de um meio físico de transmissão denominado canal físico. Este meio pode ser o ar, a fibra ótica, cabos condutores de eletricidade, etc. Dado que o canal é um meio físico, a informação enviada por ele tem sempre um caráter analógico, pois por ele transita uma onda eletromagnética, luz, sinais elétricos, etc.

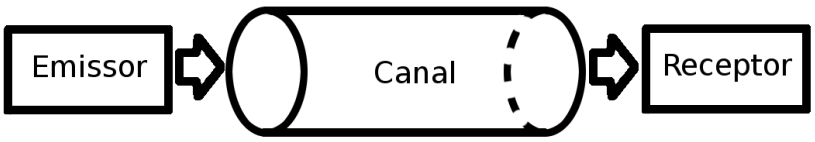

Fig. 2.1: Diagrama de uma transmissão de dados.

2.1 Modelo de um sistema de transmissão digital

Se diz que se tem uma transmissão digital de dados, quando o emissor codifica a informação enviada de forma digital, isto é, num conjunto limitado e fixo de valores e sinais possíveis. Na Figura 2.2 se pode ver como o modulador recebe dados digitais (conjunto limitado e fixo de valores) e os mapeia num conjunto limitado e fixo de sinais analógicos. Esses sinais transitam pelo canal onde sofrem transformações por causa da presença do ruido. À saída do canal, o demodulador entrega ao decodificador de canal os dados com alguns elementos errados. Este bloco é o encarregado de

diminuir a probabilidade de erro dos dados recebidos.

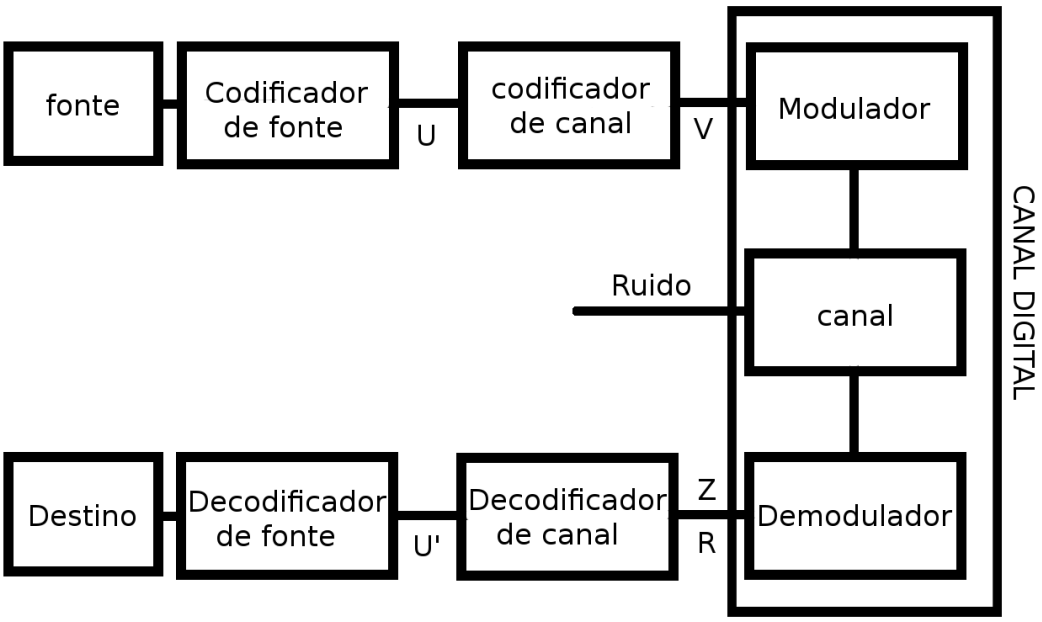

Fig. 2.2: Diagrama de uma transmissão digital de dados.

Em geral um dado digital, geralmente BCD (do inglês Binary Coded Digit), inicia seu percurso saindo da fonte. Estes dados são recebidos pelo codificador de fonte encarregado de codificar para retirar possível redundância .inerente aos dados da fonte. Dado um vetor binário U este é codificado pelo codificador de canal onde ele sofre uma transformação obtendo-se um vetor V. Isto é feito para acrescentar a ele redundância com o fim de proteger a informação através do seu trânsito pelo canal.

Quando o modulador recebe os dados ele seleciona uma forma de onda de duração T. Para uma codificação binária o modulador gera um sinal $S_0(t)$ quando ele recebe o bit 0 e um sinal $S_1(t)$ quando recebe o bit 1. Se temos um canal de faixa estreita, uma boa escolha são os seguintes sinais:

$$S_0(t) = \sqrt{\frac{2E}{T}} sin(2\pi f_0 t + \frac{\pi}{2}) \quad 0 \leq t \leq T \tag{2.1}$$

$$S_1(t) = \sqrt{\frac{2E}{T}} sin(2\pi f_0 t - \frac{\pi}{2}) \quad 0 \leq t \leq T \tag{2.2}$$

onde E é a energia de cada sinal e f_0 é a frequência da portadora. Geralmente se escolhe f_0 de tal forma que seja um múltiplo de $1/T$. A este tipo de sinalização se chama BPSK (do inglês Binary Phase Shift Keyed).

Estes sinais analógicos foram transmitidos pelo canal, sofrendo nele distorções. E o demodulador produzirá na sua saída um número real ou um conjunto finito e discreto de valores. Um demodulador ótimo sempre inclui um filtro casado, ou um correlator seguido de um seletor por amostragem. Dependendo da etapa de decisão usada podem-se obter diferentes modelos de canal. Os dados recebidos no vetor R são entregues ao decodificador de canal que usará os bits de redundância para tentar detetar e corrigir os erros, obtendo neste processo um vetor U' que se estima que é muito próximo ao vetor U enviado. Este vetor U' será utilizado no destino.

2.2 Capacidade do canal

Antes falar dos modelos de canais de comunicações é necessário falar da capacidade do canal. A capacidade do canal (C) é a máxima quantidade de informação que é possível enviar por um canal de forma confiável.

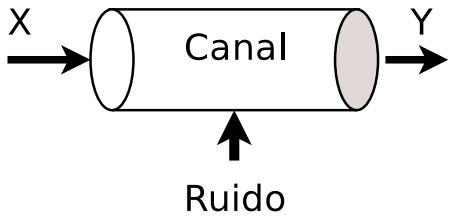

Fig. 2.3: Canal de comunicações.

Para um canal como o mostrado na Figura 2.3 onde a capacidade do canal é a máxima quantidade de informação em comum existente entre sua entrada X e sua saída Y, a equação,

$$I(X;Y) = H(Y) - H(Y|X) \tag{2.3}$$

mostra esta quantidade mediante a definição da informação mútua $I(X;Y)$ para duas variáveis aleatórias X e Y, onde as funções de entropia são dadas por (2.4) e (2.5).

$$\begin{aligned} H(Y) &= -E[log_2(Pr(y))] \\ &= \textstyle\sum_y Pr(y) log_2(\frac{1}{Pr(y)}) \end{aligned} \tag{2.4}$$

$$\begin{aligned} H(Y|X) &= -E[log_2(Pr(y|x))] \\ &= \textstyle\sum_x \sum_y Pr(x,y) log_2(\frac{1}{Pr(y|x)}) \\ &= \textstyle\sum_x \sum_y Pr(x) Pr(y|x) log_2(\frac{1}{Pr(y|x)}) \end{aligned} \tag{2.5}$$

A patir das equações (2.3), (2.4) e (2.5) é possível definir a capacidade do canal C como a máxima

quantidade de informação mutua entre X e Y sobre um canal de comunicações com uma probabilidade de entrada $Pr(x)$:

$$C = max_{Pr(x)} I(X;Y) \tag{2.6}$$

2.3 Modelos de canais

Neste ponto é importante definir os modelos de canal que serão usados. Eles se distinguem pela forma como o ruído é introduzido e o tipo de dados que aparecem na sua saída.

2.3.1 Canal BSC

Um canal binário simétrico (do inglês Binary Symmetric Channel, BSC) é um canal de comunicação derivada da probabilidade de erro de bit na sua recepção. Como pode-se ver na Figura 2.4, se o bit enviado é 0 então tem-se uma probabilidade $1-p$ de receber um bit 0 na sua saída e uma probabilidade p de receber um bit. Agora se é enviado um 1 se tem uma probabilidade $1-p$ de receber um bit 1 na sua saída e uma probabilidade p de receber um bit 0.

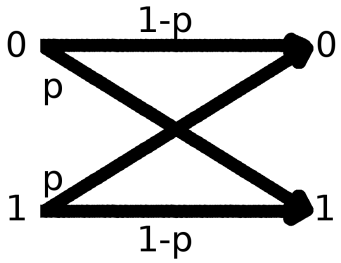

Fig. 2.4: Canal BSC.

Se definimos uma variável aleatória X na entrada do canal BSC e uma variável aleatória Y na saída, obtemos as seguintes relações:

$$Pr(Y = 0|X = 0) = 1 - p \tag{2.7}$$

$$Pr(Y = 0|X = 1) = p \tag{2.8}$$

$$Pr(Y = 1|X = 0) = p \tag{2.9}$$

$$Pr(Y = 1|X = 1) = 1 - p. \tag{2.10}$$

Um ponto interessante a notar é que num canal BSC com uma probabilidade de erro p, o valor p tem que estar entre $0 < p < 0.5$. Isto por causa da simetria do canal, pois se $0.5 < p < 1$ bastaria acrescentar uma etapa mais ao canal que invertesse os bits de 1 a 0 e vice-versa, para obter a mesma incerteza que um canal com probabilidade de errar igual a $1 - p$. Pode-se ver isto facilmente se calculamos a capacidade do canal. Lembrando que num canal BSC se transmite um só bit por uso do canal, temos,

$$C = 1 - H(p). \tag{2.11}$$

A equação (2.11) mostra que a capacidade do canal BSC é 1 bit diminuído pela quantidade de informação do erro do canal $H(p)$,

$$H(p) = -p\, log_2(p) - (1 - p)log_2(1 - p). \tag{2.12}$$

A função entropia é simétrica em torno de $p = 0.5$, pelo qual C também é simétrica em torno deste ponto. Assim a capacidade do canal para transmitir informação e igual para uma probabilidade de erro de p e $1 - p$.

2.3.2 Canal AWGN

Um canal de ruído gaussiano aditivo branco ou AWGN (do inglês Additive White Gaussian Noise) é um canal de comunicações em que o ruido é adicionado linearmente à informação transmitida. Se definimos a entrada do canal como X real, a saída do canal como Y real e ao ruído gaussiano adicionado como Z real a equação resultante seria a seguinte,

$$Y = X + Z, \tag{2.13}$$

onde Z tem a seguinte função de densidade de probabilidade,

$$f_Z(z) = \frac{1}{\sqrt{2\pi\sigma^2}}\, exp[-\frac{z^2}{2\sigma^2}]. \tag{2.14}$$

A quantidade máxima de informação que pode transportar o canal $AWGN$ de forma confiável esta dada pela seguinte equação,

$$C = \frac{1}{2}\, log_2(1 + \frac{S}{N}) \tag{2.15}$$

$$S = E[X^2] \tag{2.16}$$

$$N = E[Z^2] = \sigma^2 \tag{2.17}$$

onde C é a capacidade do canal, S é a potência do sinal na entrada do canal e N é a potência do ruído presente no canal. Se consideramos agora um canal $AWGN$ de faixa limitada a B, temos a capacidade

$$C = B\ log_2(1 + \frac{S}{N}) \tag{2.18}$$

N é agora o ruido $AWGN$ contido na faixa B, e S é a potência de sinal nesta faixa.

Na pratica, existe uma restrição física da máxima potencia P_{Maxima} possível a ser enviada pelo transmissor. A seguinte equação descreve esta relação. Assim

$$S \leq P_{Maxima} \tag{2.19}$$

2.3.3 Decodificação por decisão suave e abrupta

De acordo com o modelo de canal de transmissão empregado existem duas possibilidades para obter-se dados na saída do demodulador; eles podem ser reais ou digitais. Pode-se obter qualquer valor real na saída do demodulador, ou um dado pertencente a um conjunto limitado e fixo de valores. No primeiro caso como a saída é real se diz que o demodulador do canal usará decisão suave (do inglês "soft-decision"), Se utilizamos dados digitais falamos de uma decodificação por decisão abrupta (do inglês "hard-decision").

Decodificação por decisão suave

Um exemplo de decisão suave é a decodificação feita ao final de um canal $AWGN$. Como se viu na seção 2.3.2, a saída do canal pode tomar valores reais negativos ou positivos. É então que o decodificador do canal faz uma decisão "$soft$" ou suave dos valores recebidos. É suave no sentido que não mexe nos dados, e os trabalha tal como eles saíram do canal. É assim que neste modelo a decodificação se faz sobre um vetor R com elementos reais. É dele que irá se obter um vetor U' de K elementos binários, ver Figura 2.5.

Se pode ver que neste tipo de decodificação tem-se mas informação para ajudar na obtenção do vetor U', dado que se tem um conjunto infinito de dados possíveis recebidos. Mas esta aparente

Fig. 2.5: Decodificação por decisão suave.

vantagem gera o problema de fazer a decodificação mais complexa por causa do uso de operações com ponto flutuante em todo o processo de decodificação.

Decodificação por decisão abrupta

Quando se diz que se vai realizar uma decodificação por decisão abrupta quando na entrada do decodificador de canal se tem dados digitais, geralmente binários. Dada a natureza analógica de todos os canais de transmissão, obter dados digitais na saída do demodulador, depende de uma decisão prévia sobre os símbolos da modulação. Esta decisão é chamada decisão abrupta.

Considere um canal $BI - AWGN$ onde na entrada tenha-se um bit com valor 1 que será codificado como $+A$ e um bit 0 como $-A$, Estes valores serão enviados pelo canal $AWGN$ e obteremos na sua saída valores reais com uma distribuição do ruido como na equação (2.14). Assim, a probabilidade de receber o dado Y dado que o bit $U_l = 1$ foi enviado é dada por

$$f(Y|U_l = 1) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(y-1)^2}{2\sigma^2}} \tag{2.20}$$

e para o caso que seja enviado o bit $U_l = 0$ a probabilidade é

$$f(Y|U_l = 0) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(y+1)^2}{2\sigma^2}} \tag{2.21}$$

Neste ponto aplica-se uma decisão máximo a posteriori (MAP), que consiste em aplicar a seguinte regra

$$z = \begin{cases} 1 & f(U_l = 1|Y) > f(U_l = 0|Y) \\ 0 & caso\ \ contrario \end{cases} \tag{2.22}$$

isto é, nos perguntamos que tem maior probabilidade, que seja 1 ou que seja 0, dado que se recebeu Y. Aplicado o teorema de Bayes pode-se chegar a uma expressão como as equações

$$f(U_l = 0|Y) = \frac{f(Y|U_l = 0)p(U_l = 0)}{f(Y)} \tag{2.23}$$

$$f(U_l = 1|Y) = \frac{f(Y|U_l = 1)p(U_l = 1)}{f(Y)}, \tag{2.24}$$

que só dependem das probabilidades $f(U_l = 0|Y)$ e $f(U_l = 1|Y)$ e das probabilidades de enviar pelo canal um bit 0 ou um bit 1, $p(U_l = 0)$ e $p(U_l = 1)$, respectivamente.

Aplicando (2.23) e (2.24) na equação (2.22) obtemos:

$$z = \begin{cases} 1 & f(Y|U_l = 1)p(U_l = 1) > f(Y|U_l = 0)p(U_l = 0) \\ 0 & caso\ \ contrario \end{cases} \tag{2.25}$$

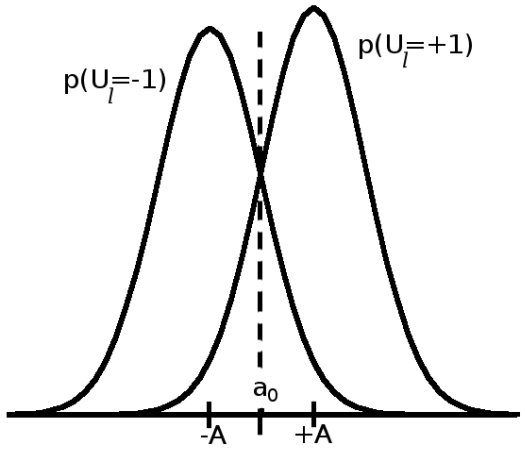

Fig. 2.6: Funções densidade de probabilidade.

A Figura 2.6 mostra de forma gráfica a equação (2.25). Todos os valores de Y maiores a a_0 indicam que foi enviado um bit $U_l = 1$, caso contrario foi enviado um bit $U_l = 0$, ficando a lei de decisão como:

$$z = \begin{cases} 1 & se\ \ Y > a_0 \\ 0 & caso\ \ contrario \end{cases} \tag{2.26}$$

Para o caso em que $p(U_l = 0) = p(U_l = 1)$ o limiar de decisão fica no valor $a_l = 0$, onde qualquer valor positivo de Y será considerado como um bit $z = 1$ e qualquer valor negativo será considerado como um bit $z = 0$. Este tipo de decodificação é chamada de "$hard$" ou abrupta, porque a decisão do valor na saída do canal é feito de forma abrupta, pois é um valor ou outro, não tem pontos intermediários.

Pode-se ver em [5] na equação (1.4) que no caso que se tenha um canal $AWGN$ com uma etapa

de decisão abrupta na sua saída, pode-se relacionar o ruido introduzido no canal $AWGN$ com a probabilidade de erro p resultante através da seguinte equação:

$$p = Q(\sqrt{\frac{2R_cE_b}{N_0}}) \tag{2.27}$$

onde

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-\frac{y^2}{2}} dy \tag{2.28}$$

onde R_c é a taxa do código (veja equação (2.30)) e E_b/N_0 é a razão sinal ruido (com E_b a energia por bit de informação e N_0 a densidade espectral do ruido). A decodificação por decisão abrupta tem a vantagem que ao trabalhar só com um número finito de valores, o cálculo é feito em ponto fixo, gerando algoritmos de decodificação mais rápidos em tempo de execução. A decisão abrupta feita faz que muita da informação necessária para uma decodificação se perca. É pois ao final uma escolha entre decisão suave ou abrupta, um compromisso entre convergência e velocidade, ou quantidade de informação e complexidade no cálculo.

2.4 Codificação

Como o canal de comunicação esta introduzindo continuamente erros aos dados enviados através dele, é necessário proteger a informação com a intenção de diminuir na medida do possível a taxa de erro do bit ou BER, (do imglês "Bit Error Rate"). Para este efeito, o codificador de canal introduz bits de redundância. Esses bits adicionais permitem detectar e corrigir alguns dos bits com erro. Assim a eficiência na diminuição do BER dependerá da capacidade do canal, da quantidade e do método de criação dos bits de redundância e do método de decodificação dos dados.

Existem basicamente dois tipos de codificação do canal: códigos de bloco e códigos convolucionais. No capitulo 3 se falará de forma mais extensa dos códigos de bloco.

Neste momento é necessário entender o uso do codificador de canal. Para esta tese se trabalhará com dados binários, então o codificador receberá um vetor U de dados binários com comprimento K, e após acrescentar a redundância devolverá outro vetor V de dados binários de comprimento N. A taxa de informação do código é dada por

$$K < N \tag{2.29}$$

$$R_c = \frac{K}{N}. \tag{2.30}$$

Se consideramos que os bits recebidos no vetor U tem a máxima entropia, então a equação (2.30) nos diz que o vetor V terá R_c bits de informação por bit enviado. Juntando esta idéia com a capacidade do canal, pode-se considerar a taxa como a máxima quantidade de informação por bit e igualá-lo com a capacidade do canal que é a máxima quantidade de informação que é possível enviar (ver Figura 2.7). Se

$$R_c \leq C, \tag{2.31}$$

Então é possível recuperar toda a informação enviada pelo canal.

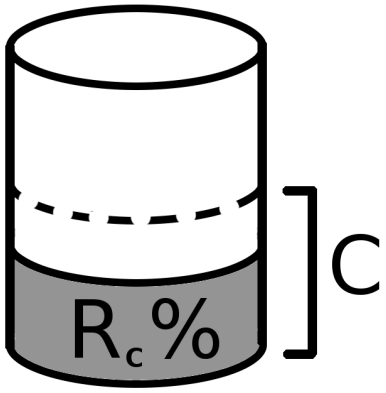

Fig. 2.7: Relação entre taxa e capacidade, o cilindro representa um bit.

Aplicando esta idéia ao canal BSC obtemos que o límite máximo da probabilidade de erro do canal (p_{max}), que permite ter a possibilidade de recuperar a totalidade dos bits enviados, está dado pelas seguintes equações:

$$R_c \leq 1 - H(p) \tag{2.32}$$

$$p_{max} = H^{-1}(1 - R_c). \tag{2.33}$$

Aplicando o mesmo critério ao canal $AWGN$ obtemos que o valor minimo de $SNR = S/N$ (do inglês Signal to Noise Ratio) para ter a possibilidade de recuperar a totalidade dos bits enviados é SNR_{min}.

$$R_c \leq \frac{1}{2}\, log_2(1 + \frac{S}{N}) \tag{2.34}$$

$$2^{2R_c} - 1 = SNR_{min} \tag{2.35}$$

Capítulo 3

Códigos corretores de erro

No capítulo 2 se viu a necessidade de proteger os dados quando estes transitam num canal ruidoso. No gráfico da Figura 3.1 pode-se ver que o codificador de canal é o encarregado de acrescentar redundância à informação enviada com a intenção de melhorar a possibilidade de incrementar a taxa do erro de bits.

O código corretor de erro usado transformará o conjunto de dados binários de K bits num subconjunto do conjunto de todos os possíveis dados de N bits. Para este efeito pode-se fazer uso dos códigos de bloco.

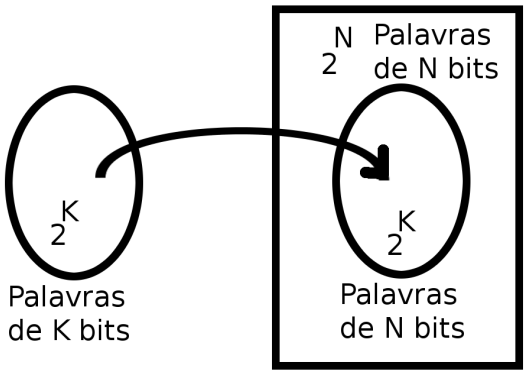

Fig. 3.1: Mapeamento feito pelo codificador de canal nas palavras código.

3.1 Códigos de bloco

Um código de bloco (N, K) associa uma palavra U onde $u_i \in GF(2)$ e $0 \leq i < K$, a uma palavra código V onde $v_i \in GF(2)$ e $0 \leq i < N$. O corpo de Galois binário $GF(2)$ é definido no apêndice A.

Para códigos de bloco lineares, esta associação é feita através da matriz geradora, G. Considere as palavras

$$U = \begin{pmatrix} u_0 & u_1 & u_2 & ... & u_{K-1} \end{pmatrix}, \tag{3.1}$$

e

$$V = \begin{pmatrix} v_0 & v_1 & v_2 & ... & v_{N-1} \end{pmatrix}. \tag{3.2}$$

Então,

$$V = U\,G, \tag{3.3}$$

onde

$$G = \begin{pmatrix} \mathbf{g_0} \\ \mathbf{g_1} \\ \mathbf{g_2} \\ \vdots \\ \mathbf{g_{K-1}} \end{pmatrix} = \begin{pmatrix} g_{00} & g_{01} & g_{02} & \cdots & g_{0(N-1)} \\ g_{10} & g_{11} & g_{12} & \cdots & g_{1(N-1)} \\ g_{20} & g_{21} & g_{22} & \cdots & g_{2(N-1)} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ g_{(K-1)0} & g_{(K-1)1} & g_{(K-1)2} & \cdots & g_{(K-1)(N-1)} \end{pmatrix} \tag{3.4}$$

Na matriz geradora G, cada elemento $g_{ij} \in GF(2)$, $0 \leq i < K$ e $0 \leq j < N$, e cada uma de suas linhas são $N - uplas$ linearmente independentes. O vetor V é um elemento no espaço vetorial formado pelos vetores g_i, $0 \leq i < K$, isto é,

$$V = u_0\mathbf{g_0} + u_1\mathbf{g_1} + u_2\mathbf{g_2} + ... + u_{K-1}\mathbf{g_{K-1}}. \tag{3.5}$$

O conjunto de todos os pontos V formam um subespaço vetorial de dimensão K do espaço vetorial de dimensão N formado por todas as $N - uplas$ com elementos em $GF(2)$, também chamado $EG(N, 2)$ (Geometria Euclidiana $N - dimensional$ em $GF(2)$).

Dado que os vetores $\mathbf{g_i}$ são linearmente independentes, se tem uma relação unívoca entre um ponto U e um ponto V, pelo qual é possível a decodificação do V para obter novamente U. O hiperplano $F = \{V : V = UG\}$ de dimensão K, tem em $EG(N, 2)$, $N - K$ vetores linearmente independentes ortogonais a cada ponto do plano F, isto é, cada palavra código V e os vetores $\mathbf{g_i}$ são ortogonais a estes $N-K$ vetores. Se chamará de $h_j \in EG(N, 2)$ a cada um desses vetores, onde $0 \leq j < N-K$. Com eles se constrói a matriz H,

$$H = \begin{pmatrix} \mathbf{h_0} \\ \mathbf{h_1} \\ \mathbf{h_2} \\ \vdots \\ \mathbf{h_{N-K-1}} \end{pmatrix} = \begin{pmatrix} h_{00} & h_{01} & h_{02} & \dots & h_{0(N-1)} \\ h_{10} & h_{11} & h_{12} & \dots & h_{1(N-1)} \\ h_{20} & h_{21} & H_{22} & \dots & h_{2(N-1)} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ h_{(N-K-1)0} & h_{(N-K-1)1} & h_{(N-K-1)2} & \dots & h_{(N-K-1)(N-1)} \end{pmatrix} \tag{3.6}$$

Como a matriz H de $N-K$ linhas e N colunas é ortogonal a V e G, H pode ser usado como uma medida de verificação de que um ponto R qualquer pertence ou não ao subespaço F. Primeiro calcula-se S,

$$RH^t = S. \tag{3.7}$$

Se $S = 0$ então R pertence a F. Também é possível achar H a partir de G, procurando seu espaço nulo, como segue,

$$GH^t = 0 \tag{3.8}$$

Nas equações (3.7) e (3.8) o símbolo t representa a transposta da matriz. Neste ponto só resta procurar algum método para obter o vetor U a partir de V, a forma mais fácil é aplicando força bruta e gerar uma tabela de equivalências entre U e V. O problema inicia quando o vetor V percorre um canal ruidoso, na sua saída este canal entrega um vetor com erro que chamaremos de R. Dado que R difere de V, já não é possível obter o vetor de informação U a partir dele. A solução é comparar o vetor recebido R com todos os vetores V da tabela e escolher como vetor enviado no canal o vetor que gere a distancia euclidiana mínima, e logo escolher a partir dele a estimativa do possível vetor vetor de informação U'. O problema deste método é o excessivo tempo de cálculo empregado, pelo cual geralmente se busca gerar uma matriz G de tal forma que seja mais fácil a obtenção de U' a partir R. É possível selecionar o vetor V mais provável, a partir do vetor R, fazendo uso dos bits de redundância, isto será estudado no capitulo 4.

3.1.1 Códigos LDPC

Os códigos $LDPC$, do inglês "Low-Density Parity-Check", foram publicados por Gallager [1] em 1963, mas eles não receberam muita atenção até o redescobrimento por MacKay [6] em 1999. Estes códigos são caracterizados por terem matrizes de verificação de paridade (H) com muitos "0"s e poucos "1"s na sua composição. Outra característica de um código (N, d_v, d_c) $LDPC$, é que tem

uma quantidade d_v fixa de "1"s por coluna e uma quantidade d_c fixa de "1"s por linha para uma quantidade N de bits codificados. É fácil notar que ao considerar fixa a quantidade de "1"s por linha e coluna, pode-se calcular a quantidade total de uns na matriz H das seguintes formas:

$$d_v N = d_c(N - K). \tag{3.9}$$

Da equação anterior se deduz a seguinte relação, chamada de densidade (γ) da matriz, onde $\gamma \ll 1$,

$$\gamma = \frac{d_v}{N - K} = \frac{d_c}{N}. \tag{3.10}$$

Além disso, também pode-se expressar a taxa do código (R_c) com a seguinte equação,

$$R_c = 1 - \frac{d_v}{d_c} = 1 - \frac{N - K}{N} = \frac{K}{N}. \tag{3.11}$$

3.1.2 Forma sistemática

Uma forma de construir a matriz G, que facilita a obtenção do vetor U a partir do vetor V, é fazendo com que dentro da palavra código enviada esteja incorporado o vetor de informação U, e o restante da palavra será completada com os bits de paridade como mostra a Figura 3.2.

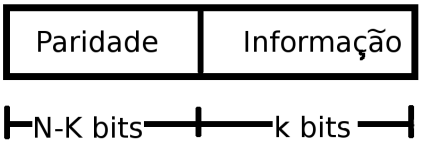

Fig. 3.2: Código sistemático.

Para fazer isto, se faz uso de uma matriz identidade de dimensão K (I_K) e uma matriz de paridade de K linhas e $N - K$ colunas ($P_{K(N-K)}$), dado por

$$P = \begin{pmatrix} p_{00} & p_{01} & p_{02} & \cdots & p_{0(N-K-1)} \\ p_{10} & p_{11} & p_{12} & \cdots & p_{1(N-K-1)} \\ p_{20} & p_{21} & p_{22} & \cdots & p_{2(N-K-1)} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ p_{(K-1)0} & p_{(K-1)1} & p_{(K-1)2} & \cdots & p_{(K-1)(N-K-1)} \end{pmatrix} \tag{3.12}$$

Assim, escrevemos G como

$$G = \begin{pmatrix} \mathbf{g_0} \\ \mathbf{g_1} \\ \mathbf{g_2} \\ \vdots \\ \mathbf{g_{K-1}} \end{pmatrix} = \begin{pmatrix} p_{00} & p_{01} & p_{02} & \cdots & p_{0(N-K-1)} & 1 & 0 & 0 & \dots & 0 \\ p_{10} & p_{11} & p_{12} & \cdots & p_{1(N-K-1)} & 0 & 1 & 0 & \dots & 0 \\ p_{20} & p_{21} & p_{22} & \cdots & p_{2(N-K-1)} & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ p_{(K-1)0} & p_{(K-1)1} & p_{(K-1)2} & \cdots & p_{(K-1)(N-K-1)} & 0 & 0 & 0 & \dots & 1 \end{pmatrix}, \tag{3.13}$$

$$G = [P\ I_K]. \tag{3.14}$$

Para obter a palavra código V usa-se a equação (3.3). Note-se que com essa disposição na matriz G o vetor V tem a forma da Figura 3.2, com os bits de informação colocados à direita do vetor linha. Outra vantagem de construir a matriz G desta maneira é que é fácil obter a matriz H, que é ortogonal a G (ver equação (3.8)). H será dada por

$$H = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & p_{00} & p_{10} & p_{20} & \cdots & p_{(K-1)0} \\ 0 & 1 & 0 & \dots & 0 & p_{01} & p_{11} & p_{21} & \cdots & p_{(K-1)1} \\ 0 & 0 & 1 & \dots & 0 & p_{02} & p_{12} & p_{22} & \cdots & p_{(K-1)2} \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & p_{0(N-K-1)} & p_{1(N-K-1)} & p_{2(N-K-1)} & \cdots & p_{(K-1)(N-K-1)} \end{pmatrix} \tag{3.15}$$

$$H = [I_{N-K}\ P^t] \tag{3.16}$$

3.1.3 Matrizes regulares e irregulares

Um ponto interessante a olhar é o desempenho das matrizes de baixa densidade quando se conserva fixo ou variável o número de "1"s por linha e coluna. Na seção 3.1.1 viu-se o caso das matrizes $LDPC$ onde a matriz H tem um número $regular$ fixo de uns por linha e coluna. Mas o fato de ter uma matriz H regular não garante que a matriz G seja regular. Para ver isto se tem que lembrar que H é uma matriz formada por $N - K$ vetores linearmente independentes que formam um hiperplano. Este hiperplano é ortogonal aos K vetores linearmente independentes que formam a matriz G. Dado que estes K vetores também formam um hiperplano, é possível escolher uma combinação linear qualquer desses vetores, de forma que se obtém um novo conjunto de K vetores linearmente independentes. Nessa infinidade de vetores possíveis, estão os casos em que os vetores escolhidos para formar a matriz G constituem matrizes regulares, sistemáticas ou simplesmente irregulares.

Pelo visto até agora na literatura pode-se ter os seguintes tipos de matrizes: Uma matriz $LDPC$ definida por Gallager [1] onde H tem um número fixo de uns por linha e coluna. Esta matriz é chamada de $regular$. O método de geração e o tipo de matriz geradora (G), para estas matrizes, dependerá do método empregado para gerar H. Um segundo tipo de matriz é a baseada no que foi visto na seção 3.1.2 acrescentando a isto a idéia de que além de ser uma matriz sistemática, P tem que ser uma matriz regular e de baixa densidade. Quando as matrizes geradoras e de paridade cumprem estas características se diz que se tem uma matriz $LDPC$ irregular e sistemática ou simplesmente $LDGM$ [7] [8], do inglês $Low\ Density\ Generator\ Matrix$.

Também é possível ter variações das matrizes $LDPC$ e $LDGM$, como as $LDPC$ irregulares e as $LDGM$ irregulares na sua matriz P. Geralmente estas irregularidades na quantidade de uns por linha e coluna não são muito abruptas, e tendem a oscilar num intervalo estreito de valores, pois uma desigualdade extrema nestes valores outorgaria maior proteção para uns bits que a outros. Isto é fácil de olhar se estudamos o caso das matrizes $LDGM$, dado que estas tem um irregularidade abrupta entre seus bits de paridade e informação. Por exemplo a matriz G

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}, \tag{3.17}$$

é uma matriz geradora sistemática e com matriz P irregular, que gera palavras código V de comprimento $N = 7$ com $K = 4$ bits de informação e $N - K = 3$ bits de paridade. Pode-se ver que com uma matriz deste tipo os bits de informação u_i ficam codificados no vetor V como mostra a Figura 3.3.

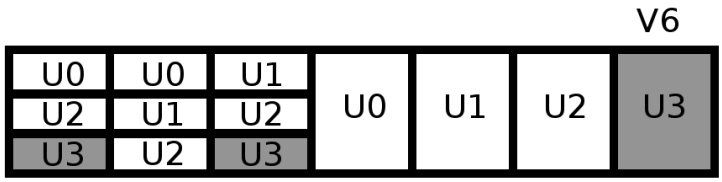

Fig. 3.3: Palavra código V formada por uma matriz G sistemática (7,4) com matriz P irregular.

As equações de paridade que definem a palavra código são:

$$\begin{aligned} v_0 &= u_0 \oplus u_2 \oplus u_3 \\ v_1 &= u_0 \oplus u_1 \oplus u_2 \\ v_2 &= u_1 \oplus u_2 \oplus u_3 \\ v_3 &= u_0 \\ v_4 &= u_1 \\ v_5 &= u_2 \\ v_6 &= u_3 \end{aligned} \quad . \tag{3.18}$$

Os bits u_i estão espalhados desigualmente em V, provocando que um erro em algum bit de V tenha maior ou menor repercussão na decodificação pra recuperar o vetor U, por exemplo de (3.18) obtemos 3 equações que contem informação do bit u_3,

$$v_0 \oplus v_3 \oplus v_5 = u_3, \tag{3.19}$$

$$v_2 \oplus v_4 \oplus v_5 = u_3 \tag{3.20}$$

$$v_6 = u_3. \tag{3.21}$$

É evidente que se tivéssemos o caso em que o bit u_3 em (3.21) fora diferente dos demais, a maior desconfiança recairia no bit v_6, dado que um erro de um bit (v_6), é mais provável que dois erros em (3.19) e (3.20), ainda assim é possível que os bits errados estejam em essas equações, dado que todos os bits de V tem a mesma probabilidade de erro. Por isso é interessante ter uma distribuição homogênea de U em V para que todos os bits tenham a mesma oportunidade de ser decodificados corretamente. Ainda assim concentrar a distribuição da informação em uns bits mais que em outros pode ajudar quando a quantidade de bits errados é baixa, para este caso quando só se erra um bit, mas se tivéssemos uma probabilidade alta de erro de bit, é muito provável ter errado na decisão. Lembrando que a matriz G esta formada por um conjunto de vetores linearmente independentes, é possível então aplicar uma transformação linear e obter outro conjunto de vetores linearmente independentes com uma distribuição de U em V mais homogênea.

Considerando (3.23) obtemos G',

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \tag{3.22}$$

$$G' = A\,G, \tag{3.23}$$

que também é ortogonal a H.

$$G' = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{3.24}$$

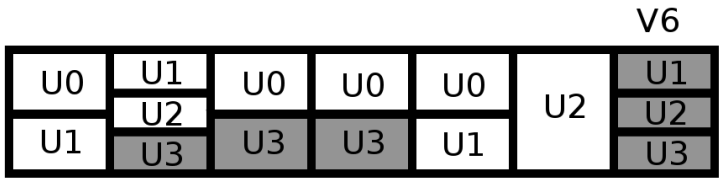

Fig. 3.4: Palavra código V formada por uma matriz G' (7,4) irregular.

Em este caso uma decodificação como em (3.19), (3.20) e (3.21) é mais difícil, em geral é muito difícil decodificar desse jeito, o comum é usar a matriz H como se verá no capítulo 4, onde na escolha da matriz H se aplica um critério similar para a escolha entre matrizes H sistemáticas, regulares e irregulares.

A dificuldade de uso de uma matriz irregular reside no fato que as operações com as matrizes geradoras e de paridade tendem a ser mais difíceis de trabalhar quando usamos estas matrizes dado que a irregularidade na sua topologia dificulta sua implementação física. Ainda assim as matrizes irregulares tem demostrado ter um melhor desempenho quanto comparado ao desempenhos das regulares.

3.1.4 Gráficos de Tanner

Tanner introduziu uma forma gráfica e eficiente de representar os códigos $LDPC$. Um gráfico de Tanner tem dois tipos de nó, nós de variável representados por círculos e nós de verificação de paridade representados por quadrados. Um exemplo disto é a Figura 3.5. Nos gráficos de Tanner os nós de paridade só estão ligados aos nós de verificação, mas não existe ligação nenhuma entre nós do mesmo tipo. Por esta característica se diz que os gráficos de Tanner são bipartidos, em outras palavras os nós estão divididos em dois conjuntos só ligados entre sim por nós de distinto tipo. A Figura 3.5 representa o gráfico de Tanner da matriz de verificação de paridade

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}. \tag{3.25}$$

O gráfico representa as ligações entre V e S ($S = VH^t$) seguindo as seguintes equações,

$$s_2 = v_2 \oplus v_4 \oplus v_5 \oplus v_6, \tag{3.26}$$

$$s_1 = v_1 \oplus v_3 \oplus v_4 \oplus v_5, \tag{3.27}$$

$$s_0 = v_0 \oplus v_3 \oplus v_5 \oplus v_6, \tag{3.28}$$

Elas mostram como cada um dos bits de verificação s_i estão ligados aos nós de variável v_i. Estas ligações estão representadas na Figura 3.5. Alem disso, o gráfico também mostra como cada um dos bits de verificação tem 4 ligações, ou seja tem um d_c regular, mas o número de ligações de cada nó de variável d_v é irregular.

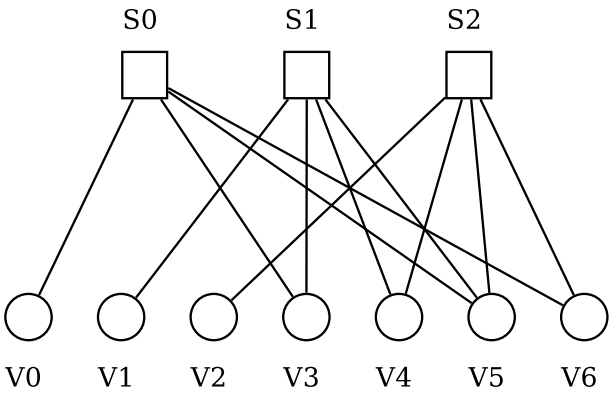

Fig. 3.5: Gráfico de Tanner correspondente à matriz de verificação de paridade em (3.25).

3.2 Geometria finita

Uma geometria finita é uma geometria com um número finito de pontos e por tanto um número finito de linhas e planos. Sua importância no estudo dos códigos de bloco é por causa do fato que é possível usar suas características para construir uma matriz com um número regular de 1's por linha e colunas, alem de se poder controlar a quantidade de 1's das mesmas.

DEFINIÇÃO 3.2.1 *Uma **operação binária** ou 2-ária (operação com dois operandos) definida num conjunto $\wp$ é uma função que trabalha sobre dois operandos que pertencem a $\wp$, e cujo resultado pertence também ao conjunto $\wp$. Para os operandos $a \in \wp$ e $b \in \wp$ se obtém um $c \in \wp$ aplicando a eles o operador $*$.*

$$\begin{aligned} * : \wp \times \wp &\to \wp \\ (a, b) &\to *(a, b) \end{aligned} \tag{3.29}$$

Na equação 3.29 o símbolo $\times$ é o produto cartesiano. Um ponto interessante é que $*(a, b)$ geralmente é representado como $a * b$.

3.2.1 Grupos finitos

Os grupos finitos são um caso particular da definição de grupo quando este tem um número finito de elementos. Um grupo está definido como:

DEFINIÇÃO 3.2.2 *Um conjunto $\wp$ onde uma operação binária $*$ é definida, é chamado de $grupo$ se as seguintes condições foram satisfeitas para a, b e c que pertencem a $\wp$.*

i) *A operação binária $*$ é associativa, quer dizer: $a * (b * c) = (a * b) * c$.*

ii) *Existe um elemento identidade e onde $a * e = e * a = a$, $\forall a$.*

iii) *Para a $\in \wp$ existe um elemento inverso a' tal que $a * a' = a' * a = e$.*

Se alem de i, ii e iii. , temos

iv) *A operação binária $*$ é comutativa: $a * b = b * a$.*

Se diz que é um grupo comutativo ou Abeliano.

O ordem de um grupo é o número de elementos que tem o conjunto que o forma. O grupo do Exemplo 3.2.1 tem ordem 2, dado que a ordem deste grupo é finita então se diz que ele é um grupo comutativo finito.

EXEMPLO 3.2.1 *Dado o conjunto finito $\wp = \{0, 1\}$ e a operação binária $\oplus$ (OR exclusivo) obtemos a Tabela 3.1.*

Olhando a Tabela 3.1 vemos que cumpre os pontos (i), (ii), (iii) e (iv) da Definição 3.2.2 pelo qual o conjunto $\wp$ e o operador $\oplus$ formam um grupo finito comutativo.

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

Tab. 3.1: Operação binária OR exclusivo entre a e b.

a	b	$a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

Tab. 3.2: Operação binária AND entre a e b.

EXEMPLO 3.2.2 *Dado o conjunto finito $\wp = \{0, 1\}$ e a operação binária $\cdot$ (AND) obtemos a Tabela 3.2.*

Olhando a Tabela 3.2 vemos que não cumpre a condição (iii) pelo qual o conjunto $\wp$ e o operador $\cdot$ não formam um grupo.

Um tema interessante a tratar são os grupos gerados usando aritmética modular ou também chamada aritmética do relógio. Esta aritmética envolve o conceito de congruência entre dois números, que é o nome que se dá quando dois números tem o mesmo resto ao serem divididos por um terceiro, este último número é chamado de módulo. Assim, os números 1 e 6 módulo 5 são congruentes dado que o resto é 1 em ambos casos. Expandindo esta ideia pode-se fazer também operações binárias aplicando um módulo "m". A soma $2 + 4$ em módulo 5 gera o número 1. Os Exemplos 3.2.1 e 3.2.2 são operações de soma e multiplicação em módulo $p = 2$.

EXEMPLO 3.2.3 *Dado o conjunto finito $\wp = \{0, 1, 2, 3, 4\}$ e uma operação binária $\boxplus$ que é a soma em módulo $p = 5$, obtemos a Tabela 3.3, nela vemos que cumpre os pontos (i), (ii), (iii) e (iv) da Definição 3.2.2 pelo qual o conjunto $\wp$ e o operador $\boxplus$ formam um grupo finito comutativo.*

EXEMPLO 3.2.4 *Dado o conjunto finito $\wp = \{1, 2, 3, 4\}$ e uma operação binária $\boxdot$ que é a multiplicação em módulo $p = 5$, obtemos a Tabela 3.4. É importante ressaltar que o elemento 0 foi retirado do conjunto pois como se viu no Exemplo 3.2.2 evita que o ponto (iii) da Definição 3.2.2 seja verdadeiro, olhando a Tabela 3.4 vemos que cumpre todos os pontos da Definição 3.2.2 pelo qual o conjunto $\wp$ e o operador $\boxdot$ formam um grupo finito comutativo.*

$\boxplus$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Tab. 3.3: Operação binária soma $\boxplus$ em módulo 5 sobre o conjunto $\wp = \{0, 1, 2, 3, 4\}$.

$\boxdot$	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

Tab. 3.4: Operação binária multiplicação $\boxdot$ em módulo 5 sobre o conjunto $\wp = \{1, 2, 3, 4\}$.

Do Exemplo 3.2.4 podemos concluir que a operação binária de multiplicação em módulo p tem problemas com o elemento 0 pelo qual é retirado do conjunto. Ainda assim, se nós escolhemos um número p que não seja primo teremos problemas devido a que na tabela gerada existe a possibilidade de aparecer o número 0 como se vê no Exemplo 3.2.5

EXEMPLO 3.2.5 *Dado o conjunto finito $\wp = \{1, 2, 3, 4, 5\}$ e uma operação binária $\boxdot$ que é a multiplicação em módulo $p = 6$, obtemos a Tabela 3.5, nela vemos que não cumpre o ponto (iii) da Definição 3.2.2 pelo qual o conjunto $\wp$ e o operador $\boxdot$ não geram um grupo.*

$\boxdot$	1	2	3	4	5
1	1	2	3	4	5
2	2	4	0	2	4
3	3	0	3	0	3
4	4	2	0	4	2
5	5	4	3	2	1

Tab. 3.5: Operação binária multiplicação $\boxdot$ em módulo 6 sobre o conjunto $\wp = \{1, 2, 3, 4, 5\}$.

3.2.2 Geometria Euclidiana

Dado que a geometria euclidiana [5],[9] estuda as propriedades dos pontos, linhas e planos em um espaço de dimensão m, este conceito pode ser tomado para trabalhar com espaços onde as m-

uplas $(a_0, a_1, a_2, \cdots, a_{m-1})$ contém componentes $a_i \in GF(2^s)$. Cada m-upla é chamada de ponto em $EG(m, 2^s)$, onde EG vem do inglês $Euclidean\ Geometry$. Assim teríamos um espaço de dimensão m com um número fixo de pontos como pode-se ver na Figura 3.6, onde se trabalha com uma geometria de dimensão 3 com eixos em $GF(2^2)$, isto é uma geometria com pontos em $EG(3, 2^2)$.

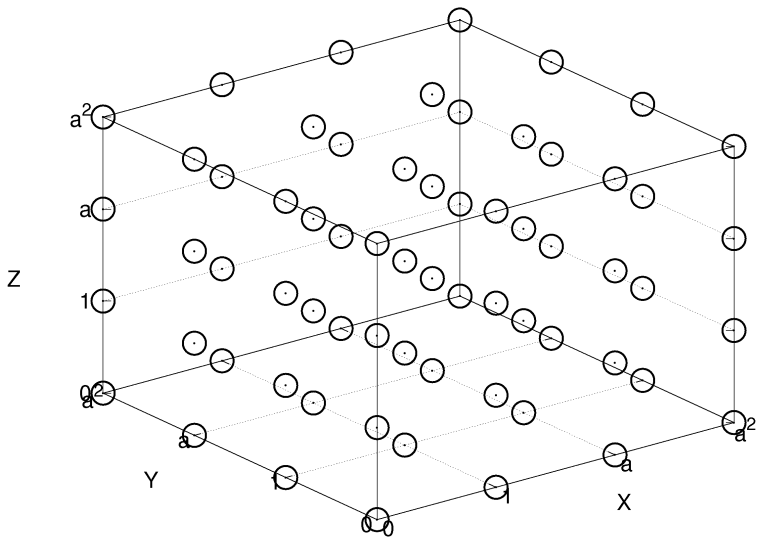

Fig. 3.6: Pontos no espaço de dimensão 3 com elementos em $GF(2^2)$.

A quantidade de pontos em $EG(m, 2^s)$ é fácil de calcular: basta multiplicar a quantidade de pontos por eixo (2^s), um número de vezes (m) de dimensões da geometria, isto é,

$$Pontos_{EG(m,2^s)} = 2^{ms}. \tag{3.30}$$

Para o caso da Figura 3.6 se tem 64 pontos em $EG(3, 2^2)$. Uma linha em $EG(m, 2^s)$ é definida por:

$$L = \{A_0 + b\, A_1 : A_0, A_1 \in EG(m, 2^s) \wedge A_1 \neq 0 \wedge b \in GF(2^s)\}, \tag{3.31}$$

onde A_0 é um ponto de referência da linha, A_1 é um vetor de direção e b é um escalar que multiplica o vetor A_1. Dado que b pertence a $GF(2^s)$ ele só pode assumir 2^s valores. Então, a quantidade de pontos por cada linha em $EG(m, 2^s)$ fica definido pela equação,

$$PontosPorLinha = 2^s. \tag{3.32}$$

Por exemplo para $EG(3, 2^2)$ com um $A_0 = (0, 1, 1)$ e um $A_1 = (1, 1, 1)$ uma linha L_0 teria 4 pontos,

$$L_0 : (0,1,1) + \left\{ \begin{array}{c} 0 \\ 1 \\ \alpha \\ \alpha^2 \end{array} \right\} (1,1,1) = \left\{ \begin{array}{c} (0,1,1) \\ (1,0,0) \\ (\alpha, 1+\alpha, 1+\alpha) \\ (\alpha^2, 1+\alpha^2, 1+\alpha^2) \end{array} \right\} \tag{3.33}$$

Aplicando operações em módulo $p(x)$, com um $p(x)$ como na Tabela A.3 obtemos (3.34).

$$L_0 : \left\{ \begin{array}{c} (0,1,1) \\ (1,0,0) \\ (\alpha, 1+\alpha, 1+\alpha) \\ (\alpha^2, 1+\alpha^2, 1+\alpha^2) \end{array} \right\} \xrightarrow{p(x)} \left\{ \begin{array}{c} (0,1,1) \\ (1,0,0) \\ (\alpha, \alpha^2, \alpha^2) \\ (\alpha^2, \alpha, \alpha) \end{array} \right\} \tag{3.34}$$

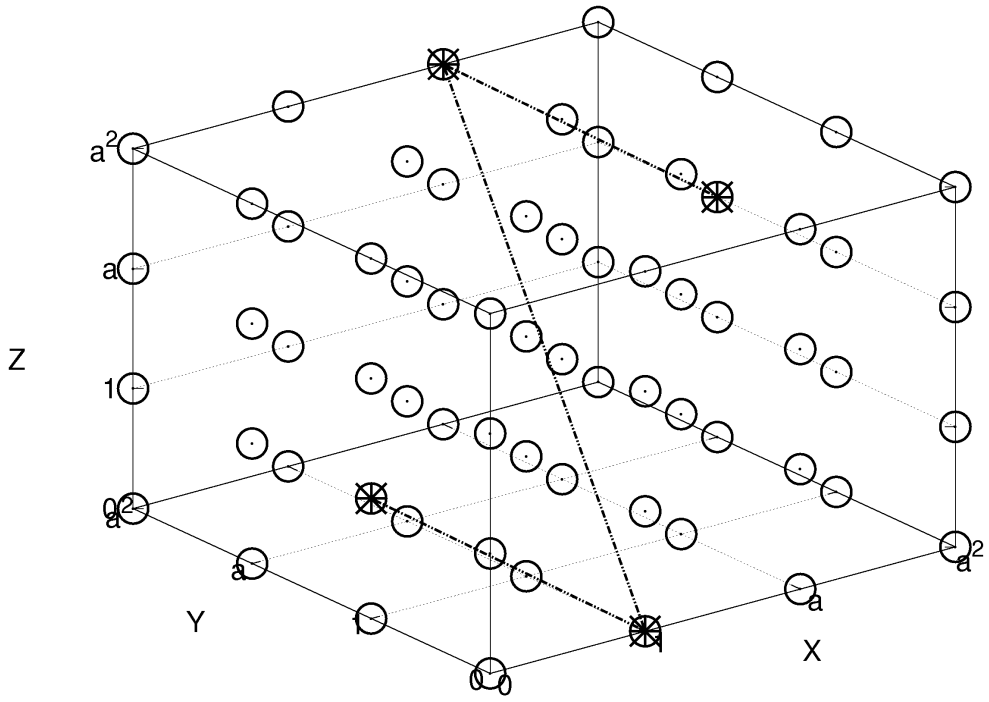

Fig. 3.7: Linha L_0 com pontos em $EG(3, 2^2)$.

Na geometria euclidiana pode-se definir as linhas paralelas. Duas linhas são paralelas se compartilham a mesma direção mas não tem nenhum ponto em comum, isto é, não tem pontos de intersecção. Para uma linha como a definida em (3.31), ela tem uma quantidade de linhas paralelas igual ao número de pontos A_0 distintos. Dado que A_0 é um ponto em $EG(m, 2^s)$, A_0 pode tomar 2^{ms} valores distintos. Mas alguns dos valores de A_0 formam a mesma linha. Como cada linha contem 2^s pontos, então, a quantidade de linhas paralelas entre si é como segue

$$LinhasParalelas = \frac{2^{ms}}{2^s} = 2^{(m-1)s} \tag{3.35}$$

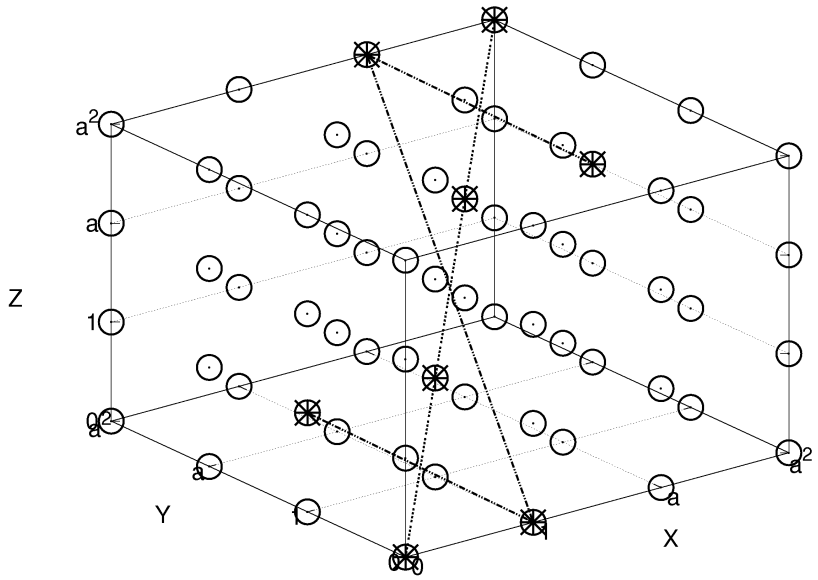

Fig. 3.8: Duas linhas paralelas $T_0 : \{(0,1,1) + b(1,1,1)\}$ e $T_1 : \{(0,0,0) + b(1,1,1)\}$ com $b \in GF(2^2)$ numa $EG(3, 2^2)$.

Também é possível calcular a quantidade de linhas que passam por um ponto A_0 qualquer. Da equação da linha (3.31), dado que bA_1 pode tomar todos os vetores de $EG(m, 2^s)$ menos o origem $(0, 0, \cdots, 0)$, temos $2^{ms} - 1$ valores distintos. E conhecendo o fato de que muitos desses vetores formam a mesma linha, e cada linha tem 2^s pontos menos 1, dado que não pegamos o vetor gerado para $b = 0$, obtemos a equação.

$$LinhasPorPonto = \frac{2^{ms} - 1}{2^s - 1}. \tag{3.36}$$

Para calcular o número total de linhas possíveis basta multiplicar a quantidades de linhas que passam por um ponto qualquer pela quantidades de linhas paralelas que existem entre si, como mostra a equação

$$\begin{aligned} LinhasTotais &= \{LinhasPorPonto\}\{LinhasParalelas\} \\ &= \frac{(2^{ms}-1)2^{(m-1)s}}{2^s-1}. \end{aligned} \tag{3.37}$$

Também é importante calcular a quantidade de linhas que não passam pelo origem de coordenadas. Para isto basta diminuir a quantidade de linhas por pontos da quantidade de linhas totais

$$\begin{aligned} LinhasTotaisSemOrigem &= \{LinhasTotais\} - \{LinhasPorPonto\} \\ &= \frac{(2^{ms}-1)2^{(m-1)s}}{2^s-1} - \frac{(2^{ms}-1)}{2^s-1} \quad . \\ &= \frac{(2^{ms}-1)(2^{(m-1)s}-1)}{2^s-1} \end{aligned} \tag{3.38}$$

A quantidade de pontos totais em $EG(m, 2^s)$ descontando a origem $(0, 0, \cdots, 0)$ é dada por

$$PontosTotaisSemOrigem = 2^{ms} - 1 \tag{3.39}$$

A quantidade de linhas que passam por um ponto $A_0 \in EG(m, 2^s)$ descontando a linha que passa pela origem, está definida na equação abaixo

$$LinhasPorPontoSemOrigem = \frac{2^{ms}-1}{2^s-1} - 1. \tag{3.40}$$

Um tema importante é a relação que existe entre o campo de Galois $GF(2^{ms})$ e a geometria euclidiana $EG(m, 2^s)$. Como exemplo podemos ver a Tabela A.4 onde cada elemento de um campo de Galois $GF(2^4)$ com um polinômio primitivo de ordem 4, $p(x) = x^4 + x + 1$ pode ser representado como um polinômio de grau 4 com coeficientes do polinômio em $GF(2)$, também pode ser representado como uma $4 - upla$ com elementos em $GF(2)$ isto é elementos em $EG(4, 2)$. Seguindo a mesma linha de pensamento podemos ver que cada elemento α^i de um campo de Galois $GF(2^{sm})$ com um polinômio primitivo $p(x)$ de grau ms pode ser representado em um polinômio de grau m com coeficientes a_{ij} em $GF(2^s)$, $i \leq 2^{ms} - 2$, como se vê nas equações abaixo

$$\alpha^i = a_{i0} + a_{i1}\alpha + a_{i2}\alpha^2 + a_{i3}\alpha^3 + \cdots + a_{i(m-1)}\alpha^{m-1} \tag{3.41}$$

$$GF(2^s) = \left\{0, 1, \beta, \beta^2, \beta^3, \cdots, \beta^{2^s-2}\right\} \tag{3.42}$$

$$\beta^{2^s-1} = \alpha^{2^{ms}-1} = 1 \tag{3.43}$$

Também pode ser representado como uma $m - upla$ com elementos em $EG(m, 2^s)$.

$$\alpha^i = (a_{i0}, a_{i1}, a_{i2}, a_{i3}, \cdots, a_{i(m-1)}) \tag{3.44}$$

EXEMPLO 3.2.6 *Um campo de Galois $GF(2^4)$ pode ser representando numa $EG(2, 2^2)$ com um*

$m = 2$ *e* $s = 2$*. O campo* $GF(2^4)$ *usa como polinômio primitivo* $p(x) = x^4 + x + 1$*, cada elemento* $\alpha^i \in GF(2^4)$ *pode ser representando mediante uma* $2 - upla$ *com elementos em* $GF(2^2)$*. O campo em* $GF(2^2)$ *contem os seguintes elementos* $\{0, 1, \beta, \beta^2\}$*. Usando a equação (3.43) obtemos* $\beta^3 = \alpha^{15}$ *pelo qual* $\beta = \alpha^5$*,* $\alpha^i = b_{i0} + b_{i1}\alpha,\ b_{ij} \in GF(2^2),\ 0 \leq i \leq 2$*.*

Utilizando $p(\alpha) = 0$ *obtemos que* $\alpha^4 = \alpha + 1$ *e* $\beta = \alpha^2 + \alpha$ *e geramos a Tabela (3.6).*

Representação em potências	Representação polinomial	Representação vetorial
0	0	$(0, 0)$
1	1	$(1, 0)$
α	α	$(0, 1)$
α^2	$\beta + \alpha$	$(\beta, 1)$
α^3	$\beta + \beta^2\alpha$	(β, β^2)
α^4	1+α	$(1, 1)$
α^5	β	$(\beta, 0)$
α^6	$\beta\alpha$	$(0, \beta)$
α^7	$\beta^2 + \beta\alpha$	(β^2, β)
α^8	$\beta^2 + \alpha$	$(\beta^2, 1)$
α^9	$\beta + \beta\alpha$	(β, β)
α^{10}	β^2	$(\beta^2, 0)$
α^{11}	$\beta^2\alpha$	$(0, \beta^2)$
α^{12}	$1 + \beta^2\alpha$	$(1, \beta^2)$
α^{13}	$1 + \beta\alpha$	$(1, \beta)$
α^{14}	$\beta^2 + \beta^2\alpha$	(β^2, β^2)

Tab. 3.6: Representação de $GF(2^4)$ com $p(x) = x^4 + x + 1$ em uma $EG(2, 2^2)$.

EXEMPLO 3.2.7 *Usando a Tabela 3.6 é possível calcular as linhas em* $EG(2, 2^2), p(x) = x^4 + x + 1, \beta = \alpha^5, m = 2, s = 2$*, que passam pelo ponto* $A_0 = \alpha^3 = (\beta, \beta^2)$*. Aplicando a equação 3.36 vemos que existem* 5 *linhas que passam por* A_0*, seguindo a definição de linha de (3.31) obtemos,*

$$L_0 : A_0 + bA_1 = \alpha^3 + \begin{Bmatrix} 0 \\ 1 \\ \beta \\ \beta^2 \end{Bmatrix} \alpha^7 = \begin{Bmatrix} \alpha^3 \\ \alpha^3 + \alpha^7 \\ \alpha^3 + \beta\alpha^7 \\ \alpha^3 + \beta^2\alpha^7 \end{Bmatrix} \overset{p(x)}{\rightarrow} \begin{Bmatrix} \alpha^3 \\ \alpha^4 \\ \alpha^{10} \\ \alpha^6 \end{Bmatrix} \tag{3.45}$$

Na definição de linha pode-se usar como ponto tanto a representação em potencias ($GF(2^4)$*) como a representação vetorial (*$EG(2, 2^2)$*). A linha* L_0 *usa os vetores direção* $bA_1 = \{0, \alpha^7, \alpha^{12}, \alpha^2\}$ *que para gerar a seguinte linha tem que se usar um vetor direção distinto deles, por exemplo* α^3

$$L_1 : \alpha^3 + \left\{ \begin{matrix} 0 \\ 1 \\ \beta \\ \beta^2 \end{matrix} \right\} \alpha^3 = \left\{ \begin{matrix} \alpha^3 \\ 0 \\ \alpha^{13} \\ \alpha^8 \end{matrix} \right\}. \tag{3.46}$$

A linha L_1 usa os vetores direção $bA_1 = \{0, \alpha^3, \alpha^8, \alpha^{13}\}$, para gerar a seguinte linha tem que se usar um vetor direção distinto deles e dos vetores direção de L_0, por exemplo α

$$L_2 : \alpha^3 + \left\{ \begin{matrix} 0 \\ 1 \\ \beta \\ \beta^2 \end{matrix} \right\} \alpha = \left\{ \begin{matrix} \alpha^3 \\ \alpha^9 \\ \alpha^2 \\ \alpha^5 \end{matrix} \right\}. \tag{3.47}$$

A linha L_2 usa os vetores direção $bA_1 = \{0, \alpha, \alpha^6, \alpha^{11}\}$, como nos casos anteriores se usa um vetor direção distinto dos já usados em L_0, L_1 e L_2, por exemplo α^4.

$$L_3 : \alpha^3 + \left\{ \begin{matrix} 0 \\ 1 \\ \beta \\ \beta^2 \end{matrix} \right\} \alpha^4 = \left\{ \begin{matrix} \alpha^3 \\ \alpha^7 \\ \alpha^1 \\ 1 \end{matrix} \right\} \tag{3.48}$$

A linha L_3 usa os vetores direção $bA_1 = \{0, \alpha^4, \alpha^9, \alpha^{14}\}$, e usaremos para L_4 o vetor direção α^5.

$$L_4 : \alpha^3 + \left\{ \begin{matrix} 0 \\ 1 \\ \beta \\ \beta^2 \end{matrix} \right\} \alpha^5 = \left\{ \begin{matrix} \alpha^3 \\ \alpha^{11} \\ \alpha^{12} \\ \alpha^{14} \end{matrix} \right\}. \tag{3.49}$$

A linha L_4 usa os vetores direção $bA_1 = \{0, \alpha^5, \alpha^{10}, 1\}$. É importante ressaltar que se usássemos como vetores de direção todos os pontos em $EG(2, 2^2)$, as linhas L_0, L_1, L_2, L_3 e L_4 teriam o ponto α^3 em comum, e os pontos que conformam estas linhas geram uma $EG(2, 2^2)$. A Figura 3.9 mostra as linhas em sua representação vetorial para $EG(2, 2^2)$.

3.3 Tipos de matrizes

Até agora justificamos a necessidade de ter matrizes de baixa densidade e com alguma regularidade na quantidade de uns na fila e coluna dela, para que a codificação e decodificação sejam da menor complexidade possível e de igual proteção a todos os bits do vetor enviado pelo canal. Neste sentido

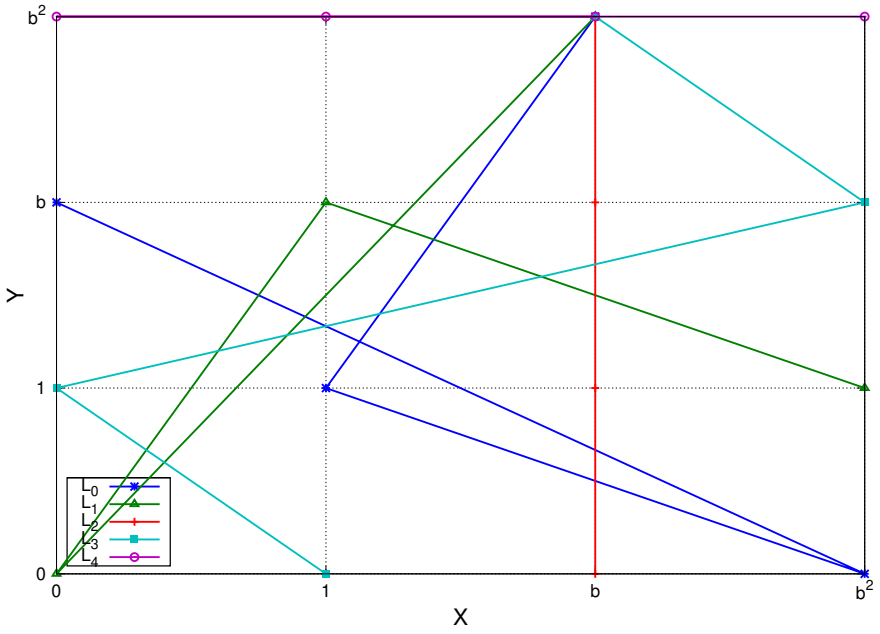

Fig. 3.9: Linhas L_0, L_1, L_2, L_3 e L_4 com pontos em $EG(2, 2^2)$.

existem na literatura algumas regras de formação de matrizes $LDPC$. Entre as mais conhecidas estão as matrizes de Gallager [1].

3.3.1 Códigos de Gallager

Gallager propõe um método para a formação de matrizes de verificação de paridade, H, do tipo $LDPC$, O primeiro que ele faz é estabelecer sua notação da tupla (n, j, k), onde n é a quantidade de colunas, j é o número de uns em cada coluna e k é o número de uns por linha de H. Para calcular a quantidade de linhas da matriz de verificação H basta calcular nj/k. A idéia para projetar uma matriz de Gallager reside primeiro em dividir as linhas da matriz de paridade em j divisões. Em cada uma das divisões só se terá um unico "1" por cada coluna. Para ordenar os uns na matriz o primeiro que se faz é gerar a primeira linha da primeira divisão colocando k uns. Logo se pula a segunda linha e se coloca k uns novamente. E se pula a terceira linha, e assim até encher com k uns todas as linhas da primeira divisão. As divisões seguintes da matriz são permutações aleatórias da primeira, só deslocando o "1" de cada coluna. Este jeito de gerar a matriz tem como consequência que o número de linhas e colunas cumpra a seguinte relação

$$\frac{Número\ de\ Linhas}{n} = \frac{j}{k}. \tag{3.50}$$

No Exemplo 3.3.1 pode-se ver uma aplicação do método de Gallager para gerar matrizes.

EXEMPLO 3.3.1 *No que segue se mostra como gerar uma matriz de paridade* $(n, j, k) = (20, 3, 4)$. *Dado que temos* 20 *colunas podemos calcular a quantidade de linhas, aplicando (3.50), e obtemos que no total temos* 15 *linhas. Estas linhas se dividem em* $j = 3$ *sessões. A primeira sessão será o padrão das seguintes. Como pode-se ver a primeira só fica cheia de uns de forma ordenada até* $k = 4$ *"1"s por linha, como abaixo*

$$H = \left(\begin{array}{cccccccccccccccccccc}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\hline
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
\hline
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{array}\right) \tag{3.51}$$

3.3.2 Códigos de MacKay

Um método de geração de matrizes de verificação de paridade foi proposto por MacKay e Neal em [10, 11]. Nele temos matrizes de tipo A e B, que são tipos derivados dos códigos de Gallager. Para achar as matrizes geradoras pode-se usar eliminação gaussiana e procurar a forma sistemática da matriz de verificação de paridade. A partir desta obtem-se facilmente a matriz geradora.

Matriz tipo 1A

É criada uma matriz de M linhas e N colunas. Cada coluna terá um peso por coluna igual a t e um peso por linha o mais homogêneo possível. A sobreposição entre duas colunas qualquer da matriz não pode ser maior que 1, isto quer dizer que não pode existir duas colunas que tenham mais de um único 1 na mesma posição. Outra forma de expressar isto, é, fazer o produto interno entre duas colunas e ver que não seja maior que um.

Matriz tipo 2A

Dado uma matriz de M linhas e N colunas, se geram até $M/2$ colunas com um peso $t = 2$. Cujos produto internos são nulos. O resto das colunas terá um peso $t = 3$, e o produto interno entre todas as colunas não pode ser maior a 1. As linhas tem que ser feitas de tal forma que o peso de cada coluna seja o mais homogêneo possível.

Matriz tipo 1B e 2B

As matrizes de tipo B são semelhantes às do tipo A só que se apaga uma pequena quantidade de colunas. O objetivo é que o grafo bipartido das matrizes B tenha ciclos de comprimento l no mínimo.

3.3.3 Códigos de Geometria Euclidiana

Os códigos $LDPC$ baseados em geometria euclidiana finita $EG(m, 2^s)$ [5, 9, 3], aplicam toda a teoria vista na sessão 3.2.2. Para conseguir isto, o primeiro ponto é lembrar que numa $EG(m, 2^s)$ existe 2^{ms} pontos e $(2^{ms} - 1)2^{(m-1)s}/(2^s - 1)$ linhas. Se descontamos o origem de coordenadas se obtém $EG^*(m, 2^s)$ com $N = 2^{ms} - 1$ pontos e $J = (2^{ms} - 1)(2^{(m-1)s} - 1)/(2^s - 1)$ linhas que não passam pelo origem de coordenadas. Cada uma das linhas de $EG^*(m, 2^s)$ podem ser representadas mediante um vetor de incidência que é uma representação binaria de cada linha, de tal forma que o vetor tenha N elementos, onde cada elemento com valor 1 significa que o ponto da $EG^*(m, 2^s)$ representada em $GF^*(2^{ms})$ pertence à linha, pelo qual pode entender-se como se cada elemento do vetor de incidência estivesse mapeado a um elemento em $GF^*(2^{ms})$. O $*$ simboliza o mesmo que no caso de $EG^*(m, 2^s)$, isto é, o elemento 0 é excluído da geometria. Dado que numa $EG^*(m, 2^s)$ todas as linhas tem uma quantidade fixa de pontos, os vetores de incidência também terão uma quantidade fixa de uns igual a 2^s. Um vetor de incidência é um vetor binário, onde cada bit do vetor representa a um elemento de uma $GF^*(2^{ms})$, isto quer dizer que pode-se representar os pontos de uma linha por um vetor de incidência. Por exemplo, se pegamos as linhas que não passam pela origem do Exemplo 3.2.7 que é uma $EG^*(2, 2^2)$, obtemos

$$\begin{array}{ll} L_0 = \{\alpha^3, \alpha^4, \alpha^{10}, \alpha^6\} & L_2 = \{\alpha^3, \alpha^9, \alpha^2, \alpha^5\} \\ L_3 = \{\alpha^3, \alpha^7, \alpha^1, \alpha^0\} & L_4 = \{\alpha^3, \alpha^{11}, \alpha^{12}, \alpha^{14}\} \end{array} \tag{3.52}$$

Cada uma destas linhas pode ser representada como vetores de incidência como mostra a Tabela 3.7.

É fácil ver na Tabela 3.7 que cada uma das linha L_j para $j = 0, 2, 3, 4$ é um deslocamento de bits de uma linha padrão. Isto acontece por que o vetor de incidência representa uma linha $L_j = A_0 + b\, A_1^j$ onde $A_0\ e\ A_1^j \in EG^*(m, 2^s)$ e $b \in GF(2^s)$ para $m = 2$ e $s = 2$. Esta linha indica um ponto de

.	α^0	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}
L_0	0	0	0	1	1	0	1	0	0	0	1	0	0	0	0
L_2	0	0	1	1	0	1	0	0	0	1	0	0	0	0	0
L_3	1	1	0	1	0	0	0	1	0	0	0	0	0	0	0
L_4	0	0	0	1	0	0	0	0	0	0	0	1	1	0	1

Tab. 3.7: Vetores de incidência das linhas de $EG^*(2, 2^2)$ que passam por o ponto α^3 do Exemplo 3.2.7.

partida A_0 um escalar b e um vetor de direção A_1^j. Se nós multiplicamos esta linha representada em $GF^*(2^{ms})$ por um $\alpha^l \in GF^*(2^{ms})$ obteríamos uma linha $\alpha^l L_j$ que é um deslocamento de l bits de L_j. Mas também pode entender-se como uma nova linha com um ponto de partida em $\alpha^l A_0$ e na direção $\alpha^l A_1^j$. Assim, se vê que existem a mesma quantidade de linhas que elementos em $GF^*(2^{ms})$, e todas são deslocamentos de uma linha qualquer deste grupo.

Códigos EG-LDPC Tipo-I

Conhecendo a representação das linhas numa $EG^*(m, 2^s)$ mediante vetores de incidência, pode-se usar isto para modelar uma matriz de verificação de paridade H com $N = 2^{ms} - 1$ colunas e $J = (2^{ms} - 1)(2^{(m-1)s} - 1)/(2^s - 1)$ linhas como o conjunto de todos os vetores de incidência que representam a todas as linhas de $EG^*(m, 2^s)$. Deste jeito se obtém uma matriz H com uma quantidade fixa de $\rho = 2^s$ uns por linha, já que este valor representa a quantidade de pontos por linha que é fixo. Para calcular a quantidade de uns por coluna só se tem que conhecer que os uns nas colunas representam a quantidade de linhas que passam por um ponto em $EG^*(m, 2^s)$, lembrando a equação (3.40) vemos que temos $\gamma = (2^{ms} - 1)/(2^s - 1) - 1$ uns por coluna. As matrizes H geradas deste jeito chamam-se códigos $EG - LDPC$ de tipo I, e contem em geral muitas linhas que não são linearmente independentes, mas estas linhas não tem que ser apagadas porque elas cumprem a função de homogenizar os pesos por coluna da matriz H, além disso dado que se tem N colunas em J linhas e as linhas correspondem aos N deslocamentos cíclicos de uma linha padrão, então, pode-se calcular a quantidade de linhas padrões η como o quociente entre J e N.

$$\eta = J/N \tag{3.53}$$

Cada linha padrão gera uma matriz quadrada H_i de N colunas. Assim, a matriz H estará formada como segue

$$H = \begin{bmatrix} H_0 \\ H_1 \\ \vdots \\ H_{\eta-1} \end{bmatrix}. \tag{3.54}$$

Já que é possível obter a matriz H toda deslocando ciclicamente as **linhas padrão**, então H pode ser representada mediante η polinômios $h_l(X),\ 0 \leq l < \eta$, onde $h_l(X)$ pode ser qualquer polinômio linha de H_l, mas, para manter uma notação se usa a linha com o polinômio de menor grau.

Para obter o polinômio gerador $g(X)$ da matriz G mediante a seguinte equação

$$X^N + 1 = g(X)h(X), \tag{3.55}$$

é necessário acha-se primeiro $h(X)$, para isto se tem que colocar todas as linhas de H na sua representação polinomial $l_i(X),\ i = 0, ..., J-1$. Obter o máximo divisor comum

$$h_r(X) = MDC\left\{l_0(X), l_1(X), ..., l_{J-1}(X)\right\}, \tag{3.56}$$

de todos os polinômios e logo calcular o polinômio recíproco

$$h(X) = X^K h_r(X^{-1}), \tag{3.57}$$

onde K é o grau do polinômio $h_r(X)$. O objetivo disto é obter um polinômio ortogonal ao fator comum de todos os polinômios linha de H. Este polinômio ortogonal será nosso polinômio gerador $g(X)$.

Um método para obter o máximo divisor comum (MDC) de dois polinômios é realizado mediante divisões sucessivas. O MDC será o último dividendo que deixe um resíduo igual a cero. Isto implica que tenham que se achar par a par o MDC das J linhas, que para um valor de J grande pode levar a um gasto desnecessário de tempo, dado que se pode utilizar o dado que as linhas de H são deslocamentos cíclicos, então o que se faz é obter os polinômios padrão $h_l(X)$, e logo fazer $\rho - 1$ deslocamentos do polinômio de modo que o elemento de grau zero sempre esteja presente no polinômio. Asim obteríamos apenas ρ linhas por cada H_l que chamaremos da $FAMILIA\left\{h_l(X)\right\}$

$$FAMILIA\left\{h_l(X)\right\} = \left\{h_l^0(X), h_l^1(X), \cdots, h_l^{\rho-1}(X)\right\}. \tag{3.58}$$

A partir desta família se obterá $h_l^{MDC}(X)$ que é o MDC de todos os polinômios linha de H_l

$$h_l^{MDC}(X) = MDC\left\{FAMILIA\left\{h_l(X)\right\}\right\}. \tag{3.59}$$

Finalmente se achará o MDC de todos os η polinômios $h_l^{MDC}(X)$ com:

$$h_r(X) = MDC\left\{h_0^{MDC}(X), h_1^{MDC}(X), \cdots, h_{\eta-1}^{MDC}(X)\right\} \tag{3.60}$$

EXEMPLO 3.3.2 *Para uma $EG^*(m, 2^s)$ com $m = 2$ e $s = 2$ e um polinômio gerador $p(X) = 1 + X + X^4$ obtêm-se uma matriz H com $N = 15$ colunas e $J = 15$ linhas, com $\rho = 4$ uns por linha e $\gamma = 4$ uns por coluna. Usando a definição de Linha $L_i = A_0 + bA_1^i$ onde $A_0, A_1^i \in GF(2^{ms}) = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{14}\}$, $b \in GF(2^s) = \{0, 1, \beta, \beta^2\}$, $\beta^3 = \alpha^{15} = 1$ e $\eta = J/N = 1$, se inicia com um valor qualquer de $A_0 = 1$ e $A_1^0 = \alpha$ obtendo,*

$$L_0 = \left\{1, \alpha^4, \alpha^{13}, \alpha^{12}\right\} \tag{3.61}$$

Dado que a linha L_0 não contem o elemento zero então a linha pertence a $EG^*(m, 2^s)$, e como se tem $\eta = 1$ linhas padrão L_0 pode gerar toda a matriz H como mostra a equação abaixo

$$H = \begin{pmatrix} L_0 \\ \alpha^1 L_0 \\ \alpha^2 L_0 \\ \alpha^3 L_0 \\ \alpha^4 L_0 \\ \alpha^5 L_0 \\ \alpha^6 L_0 \\ \alpha^7 L_0 \\ \alpha^8 L_0 \\ \alpha^9 L_0 \\ \alpha^{10} L_0 \\ \alpha^{11} L_0 \\ \alpha^{12} L_0 \\ \alpha^{13} L_0 \\ \alpha^{14} L_0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} \tag{3.62}$$

Para obter o polinômio gerador $g(X)$ tem-se que colocar L_0 na sua representação polinomial $l_0(X) = 1 + X^4 + X^{13} + X^{12}$, dado que $l_0(X)$ contem um "1" que é o elemento de grau zero, então $l_0(X)$ pertence à família de $h_0(X)$. Para obter a $FAMILIA\{h_0(X)\}$, $l_0(X)$ será deslocado $\rho - 1 = 3$ vezes de tal maneira que todos os polinômios contenham um elemento de grau zero e considerando que $X^N = X^{15} = 1$. A família de $l_0(X)$ é dada por

$$h_0^0(X) = l_0(X) = 1 + X^4 + X^{13} + X^{12} \tag{3.63}$$

$$h_0^1(X) = X^{11} l_0(X) = X^{11} + 1 + X^9 + X^8 \tag{3.64}$$

$$h_0^2(X) = X^2 l_0(X) = X^2 + X^6 + 1 + X^{14} \tag{3.65}$$

$$h_0^3(X) = X^3 l_0(X) = X^3 + X^7 + X + 1. \tag{3.66}$$

Aplicando a equação (3.59) se obtêm $h_0^{MDC}(X) = 1 + X + X^3 + X^7$ *e dado que só existe um polinômio padrão* $h_r(X) = h_0^{MDC}(X)$*, para obter o polinômio* $h(X)$ *se aplica a equação*

$$h(X) = X^7 h_r(X^{-1}) = 1 + X^4 + X^6 + X^7, \tag{3.67}$$

e logo se obtém $g(X)$ *mediante a seguinte equação*

$$g(X) = (X^{15} + 1)/h(X) = 1 + X^4 + X^6 + X^7 + X^8. \tag{3.68}$$

Este polinômio sera usado para gerar a matriz G

$$G = \begin{pmatrix} g(X) \\ Xg(X) \\ X^2g(X) \\ X^3g(X) \\ X^4g(X) \\ X^5g(X) \\ X^6g(X) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}. \tag{3.69}$$

Capítulo 4

Algoritmos de Decodificação

Nos capítulos 2 e 3 foram descritos os fundamento teóricos da transmissão digital e os códigos corretores de erros utilizados nesta dissertação. Neste capítulo veremos os distintos algoritmos de decodificação aplicados aos códigos do capítulo 3.

Gallager [1] ao introduzir a classe de códigos $LDPC$, também propôs um algoritmo muito simples de decodificação por decisão abrupta, isto é, um limiar de decisão é usado no valor real observado na saída do canal $AWGN$, gerando um vetor recebido binário. Este algoritmo ficou conhecido como Bit-Flipping (BF). Sipser e Spielman estudaram em [2] uma sub-classe dos códigos $LDPC$, chamados "expander codes", e propuseram duas formas novas de algoritmos BF com decisão abrupta. Kou et al. propuseram em [3] uma versão da decodificação BF para decisão suave, isto é, um algoritmo que trabalha com o valor real da saída do canal. Esta versão é conhecida como decodificação weighted BF (WBF). Posteriormente, Zhang e Fossorier [4] modificaram a função flipping do algoritmo WBF introduzindo um fator de correção que depende de um parâmetro a ser otimizado. Esta variante é denominada modified WBF ($MWBF$). Este capítulo propõe e estuda duas versões modificadas dos algoritmos de Sipser e Spielman, denominadas Serial Hard Bit-Flipping $(SHBF)$ e Parallel Hard Bit-Flipping $(PHBF)$.

4.1 Limites teóricos

Antes de iniciar-se o estudo dos algoritmos de decodificação, é importante ter em mente que nem sempre é possível realizar-se uma transmissão de informação com uma taxa de erro de bit (BER, do inglês, Bit Error Rate) próximo a zero, como visto na seção 2.4. A possibilidade de decodificar os bits com uma BER próxima de zero depende em principio da taxa de codificação, isto é, da quantidade de bits de redundância. A consideração disto nos levará a obter o nosso primeiro limitante de desempenho.

4.1.1 Limite de Shannon

O limite de Shannon para um canal de comunicações é a máxima taxa teórica de transmissão de informação no canal para um nível dado de ruído. Na equação (2.31) da seção 2.4 foi visto que para se ter uma comunicação livre de ruído é necessário que a taxa do código, R_c, seja menor do que a capacidade do canal, C. Em outras palavras, que a quantidade de informação no bit codificado (Rc) seja menor que a quantidade máxima de informação (C) que o canal pode transportar de forma confiável . Então, o limite de Shannon é atingido quando igualamos a taxa do código e a capacidade do canal

$$R_c = C \tag{4.1}$$

Dado que a capacidade de canal se obtêm em função da quantidade de ruído que tem no canal e também da energia gasta por bit de informação, ela pode ser escrita das três maneiras seguintes

$$C = f_1(E_b/\sigma^2) \tag{4.2}$$
$$= f_2(E_b/N_0) \tag{4.3}$$
$$= f_3(p), \tag{4.4}$$

onde E_b é a energia media por bit de informação , $N_0/2 = \sigma^2$ é a densidade espectral de potência do ruído de banda-dupla no canal $AWGN$ e p é a probabilidade de erro do bit no canal BSC derivado do canal $AWGN$. A Figura 4.1 representa a melhor curva do desempenho de um código corretor de erro, como se vê para valores da relação sinal ruido menores que o limite de Shannon é impossível obter uma BER igual a zero, o método para obter a curva será estudado na seção 4.1.3.

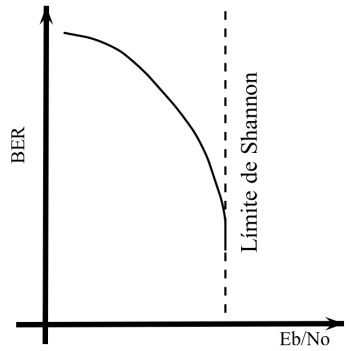

Fig. 4.1: Gráfico de BER versus E_b/N_0 atingindo o limite de Shannon.

Limite de Shannon para um canal AWGN

A definição de um canal $AWGN$ foi vista na seção 2.3.2. Neste modelo de canal tem-se uma entrada $X \in \Re$ uma saída $Y \in \Re$ e um ruído aditivo gaussiano e branco $Z \in \Re$ (veja Figura 4.2). A capacidade deste canal é $C = \frac{1}{2}\, log_2(1 + \frac{S}{N})$. A partir desta expressão, e, usando a equação (4.1), pode-se expressar o limite Shannon como segue

$$\frac{S}{N} = 2^{2R_c} - 1. \tag{4.5}$$

Na equação (4.5), a energia por símbolo enviado, E_s, pode ser expressa como a energia do bit de informação, E_b, multiplicada pela taxa R_c. Esta expressão é fácil de deduzir se aplicamos o critério de conservação da energia, e igualamos a energia do vetor de informação à energia do vetor de dados codificados, $kE_b = nE_s$, onde k é quantidade de bits de informação e n é a quantidade de bits codificados. A densidade espectral de potência do ruído de banda-dupla pode ser trocada por $N = N_0/2$ e combinando com $E_s = R_c E_b$, vem

$$E_b = E[X_l^2]/R_c \tag{4.6}$$

$$\frac{E_b}{N_0} = \frac{2^{2R_c} - 1}{2R_c} \tag{4.7}$$

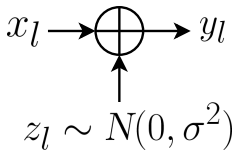

Fig. 4.2: Canal AWGN.

Nas subseções seguintes se estudará os modelos de canais derivados do canal $AWGN$ visto na seção 2.3.2, e além disso se achará o modelo equivalente para o canal BSC visto em 2.3.1.

Limite de Shannon para um canal BI-AWGN

Uma variação do canal $AWGN$ é o canal $BI-AWGN$ (do inglês, Binary Input $AWGN$). Este é um canal com ruído branco aditivo e gaussiano, mas na entrada do canal, X_l só pode ter dois valores, isto é, tem uma entrada binária (ver Figura 4.3).

As probabilidades de transição deste canal são dadas por:

$$p(y|x = \pm 1) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y \mp 1)^2}{2\sigma^2}} \tag{4.8}$$

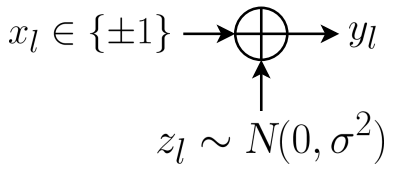

Fig. 4.3: Canal BI-AWGN

E a probabilidade da saída é:

$$p(y) = p(y|x = +1)p(x = +1) + p(y|x = -1)p(x = -1) \tag{4.9}$$

Se os símbolos de entrada são equiprováveis, isto é, $p(x = -1) = p(x = +1) = 0.5$, então

$$p(y) = \frac{1}{2}\{p(y|x = +1) + p(y|x = -1)\}. \tag{4.10}$$

Neste caso, a capacidade do canal $BI\text{-}AWGN$, $C_{BI-AWGN}$, pode ser calculada como segue

$$C_{BI-AWGN} = \boldsymbol{M}(\sigma^2) = \frac{1}{2}\sum_{x=\pm 1}\int_{-\infty}^{+\infty} p(y|x)log_2(\frac{p(y|x)}{p(y)})dy \tag{4.11}$$

$$= -\int_{-\infty}^{+\infty} p(y)log_2(p(y))dy - 0.5log_2(2\pi e\sigma^2). \tag{4.12}$$

Note que a energia por símbolo foi normalizada para a unidade, isto é, $E_s = 1$, e assim $p(y)$ e $p(y|x)$ dependem apenas da variância σ^2.

Mas é necessário chegar a uma relação entre Eb/N_0 e σ^2. Podemos considerar que a energia média do ruído é $E_Z = E[Z^2{}_l] = \sigma^2$, pois como a densidade espectral de potencia é constante com valor $S_Z(f) = N_0/2$, temos para uma largura de faixa infinita

$$E_Z = \lim_{B\to\infty}\frac{1}{2B}\int_{-B}^{B}(N_0/2)df = N_0/2 = \sigma^2. \tag{4.13}$$

De (4.6), (4.13) e lembrando que $E_b = 1$ obtemos

$$E_b/N_0 = 1/(2R_c\sigma^2). \tag{4.14}$$

Aplicando $R_c = C_{BI-AWGN}$, (4.11) a (4.14) vem

$$E_b/N_0 = 1/(2R_c\boldsymbol{M}^{-1}(R_c)), \tag{4.15}$$

onde E_b/N_0 é o limite de Shannon para um canal $BI\text{-}AWGN$.

Limite de Shannon para um canal BI-AWGN com decisão abrupta

O canal $BI\text{-}AWGN$ com decisão abrupta (do inglês, "hard-decision") está ilustrado na Figura 4.4. A saída do canal $BI - AWGN$, Y_l, é aplicada a um decisor F obtendo-se o valor X_l^*,

$$X_l^* = F(Y_l) = \begin{cases} 0 & Y_l \leq 0 \\ 1 & Y_l > 0 \end{cases} \tag{4.16}$$

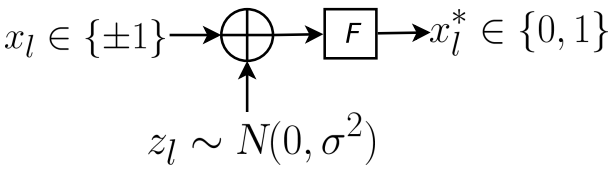

Fig. 4.4: Canal BSC

Na Figura 4.4 pode-se ver como a decisão abrupta torna o canal $BI\text{-}AWGN$ num canal BSC visto na seção 2.3.1, onde a relação entre p da equação (2.8) e E_b/N_0 é dada por

$$p = Q(\sqrt{2R_c E_b/N_0}). \tag{4.17}$$

Juntando (4.17), dado que $R_c = C$ e que a capacidade de canal para um canal BSC é $C = 1 - H(p)$, onde $H(.)$ é a entropia definida na equação (2.12), podemos obter a equação abaixo para o limite de Shannon

$$E_b/N_0 = \frac{1}{2R_c}\{Q^{-1}(H^{-1}(1 - R_c))\}^2. \tag{4.18}$$

A Tabela 4.1 e a Figura 4.5 comparam os limites de Shannon para os três canais descritos anteriormente.

4.1.2 Análise do desempenho de códigos LDGM sistemáticos

No artigo de Garcia-Frias e Zhong [8] foi descrito um limitante inferior para a taxa de erro de bit, BER (do inglês, Bit Error Rate), considerando uma matriz de verificação de paridade sistemática de um código $LDGM$ com $N-K$ nós de verificação paridade de grau $Y+1$, e N nós de bits codificados onde $N - K$ deles tem grau 1 e os restantes K nós tem grau X.

A Figura 4.6 mostra a matriz H transposta correspondente. Nesta classe de códigos pode-se observar uma BER alta para um canal com razão sinal-ruído alta devido aos bits codificados de grau um. Pode-se ver nestes nós que se aplicássemos o algoritmo de passagem de mensagem, a mensagem propagada do bit codificado de grau um para bit de verificação de paridade sempre será a mesma, não

Rc	$(E_b/N_0)_{AWGN}\ (dB)$	$(E_b/N_0)_{BI-AWGN}\ (dB)$	$(E_b/N_0)_{BSC}\ (dB)$
0.05000	-1.44036	-1.440140	0.479204
0.10000	-1.28724	-1.285578	0.593859
0.15000	-1.13238	-1.127028	0.713921
0.20000	-0.97578	-0.963520	0.839975
0.25000	-0.81746	-0.794059	0.972710
0.30000	-0.65740	-0.617563	1.112949
0.35000	-0.49562	-0.432803	1.261682
0.40000	-0.33213	-0.238341	1.420119
0.45000	-0.16692	-0.032437	1.589760
0.50000	0.00000	0.187060	1.772501
0.55000	0.16861	0.422898	1.970789
0.60000	0.33892	0.678679	2.187866
0.65000	0.51090	0.959306	2.428168
0.70000	0.68455	1.271732	2.698015
0.75000	0.85986	1.626371	3.006897
0.80000	1.03683	2.039998	3.370134
0.85000	1.21542	2.542627	3.815174
0.90000	1.39565	3.197745	4.400136
0.95000	1.57748	4.191145	5.295271

Tab. 4.1: Limite de Shannon para os canais $AWGN$, $BI\text{-}AWGN$ e BSC.

afetando o processo de decodificação. Além disso, um erro em algum destes $N-K$ bits de grau um se propagará aos nós de informação que estejam conectados ao mesmo nó de verificação de paridade. Na gráfico da Figura 4.7 $V_i,\ i=0,...,N-1$, são os bits codificados, $U_j,\ j=0,...,K-1$, são os bits de informação, $C_k,\ k=0,...,N-K-1$, são os bits de paridade, e, $S_l,\ l=0,...,N-K-1$, são os bits de verificação de paridade. Seguindo as linhas com setas pode-se ver como um erro no bit de paridade C_1 se propaga aos nós de informação conectados ao nó de verificação de paridade S_1.

Na Figura 4.8 pode-se ver que para que um bit de informação U_0 seja decodificado com erro é necessário que mais da metade dos X bits de paridade C_k ligados aos X bits de verificação de paridade S_l, estejam com erro. Em outras palavras, a probabilidade de errar um bit de informação U_j é a probabilidade de que mais da metade dos X bits C_k ligados a ele estejam errados. Para um X ímpar e uma probabilidade de erro p num canal BSC temos

$$BER_{Limite\ Impar}(p) \approx \sum_{k=(X+1)/2}^{X} \binom{X}{k} p^k (1-p)^{X-k}. \tag{4.19}$$

Se X fosse par a probabilidade de errar um bit de informação U_j é a mesma probabilidade que

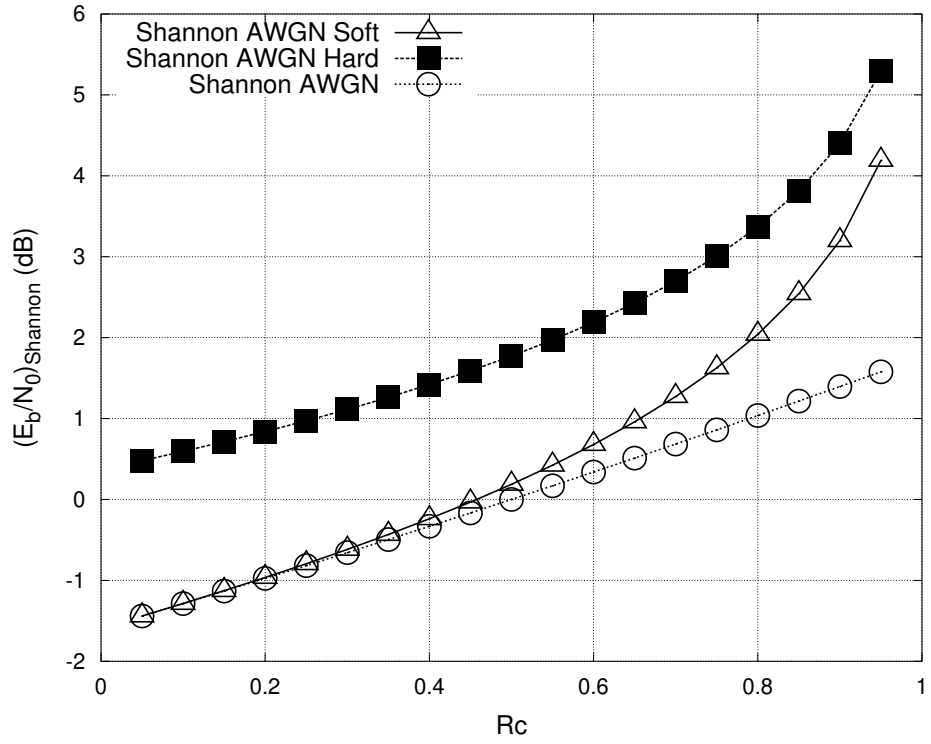

Fig. 4.5: Gráfico do limite de Shannon versus a taxa do código R_c.

$$H^T = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & 1 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 1 \\ 0 & 1 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & 0 & 1 & \cdots & 0 \end{bmatrix} \begin{matrix} \text{Grau 1} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \text{Grau 1} \\ \text{Grau X} \\ \vdots \\ \vdots \\ \vdots \\ \text{Grau X} \end{matrix}$$

Grau Y+1

Fig. 4.6: Grau das linhas e colunas de uma matriz $LDPC$ sistemática.

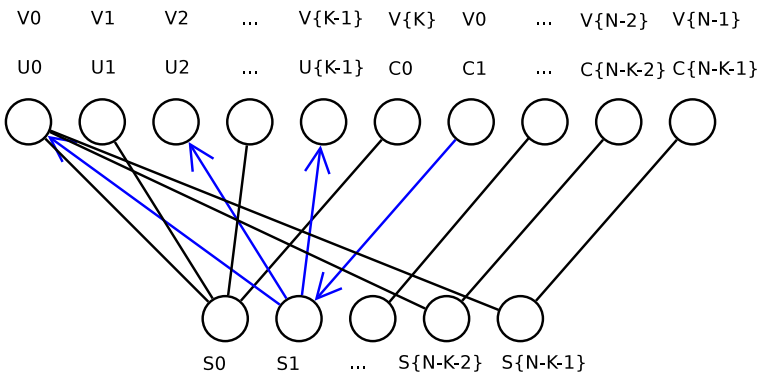

Fig. 4.7: Diagrama de Tanner de uma matriz LDGM com $Y = 3$.

mais da metade dos X bits C_k ligados a ele estejam errados somada à probabilidade de que exatamente a metade dos bits estejam errados (pelo qual teríamos uma probabilidade p de errar nessa decisão). Assim,

$$BER_{Limite\ Par}(p) \approx \sum_{k=(X/2)+1}^{X} \binom{X}{k} p^k (1-p)^{X-k} + p\binom{X}{X/2} p^X/2(1-p)^{X/2} \tag{4.20}$$

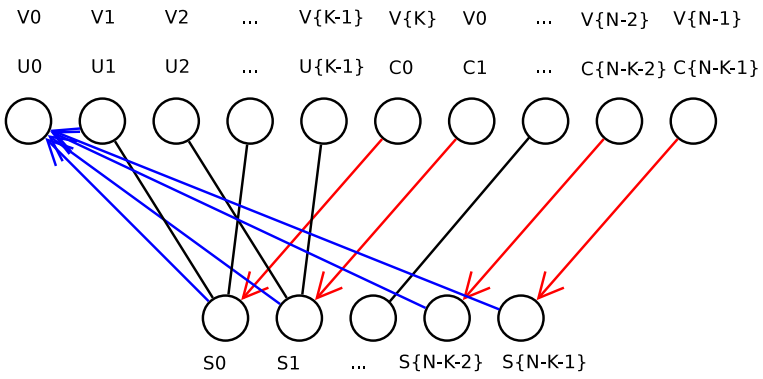

Fig. 4.8: Diagrama de Tanner de uma matriz LDGM com $X = 4$.

As Figuras 4.9 e 4.10 mostram curvas BER limite para um BSC tendo p e E_b/N_0 como parâmetros, respectivamente. São mostradas curvas para distintos valores de X e também o limite de Shannon para uma taxa do código $R_c = 0.85$.

É importante lembrar que as aproximações das equações (4.19) e (4.20) foram feitas considerando que só acontece um erro nos bits de informação, pelo qual as curvas obtidas experimentalmente mediante o uso dos algoritmos de decodificação ficarão próximas a estas só na região de baixa probabilidade de erro do canal.

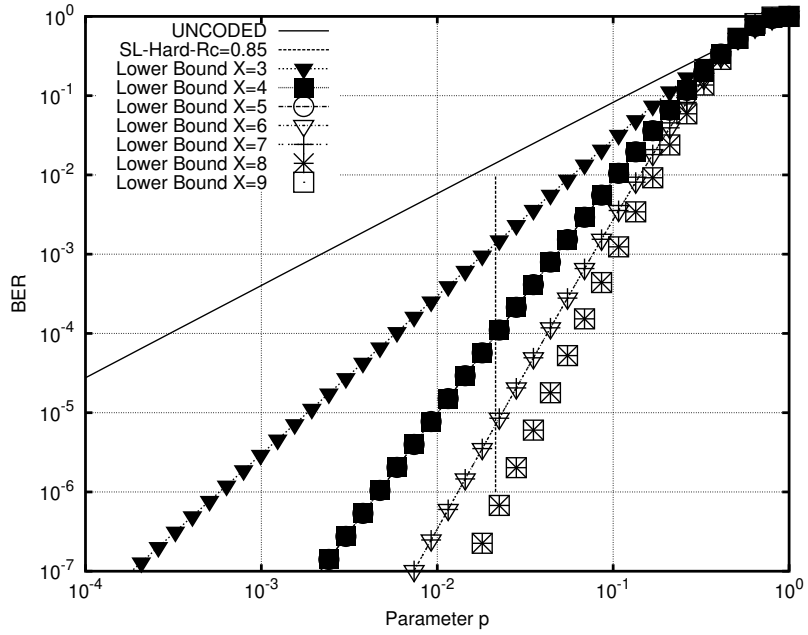

Fig. 4.9: Diagrama do BER limite para um canal BSC.

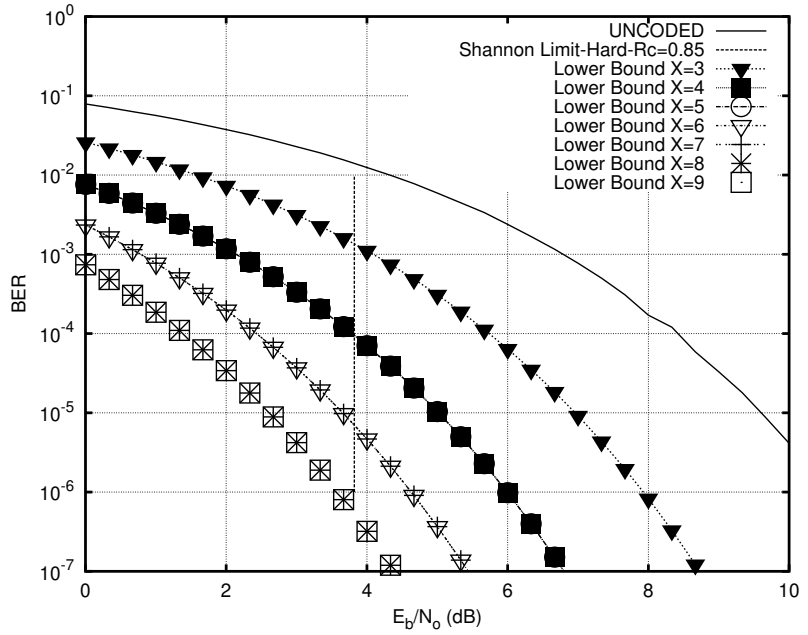

Fig. 4.10: Diagrama do BER limite para um canal $BI\text{-}AWGN$ com decisão abrupta.

4.1.3 BER para canais AWGN com sinalização binária

Na seção 4.1.1 mostrou-se o limite de Shannon para três canais distintos. Ele mostra a razão sinal-ruído mínima necessária para recuperar toda a informação enviada pelo canal. Entretanto, isto

não quer dizer que com uma razão sinal-ruído inferior, toda a informação está perdida. Neste caso, parte da informação é recuperável e a informação restante corresponde à informação gerada com uma taxa de erro de bit. Na Figura 4.11 pode-se ver como na seção desenhada com quadros é onde o canal tem uma razão sinal-ruído superior ao limite de Shannon, e a seção desenhada com linhas é a região onde a razão sinal-ruído ultrapassou o limite de Shannon. Neste ponto pode-se ver que se tem uma BER mínima que pode ser atingida.

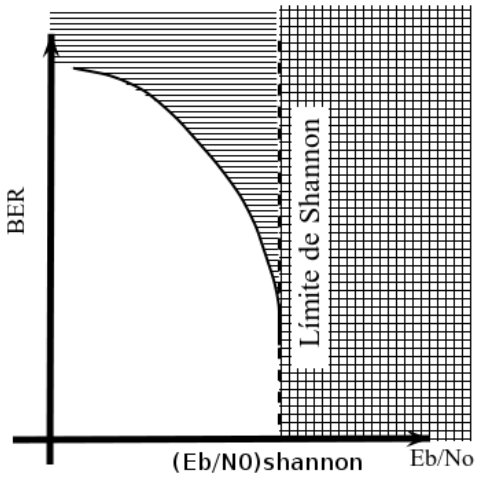

Fig. 4.11: Limite de Shannon; em quadros seção onde é possível não ter perda de dados; em linhas seção em que se tem perda de dados.

Como pode-se ver em [12], é possível calcular-se este limite. Para isto é necessário primeiro considerar que um bit de informação tem incluído nele uma BER, como mostra a Figura 4.12. O cilindro U_i representa um bit de informação onde uma parte do bit está cheio com a informação do erro expressada como $H(BER)$ e o resto do bit corresponde a parte da informação que ficou sem erro. Esta porção pode ser expressa por $1 - H(BER)$. Os k bits de informação são codificados em n bits. Para $k < n$, esse novo bit codificado V_i contem no seu interior o bit de informação. Na Figura 4.12 pode-se ver que a taxa do código R_c corresponde à proporção do bit de informação no bit codificado. Entretanto, por causa do erro, só uma porção dele é valida no bit codificado. Esta nova porção gera uma taxa R'_c virtual. Chamamos ela de virtual por que o bit V_i só tem essa proporção de informação, isto é, o bit V_i só tem $R'_c\%$ dele preenchido com informação.

Assim definimos R'_c como

$$R'_c = R_c(1 - H(BER)) \tag{4.21}$$

R'_c é a quantidade de informação de U_i no bit codificado V_i. Então lembrando que a capacidade do canal C é a máxima quantidade de informação que é possível enviar para um nível de erro do canal, podemos deduzir as seguintes relações. Para uma transmissão de dados livre erros, $R_c \leq C$, e isto

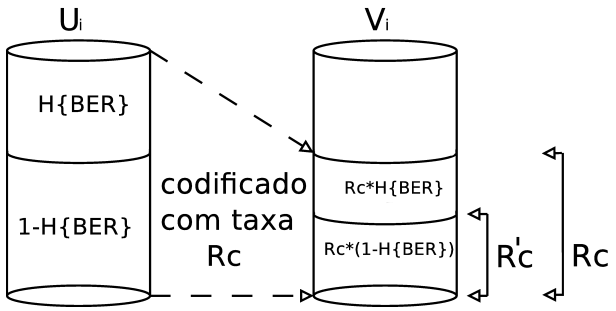

Fig. 4.12: Taxa R'_c virtual no bit codificado V_i gerada pela presença do BER no bit de informação U_i.

gera a região cheia de quadros da Figura 4.11. Para recuperar parte do bit de informação, $R'_c \leq C$, e isto gera a região cheia de linhas da Figura 4.11. Para achar a curva limite, se iguala a taxa virtual R'_c e a capacidade de canal C, isto é, $R'_c = C$.

Dependendo do tipo de canal que se esteja considerando, C poderá estar em função de p (para um canal BSC) ou E_b/N_0 (para os canais $AWGN$ e $BI - AWGN$). O gráfico da Figura 4.13 mostra a curva limite e o limite de Shannon para um canal BSC com uma taxa $R_c = 0.8$ e uma capacidade de canal $C = 1 - H(p)$. Como pode-se ver, a curva limite (desenhada com uma linha) converge assintoticamente para o limite de Shannon (desenhada com pontos). Assim, a equação do gráfico da curva limite ficaria definida como segue

$$C = 1 - H(p) = R_c(1 - H(BER)). \tag{4.22}$$

Ou seja

$$BER(p) = H^{-1}\{1 - \frac{1 - H(p)}{R_c}\}, \tag{4.23}$$

a única restrição para as equações (4.22) e (4.23) é que a taxa do código R_c tem que ser maior que a capacidade C, que é o mesmo que dizer que a informação contida no erro tem que ser maior que zero, isto é,

$$C \leq R_c \tag{4.24}$$

$$0 \leq H(BER). \tag{4.25}$$

Ambas restrições podem ser facilmente deduzidas olhando a Figura 4.12.

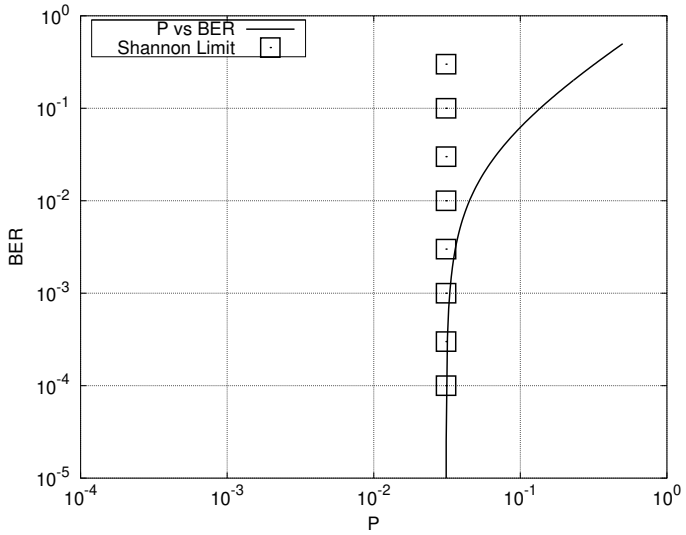

Fig. 4.13: Curva limite gerada pela presença de erro no bit de informação para uma $R_c = 0.8$ e um canal BSC.

4.2 Bit-flipping

O algoritmo de decodificação bit-flipping (BF) foi idealizado por Gallager [1] como uma proposta para identificar e corrigir os erros na transmissão de informação num canal BSC. Depois de que o vetor codificado, $V = U\ G$, transita através do canal com ruido, este chega a seu destino com erros. Este novo vetor binário é chamado de Z, e, para corrigir possíveis erros nele, o algoritmo atribui a cada bit z_n de Z uma confiabilidade E_n. Esta confiabilidade indica a menor ou maior possibilidade de que esse bit esteja errado. O critério para atribuir as confiabilidades segue a equação

$$E_n = \sum_{m \in \mathcal{M}(n)} s_m. \tag{4.26}$$

A matriz H pode ser representada mediante M vetores binários h_m de comprimento N, onde $m = 0, 1, \ldots, M-1$. Assim, $s_m = Z\ h_m^t$, onde h_m^t é um vetor coluna. Deste modo, definimos $\mathcal{M}(n)$ como o conjunto de todos os valores que pode tomar m tal que $h_{m,n} = 1$, ou seja,

$$\mathcal{M}(n) = \{m | h_{m,n} = 1\}. \tag{4.27}$$

Pode-se entender melhor $\mathcal{M}(n)$ mediante a Figura 4.14, onde $\mathcal{M}(1) \in \{1, 2\}$ representa o conjunto dos índices dos bits s_m que estejam ligados ao bit z_1.

Após estabelecer o critério de geração de confiabilidades E_n para cada bit z_n é necessário gerar também um critério para trocar os bits. O algoritmo BF troca todos os bits com $E_n \geq \delta$. Na

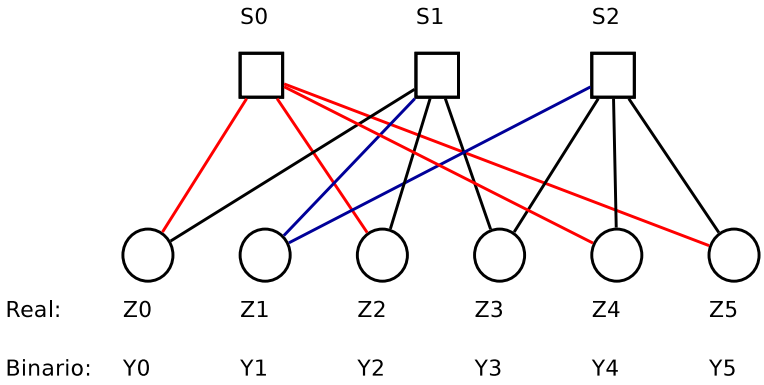

Fig. 4.14: Gráfico de Tanner de um código (N,K)=(6,3).

implementação que segue do algoritmo BF, se usa como limiar δ o valor máximo de E_n no vetor atual. O algoritmo 1 descreve todo o procedimento.

Algoritmo 1 Decodificação Bit-Flipping

1. Inicia-se com $k = 0$ e o vetor de decisão abrupta $Z^{(k)} = (z_1^{(k)}, \cdots, z_n^{(k)}, \cdots, z_N^{(k)}) = \mathbf{z}$.
2. Calcule a síndrome $s_m = \sum_n h_{m,n} z_n^{(k)}$ (mod 2) para $m = 1, \cdots, M$. Se todos os valores são zero então pára a decodificação e se retorna o vetor $Z^{(k)}$.
3. Para $n = 1, \cdots, N$ Se calcula a função de flipping E_n em (4.26).
4. Identifica-se o conjunto $\{n^*\}$, onde $n^* = \arg\max_n E_n$.
5. Troca-se de valor todos os bits $z_n^{(k)}$ onde $n \in \{n^*\}$, e faz-se $Z^{(k+1)} = Z^{(k)}$.
6. Se o número máximo de iterações não é atingido, faz-se $k = k + 1$ e continua-se no passo 2. Caso contrário se detêm a decodificação e se retorna **Z** .

Pelo visto no Algoritmo 1 a decodificação BF pode-se entender como um algoritmo baseado em confiabilidades, onde o conjunto de bits menos confiáveis são considerados errados e seu valor é trocado. A confiabilidade de cada bit e calculada somando a quantidade de bits de verificação de paridade que ligados ao bit codificado indiquem a existência de um erro. Muito parecido a uma votação, onde só os votos em contra são contabilizados para o bit em estudo, ao final o conjunto de bits que tenham mais votos em contra serão trocados.

4.3 O Algoritmo Weighted Bit-Flipping

O algoritmo de decodificação weighted bit-flipping (WBF) é uma variante do algoritmo BF [3]. Ele está orientado a trabalhar com um vetor real Y como dado de entrada. Este vetor Y pode ser obtido por exemplo na saída de um canal $AWGN$. Para a geração das confiabilidades E_n de cada bit usa-se uma ponderação de $|y_n|$ e dos valores de s_m, usando o valor $+1$ para $s_m = 1$ e o valor -1 para $s_m = 0$. Diferentemente do algoritmo BF, o algoritmo WBF usa os bits de verificação de paridade corretos, com $s_m = 0$, além dos bits de verificação errados, com $s_m = 1$, e atribui a cada bit s_m um fator de ponderação w_m, como segue

$$E_n = \sum_{m \in \mathcal{M}(n)} (2s_m - 1)w_m, \tag{4.28}$$

onde

$$w_m = \min_{i \in \mathcal{N}(m)} |y_i|. \tag{4.29}$$

O conjunto $\mathcal{M}(n)$ de (4.28) é igual ao do algoritmo BF. $\mathcal{N}(m)$ é o conjunto de todos os índices dos bits y_i ligados a s_m. Por exemplo na Figura 4.14 o conjunto $\mathcal{N}(0) = \{0, 2, 4, 5\}$. Assim w_m é o valor mínimo derivado do valor absoluto dos y_i ligados ao bit s_m. Obtidas as confiabilidades para cada bit, o algoritmo WBF troca o bit que tenha o maior valor. Dado que a confiabilidade usa valores reais, o mais provável é que se troque um só bit.

Algoritmo 2 Decodificação Weighted Bit-Flipping

1. Inicia $k = 0$ e o vetor $Z^{(k)} = (z_1^{(k)}, \cdots, z_n^{(k)}, \cdots, z_N^{(k)})$ com a decisão abrupta do vetor Y.
2. Avalia o síndrome $s_m = \sum_n h_{m,n} z_n^{(k)}$ (mod 2) para $m = 1, \cdots, M$. Se todos os valores são cero então se detêm a decodificação e se retorna o vetor $Z^{(k)}$.
3. para $n = 1, \cdots, N$ Se avalia a função de flipping E_n em (4.28).
4. Identifica o E_j^* que é o major valor de todos os E_n.
5. Troca-se de valor o elemento $j - esimo$ de $Z^{(k)}$, e $Z^{(k+1)} = Z^{(k)}$.
6. Se o máximo número de iterações não é atingido, coloque $k = k + 1$ e vá ao passo 2. Caso contrario se detém a decodificação e retorna **Z** .

4.4 Os algoritmos: Modified WBF e Improvement Modified WBF

O algoritmo Modified WBF ($MWBF$) foi proposto em [4] como uma modificação ao algoritmo WBF. O algoritmo $MWBF$ utiliza a função de flipping

$$E_n(\alpha) = \sum_{m \in \mathcal{M}(n)} (2s_m - 1)w_m - \alpha|y_n|. \tag{4.30}$$

Na equação (4.30) pode-se ver a confiabilidade descrita pelo bit de verificação de paridade e também pelo bit do vetor recebido. Esta última é ponderada com um sinal negativo dado que o comportamento desta é inverso ao da confiabilidade do bit de verificação de paridade. O fator de ponderação $\alpha > 0$ é colocado devido ao fato de que é provável que a matriz de verificação de paridade, H, seja irregular e $\mathcal{M}(n)$ tenha distintas cardinalidades. Por isto é necessário obter experimentalmente o melhor valor para α.

$MWBF$ usa o algoritmo 2 para a decodificação, com a diferença de que usa a equação (4.30) para a geração de confiabilidades E_n. O algoritmo de decodificação improvement $MWBF$ ($IMWBF$) é uma variante do $MWBF$ proposta em [13]. $IMWBF$ vê a decodificação WBF como uma primeira aproximação do algoritmo de passagem de mensagem. Nesse algoritmo a confiabilidade enviada de um bit de verificação de paridade exclui a mensagem do bit que recebe a informação. Na equação (4.32) pode-se ver como o n-ésimo elemento é excluído do cálculo de $w_{n,m}$, ficando a confiabilidade E_n para o algoritmo $IMWBF$ como

$$E_n(\alpha) = \sum_{m \in \mathcal{M}(n)} (2s_m - 1)w_{n,m} - \alpha|y_n|, \tag{4.31}$$

onde

$$w_{n,m} = \min_{i \in \mathcal{N}(m) \backslash n} |y_i|. \tag{4.32}$$

$IMWBF$ usa o algoritmo 2 para a decodificação, com a diferença que usa as equações (4.32) e (4.31).

4.5 Algoritmos de Sipser e Spielman

Sipser e Spielman propuseram em [2] uma nova maneira de gerar o vetor de confiabilidade e a estimação dos bits errados para um canal com saída abrupta. O primeiro algoritmo que propuseram foi um algoritmo de decodificação sequencial. Ele é mostrado como algoritmo 3.

O algoritmo 3 pode ser descrito como os algoritmos de bit-flipping. Por exemplo, o passo 1 é equivalente a: atribua uma confiabilidade $+1$ quando $s_m = 1$ e uma confiabilidade -1 quando

Algoritmo 3 Decodificação de Sipser e Spielman

1. Se houver um bit z_n que tem mais bits de verificação de paridade s_m insatisfeitos que satisfeitos, então troca-se este bit.
2. Repita o passo anterior até que nenhuma variável fique com bits de verificação de paridade insatisfeitos.

$s_m = 0$. Caso contrário, some as confiabilidades de todos os bits s_m ligados a z_n. Se a confiabilidade resultante E_n é maior que zero, troque o bit z_n. Assim, a função de flipping é dada por

$$E_n = \sum_{m \in \mathcal{M}(n)} (2s_m - 1), \tag{4.33}$$

onde $\mathcal{M}(n)$ é como na equação (4.27).

Note que não se fala nada sobre que bit deve ser trocado. Dado que o algoritmo é sequencial se trocará um só bit por vez e como no algoritmo BF se procederá ao cálculo da síndrome. Se ela for zero, finaliza-se a decodificação. Se for distinta de zero, se aplica o algoritmo 3 até que todas as síndromes sejam nulas, ou se atinja um número máximo de iterações. Sipser e Spielman também propuseram um algoritmo paralelo que consiste em realizar a troca de bits (flipping) de todos os bits com $E_n > 0$ numa única iteração. A partir de agora denominaremos o algoritmo sequencial de Sipser e Spielman como $SSSBF$ ($SequencialSipserSpielmaBit - Flipping$) e o paralelo de $PSSBF$ ($ParallelSipserSpielmaBit - Flipping$).

4.6 Algoritmos BF modificados com decisão abrupta

Nesta seção propõe-se uma modificação do algoritmo de decodificação $SSSBF$. Esta modificação será denominada $PHBF$ (do inglês, Parallel Hard Bit-Flipping). Ela trabalha sobre um vetor de entrada binário Z e tem a mesma regra de geração de confiabilidade, E_n, do algoritmo $SSSBF$. A regra foi vista na equação (4.33). A diferença aqui está no critério de troca dos bit errados: é igual ao algoritmo BF. Troca-se todos os bits com um $E_n > \delta$. Para a implementação atual usou-se como δ o maior valor de E_n. O algoritmo 4 detalha todo o procedimento para a decodificação $PHBF$.

Existe uma relação entre o algoritmos BF e $PHBF$ pois ambos utilizam um regra de confiabilidade parecida. A primeira vista poderia parecer que a confiabilidade de $PHBF$ (veja equação 4.33) é simplesmente uma combinação linear da confiabilidade de BF (veja equação 4.26). Assim, não deveria existir nenhuma diferença quando se acha a confiabilidade do valor máximo. E portanto também na decodificação de Z. Isto é verdade só no caso em que o número de elementos de $\mathcal{M}(n)$ é constante. A verificação disto é fácil de ver se consideramos a quantidade de bits por coluna de H,

Algoritmo 4 Decodificação Parallel Hard Bit-Flipping

1. Inicia-se com $k = 0$ e o vetor de decisão abrupta $Z^{(k)} = (z_1^{(k)}, \cdots, z_n^{(k)}, \cdots, z_N^{(k)}) = \mathbf{Z}$.
2. Calcula-se a síndrome $s_m = \sum_n h_{m,n} z_n^{(k)}$ (mod 2) para $m = 1, \cdots, M$. Se todos os valores são nulos, então se detêm a decodificação e se retorna o vetor $Z^{(k)}$.
3. Para $n = 1, \cdots, N$, avalia-se a função de flipping E_n em (4.33).
4. Identifica-se o conjunto $\{n^*\}$, onde $n^* = \arg\max_n E_n$.
5. Troca-se de valor todos os bits $z_n^{(k)}$ onde $n \in \{n^*\}$, e faz-se $Z^{(k+1)} = Z^{(k)}$.
6. Se o máximo número de iterações não é atingido, faz-se $k = k + 1$ e vai-se ao passo 2. Caso contrário, se detêm a decodificação e se retorna $\mathbf{Z}$.

$X(n)$,

$$X(n) = \sum_{m=0}^{M-1} h_{m,n}. \tag{4.34}$$

$X(n)$ também por ser interpretado como a quantidade de bits de verificação de paridade s_m que estão conectados à z_n.

Se consideramos que se tem l bits de verificação de paridade errados para o bit z_n então o valor da confiabilidade segundo o algoritmo BF seria $E_n^{BF} = l$. Por outro lado, a confiabilidade para o algoritmo $PHBF$ seria $E_n^{PHBF} = (+1)l + (-1)(X(n) - l) = 2l - X(n)$. Unindo ambas ideias, obteríamos a equação

$$E_n^{PHBF} = 2E_n^{BF} - X(n). \tag{4.35}$$

A equação (4.35) mostra claramente que se todos os bits z_n possuem a mesma quantidade de bits de verificação de paridade, isto é, se a matriz H possui um grau de coluna constante, os algoritmos BF e $PHBF$ levam ao mesmo resultado na decodificação.

Uma variante do algoritmo $PHBF$ é o serial hard bit-flipping ($SHBF$). Neste algoritmo, a diferença do $PHBF$ está na troca de um só bit por iteração e no fato de dar-se prioridade ao bit de informação para o caso de matrizes sistemáticas. O algoritmo $SHBF$ está completamente detalhado no Algoritmo 5.

Algoritmo 5 Decodificação Serial Hard Bit-Flipping.

1. Inicia-se com $k = 0$ e o vetor de decisão abrupta $Z^{(k)} = (z_1^{(k)}, \cdots, z_n^{(k)}, \cdots, z_N^{(k)}) = \mathbf{Z}$.
2. Calcula-se a síndrome $s_m = \sum_n h_{m,n} z_n^{(k)}$ (mod 2) para $m = 1, \cdots, M$. Se todos os valores são nulos, então, se detêm a decodificação e se retorna o vetor $Z^{(k)}$.
3. Para $n = 1, \cdots, N$, se avalia a função de flipping E_n em (4.33).
4. Identifica-se o conjunto $\{n^*\}$, onde $n^* = \arg\max_n E_n$.
5. Troca-se o valor do primeiro bit de informação de $z_n^{(k)}$ onde $n \in \{n^*\}$. Se este conjunto não contem nenhum bit de informação, troca-se o valor do primeiro bit de verificação de paridade achado.
6. Se o máximo número de iterações não é atingido, faz-se $k = k + 1$ e vai-se ao passo 2. Caso contrário, se detêm a decodificação e se retorna **Z**.

4.7 Resultados de simulações

Nesta seção compara-se os desempenhos dos distintos algoritmos "bit-flipping". É importante lembrar que os algoritmos BF, $SHBF$, $PHBF$, $SSSBF$ e $PSSBF$ usam decisão abrupta na decodificação, isto quer dizer que o vetor de entrada para estes algoritmos é um vetor binário que pode ser obtido usando um modelo de canal BSC ou um $BI-AWGN$ com decisão abrupta, ambos modelos de canal serão equivalentes usando a equação (4.17). Por outro lado os algoritmos WBF e $MWBF$ usam decisão suave na decodificação, isto é, o vetor de entrada dos algoritmos tem valores reais. Conhecendo as necessidades dos dados de entrada dos distintos algoritmos bit-flipping, se vê que a melhor decisão para poder simular todos os algoritmos e poder compará-los de uma maneira justa, é usando como modelo de canal, um canal $BI-AWGN$ e usando sua saída de decisão suave para alimentar os algoritmos WBF e $MWBF$. A esta saída suave se aplicará uma decisão abrupta e o vetor binário gerado será entregue aos algoritmos BF, $SHBF$, $PHBF$, $SSSBF$ e $PSSBF$. Outro ponto importante a ressaltar é que por causa de ter sido usado dois modelos de canal distintos, o $BI-AWGN$ com decisão abrupta e o $BI-AWGN$ com decisão suave, se tem também dois limites de Shannon distintos. Como os algoritmos nesta tese são o $PHBF$ e o $SHBF$, e eles são simulados num canal $BI-AWGN$ com decisão abrupta ou BSC, se usará então o limite de Shannon para este modelo em todos os gráficos.

Nas simulações se usarão dois tipos de matrizes, as $LDGM$ explicadas na seção 3.1.2 e $EG-LPDC$ na seção 3.3.3. No caso das matrizes $LDGM$ a matriz de paridade P de $G = [P\ I_K]$ pode ser obtida de três maneiras. A primeira é utilizando uma matriz de verificação de paridade $LDPC$ regular como nossa matriz P. Todas as matrizes $LPDC$ regulares usadas foram obtidas do site de MacKay

[14]. A segunda maneira de obter-se a matriz P é gerando-a aleatoriamente com uma quantidade de uns por linha constante. A terceira maneira é quase igual à anterior com a diferença que além de tentar conservar fixa a quantidade de uns na linha também se tenta conservar a quantidade de uns na coluna, isto é, procura-se uma matriz P que seja $LDPC$ e regular. No caso das matrizes $EG-LDPC$ se usarão matrizes $EG-LDPC$ de tipo I. Diferentemente das $LDGM$, as $EG-LDPC$ são matrizes regulares com uma quantidade de uns por linha e coluna da matriz de verificação de paridade muito superior as $LDGM$ e com a vantagem de que as $EG-LDPC$ são matrizes cíclicas. Assim, a geração e verificação de paridade é mais simples de implementar. Com respeito a notação, se usará d_v e d_c para indicar a quantidade de uns por coluna e linha respectivamente da matriz de verificação de paridade H, e se usará d_v^* e d_c^* para indicar a quantidade de uns por coluna e linha, respectivamente, da matriz de paridade P. Além de obter-se os gráficos dos distintos algoritmos $bit-flipping$ também se mostrará os gráficos da curva estimada por Javier Garcia-Frias e Wei Zhong no seu artigo [8]. É uma curva para a taxa de erro do bit (BER) na região inferior de erro para matrizes $LDGM$. Esta curva é chamada nos gráficos como "Lower Bound" e depende da quantidade de uns por linha d_v^* da matriz P, e é descrita pelas equações (4.19) e (4.20). Quando um algoritmo $bit-flipping$ atinge uma quantidade IT máxima de iterações, ao reparar os bits do vetor codificado recebido, o algoritmo desiste e entrega na sua saída o mesmo vetor entregado a ele na sua entrada sem nenhuma modificação, e subtrai dele o vetor de informação. Para códigos $LDGM$ isto é simples dado que o vetor de informação é uma porção do vetor codificado. No caso dos códigos $EG-LDPC$ as simulações para mostrar a BER contabilizam os bits errados no vetor codificado, não do vetor de informação como é mais comum. Isto, no caso dos códigos $EG-LDPC$ é uma boa aproximação estatística dado que a matriz de verificação de paridade é uma matriz regular e todos os bits descodificados por ela tem a mesma possibilidade de serem descodificados corretamente. Assim a estatística do total (vetor codificado) é aproximadamente igual a estatística de uma porção dele (vetor de informação). Esta tendência é maior quanto maior seja a quantidade N de bits codificados.

Todos os programas utilizados nesta tese podem ser descarregados de [15] e seu funcionamento estão explicado no apêndice B.

A Figura 4.15 mostra o desempenho dos algoritmos BF, $SHBF$, $PHBF$, $SSSBF$, $PSSBF$, WBF e $MWBF$, num canal $BI-AWGN$, usando na codificação uma matriz $LDGM$ com $N=192$ bits codificados e $K=128$ bits de informação e taxa do código $R_c=0.6667$. A matriz P da matriz sistemática foi gerada aleatoriamente tentando conservar constante a quantidade de uns por linha e coluna obtendo $d_v^*=9$ e $d_c^*=18$. Para todos os algoritmos $bit-flipping$ usou-se uma quantidade máxima de iterações $IT=6$. Como pode se observar, o algoritmo $PSSBF$ tem o pior desempenho e os algoritmos WBF e $MWBF$ tem o melhor desempenho, lembrando que estos dois últimos algoritmos necessitam dados em ponto flutuante para decodificar, o que implica numa

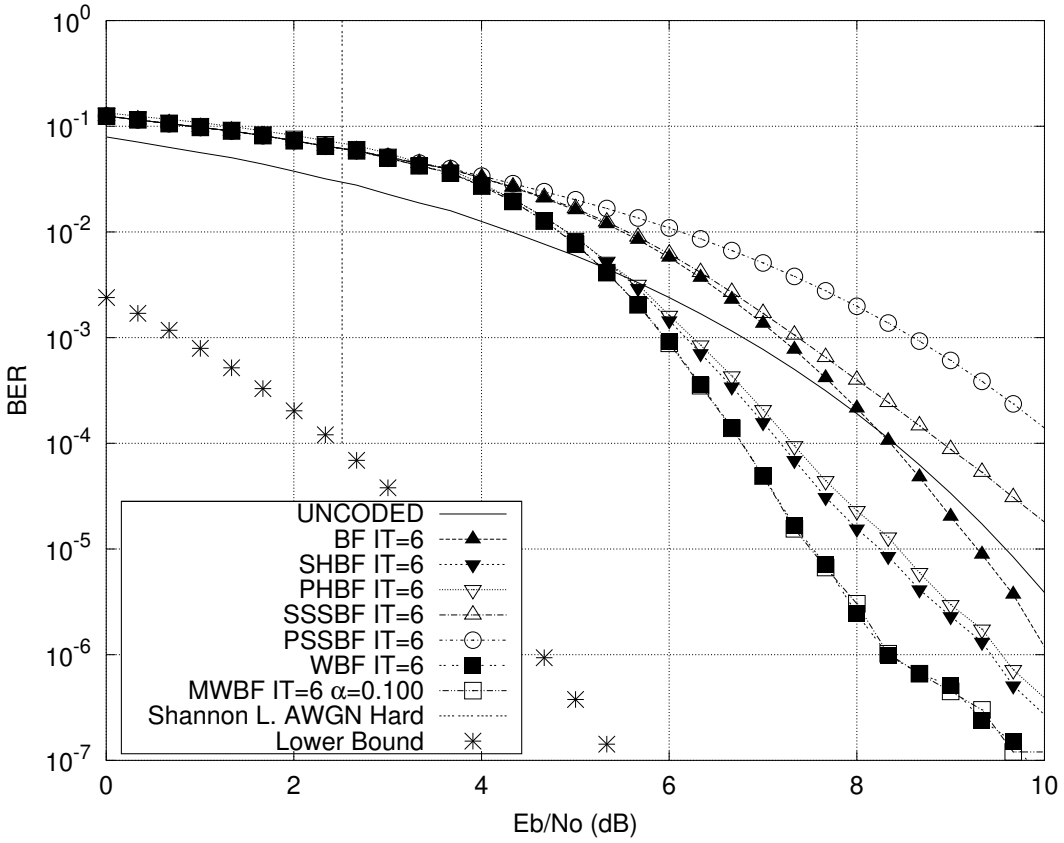

Fig. 4.15: Simulação dos algoritmos bit-flipping num canal $BI-AWGN$ com uma $LDGM$ de K=128 bits de informação, N=192 bits codificados, 6 iterações como máximo, $d_v^* = 9$ e $d_c^* = 18$.

grande necessidade de computação, comparado em desempenho com os algoritmos $SHBF$ e $PHBF$ que fazem uso de cálculos em ponto fixo. $SHBF$ e $PHBF$ tem um desempenho aceitável mas não melhor que WBF, isto acontece em parte pelo baixo numero de elementos no vetor codificado ($N = 192$) e pela topologia da matriz de verificação de paridade H. No gráfico da Figura 4.16 tem-se o mesmo código $LDGM$ que na Figura 4.15 com a diferença que a quantidade de iterações máximas é 12. Pode se observar que as curvas não recebem uma melhora significativa, isto porque as curvas já são o melhor que elas podem ser, e incrementar a quantidade de iterações não melhoraria muito seu desempenho. Esta quantidade de iterações máxima obtida é experimental e depende da observação do comportamento da curva. Outra forma de encontrar o momento em que as curvas chegam a ser o melhor que elas podem ser, é observando os algoritmos $SHBF$ e $PHBF$. Dado que o primeiro é um algoritmo de decodificação sequencial e o segundo de decodificação paralela, este último tem que chegar mais rápido a seu melhor desempenho. Uma boa forma de observar como se está perto de atingir a quantidade de iterações máximas ótima, é observar como a curva de $SHBF$ e $PHBF$ ficam

próximas. Assim, a quantidade de iterações máximas (IT) usadas na decodificação é quase-ótima se não é mais possível aumentar um pouco IT, isto é, em teoria os algoritmos sequenciais são melhores que os paralelos no desempenho só que convergem a este valor mais lentamente.

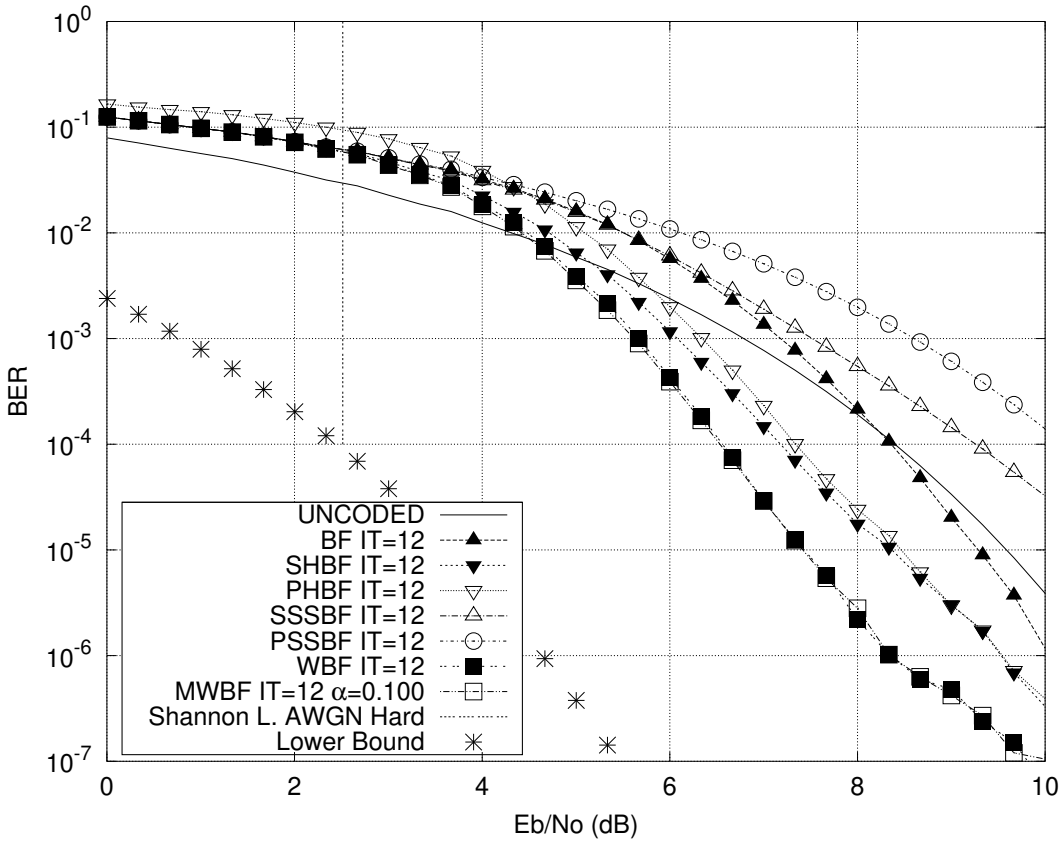

Fig. 4.16: Simulação dos algoritmos bit-flipping num canal $BI-AWGN$ com uma $LDGM$ de K=128 bits de informação, N=192 bits codificados, 12 iterações como máximo, $d_v^* = 9$ e $d_c^* = 18$.

Como pode-se ver na Figura 4.15 os algoritmos $SHBF$ e $PHBF$ já ficam muito próximos: não tinha muita necessidade de aumentar a quantidade de iterações máximas.

No gráfico da Figura 4.17 se usa a mesma taxa e a quantidade de uns por linha e coluna que a Figura 4.15 com a diferença que se incrementa a quantidade de bits de informação enviada. Pode-se notar no gráfico que o melhor desempenho é do algoritmo $PHBF$ seguido pelos algoritmos WBF e $MWBF$. Outro ponto interessante a notar é que os algoritmos $SHBF$ e WBF ficam superpostos, e ligeiramente distante de $PHBF$, o que quer dizer que a quantidade máxima de iterações empregadas na simulação não é o valor ótimo e pode ser acrescentado mais um pouco.

No gráfico da Figura 4.18 se observa o mesmo código que na Figura 4.17 com a diferença que se aumenta a quantidade de iterações máximas para 12. Isto faz com que os algoritmos $SHBF$, WBF

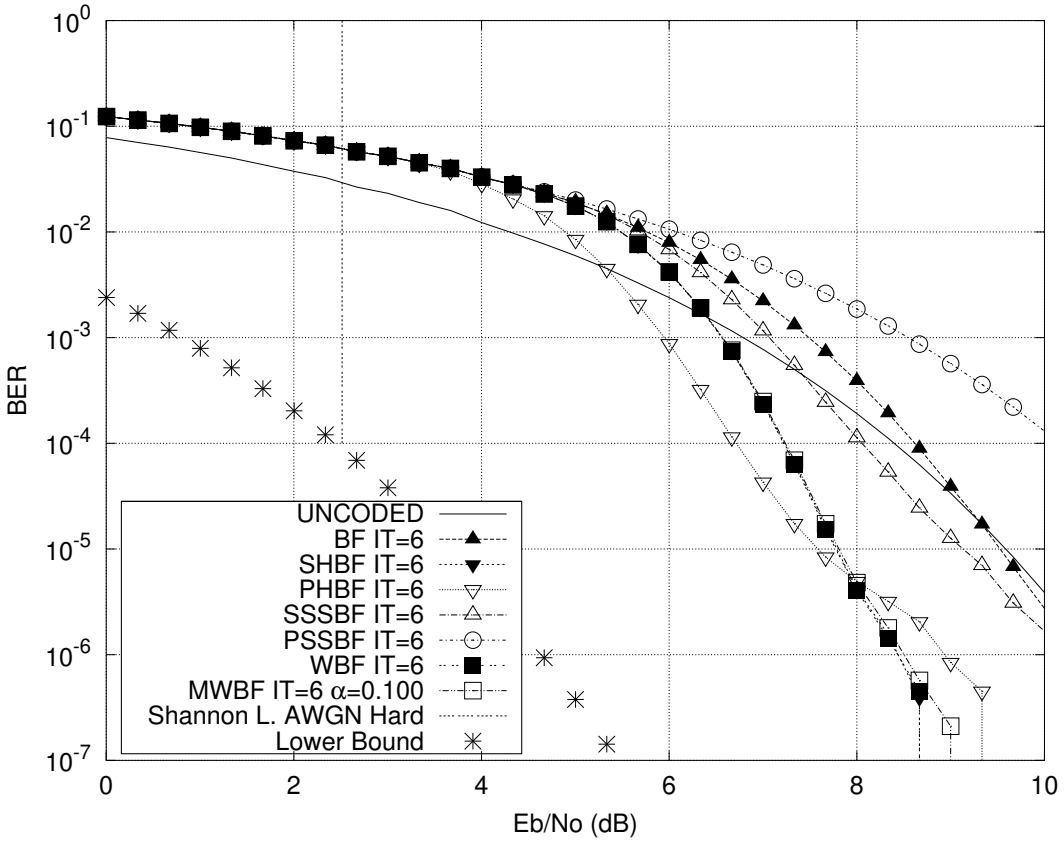

Fig. 4.17: Simulação dos algoritmos bit-flipping num canal $BI-AWGN$ com uma $LDGM$ de K=256 bits de informação, N=384 bits codificados, 6 iterações como máximo, $d_v^* = 9$ e $d_c^* = 18$.

e $MWBF$ fiquem com um desempenho melhor que o algoritmo de decodificação paralela $PHBF$. Isto indica que já se está próximo à quantidade de iterações máximas ótima para estes três algoritmos.

Até agora se tem trabalhado analisando de forma experimental o efeito da quantidade máxima de iterações sobre o desempenho da curva BER dos algoritmos de decodificação, usando uma quantidade de iterações máximas entre 6 e 12, para una quantidade de bits codificados N entre 192 e 384. Isto quer dizer que para o caso $N = 192$ se considera que como máximo nos algoritmos sequenciais se poderá reparar entre 3.125% $(6/192)$ e 6.25% $(12/192)$ dos bits. E para o caso $N = 384$ se considera que como máximo nos algoritmos sequenciais se poderá reparar entre 1.5625 % (6/384) e 3.125% dos bits. Isto indica um método de achar a "queda de água" nas curvas BER das distintas versões dos algoritmos BF sequenciais. Dado um E_b/N_0, chamado wf^*, que caracteriza o ponto onde acontece a queda de água da curva do BER dos algoritmos de decodificação sequencial, se acha a fracção de erros que podem ser corrigidos:

$$p_{wf} = \frac{IT}{N}, \tag{4.36}$$

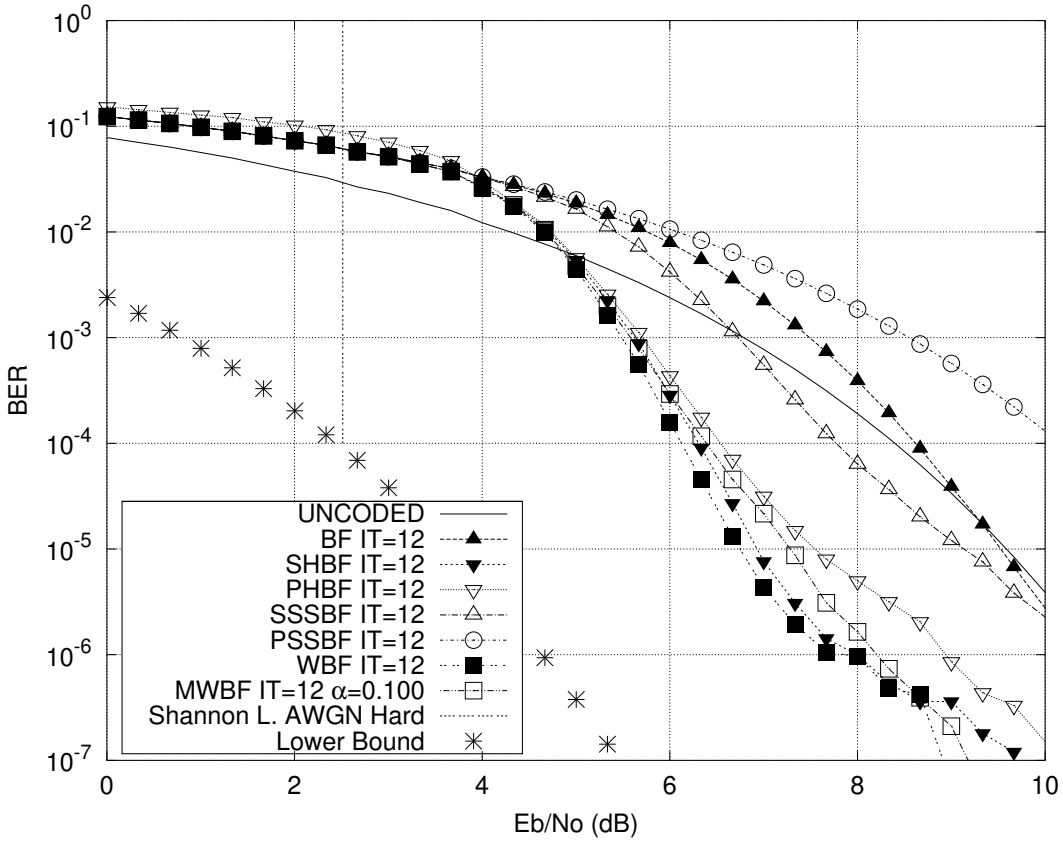

Fig. 4.18: Simulação dos algoritmos bit-flipping num canal $BI-AWGN$ com uma $LDGM$ de K=256 bits de informação, N=384 bits codificados, 12 iterações como máximo, $d_v^* = 9$ e $d_c^* = 18$.

e então se usa o comportamento de um sistema não codificado para estimar wf^*, isto é, p_{wf} em teoria é a probabilidade de erro de bit do canal, ou pelo menos a máxima que é possível de corrigir. Usando a equação (4.17) se obtém

$$wf^* = \frac{(Q^{-1}(p_{wf}))^2}{2R}, \tag{4.37}$$

onde $Q(x) = 0.5erfc(x/\sqrt{2})$ com $erfc(.)$ como a função de erro complementar.

Como exemplo pode-se ver na Figura 4.19, wf^* tem um valor aproximadamente igual a 6.1dB para o algoritmo de decodificação $SHBF$ e cumpre com a equação (4.37). Por outro lado na Figura 4.20, wf^* aplicando a equação (4.37) tem um valor igual a 0.9 dB para a decodificação $SHBF$. Este valor fica muito longe do valor mostrado na Figura 4.20. Isto pode explicar-se assumindo que existe um ponto de saturação, conclusão já obtida analisando o comportamento das curvas $SHBF$ e $PHBF$. É interessante notar que da simulação da Figura 4.20 pode obter-se experimentalmente o valor ótimo de IT. Nessa figura se indica que a queda de água acontece quando wf^* é aproximada-

mente 4.5dB. Usando este valor nas equações (4.37) e (4.36) o valor de IT é igual a 785. Este valor é o IT que otimiza o desempenho das curvas dos algoritmos sequenciais BF. Qualquer valor superior a este é desnecessário.

O gráfico da Figura 4.19 apresenta as comparações de todas as versões do algoritmos BF estudados nesta dissertação, incluindo a curva limite proposta por Javier Garcia-Frias e Wei Zhong [8], e o limite de Shannon para um canal BSC ou $BI - AWGN$ com decisão abrupta. É usado para a codificação de uma matriz $LDGM$ com $N = 30000$ bits codificados e $K = 20000$ bits de informação e taxa do código $R_c = 0.6667$. A matriz P da matriz sistemática foi gerada aleatoriamente tentando conservar constante só a quantidade de uns por linha, obtendo um $d_v^* = 9$. Para todos os algoritmos BF se usou uma quantidade máxima de iterações $IT = 300$.

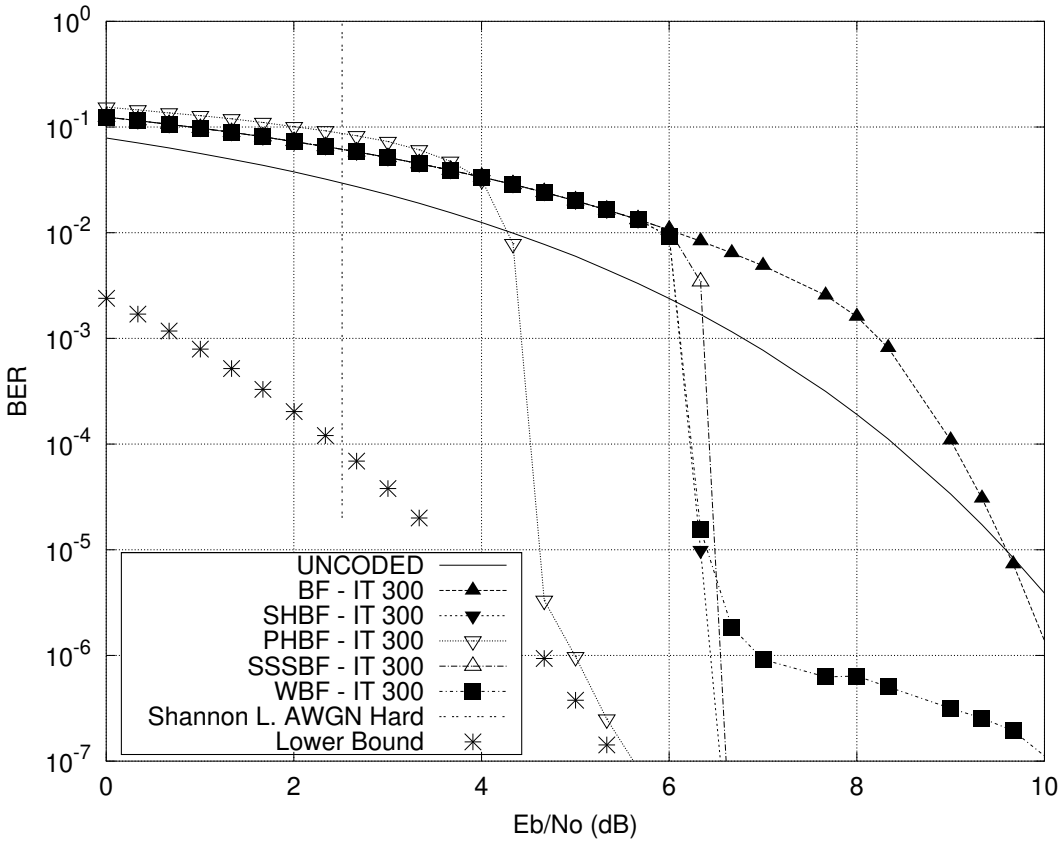

Fig. 4.19: Simulação dos algoritmos bit-flipping num canal $BI - AWGN$ com uma $LDGM$ de N=30000 bits de informação, K=20000 bits codificados, 300 iterações como máximo, $d_v^* = 9$ e $d_c^* = aleatorio$.

Na Figura 4.19 se mostra que o melhor desempenho, é obtido pela decodificação $PHBF$, e as decodificações $SHBF$ e WBF ficam longe deste valor para uma estimação de um 1% dos bits

errados como máximo por vetor codificado, ver equação (4.36). A Figura 4.20 é a mesma que a Figura 4.19 com a diferença que se usa uma estimação de que 10% dos bits no máximo estão errados em cada vetor codificado.

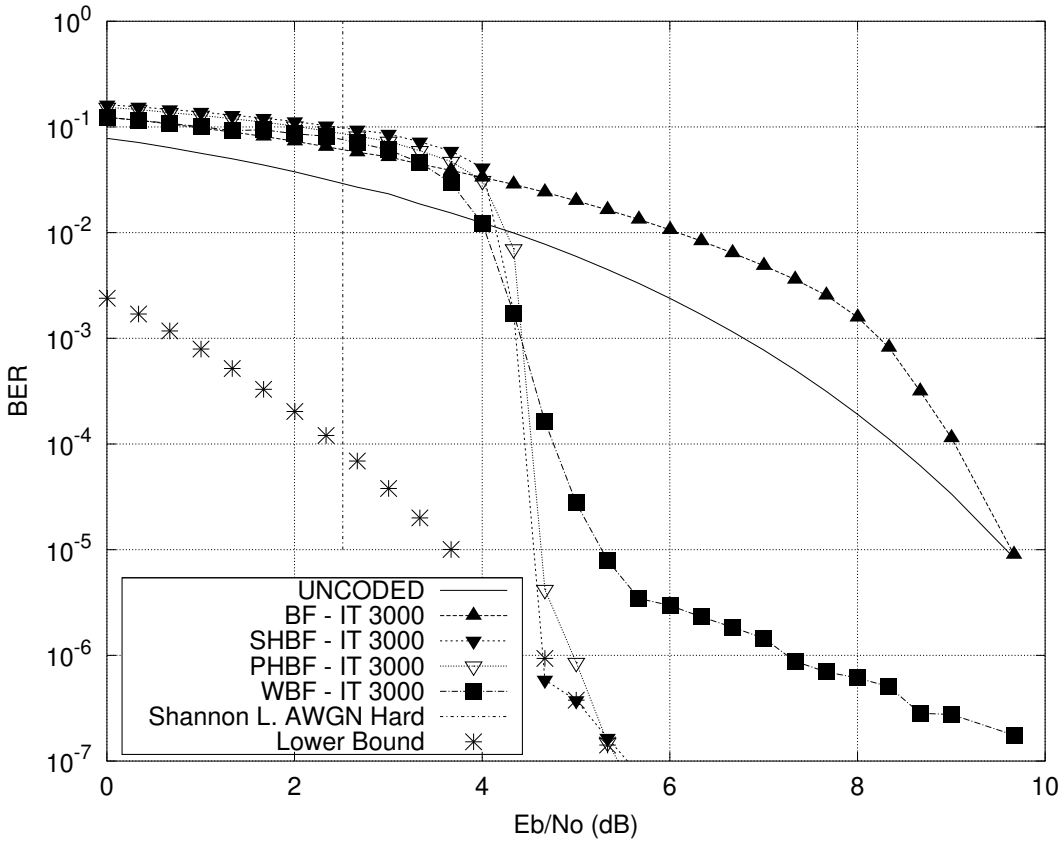

Fig. 4.20: Simulação dos algoritmos bit-flipping num canal $BI - AWGN$ com uma $LDGM$ de N=30000 bits de informação, K=20000 bits codificados, 3000 iterações como máximo, $d_v^* = 9$ e $d_c^* = aleatorio$.

O gráfico da Figura 4.21 apresenta as comparações de todas as versões do algoritmos BF e a curva limite de Javier Garcia-Frias e Wei Zhong, além do limite de Shannon para um canal $BI - AWGN$ com decisão abrupta. É usado para a codificação uma matriz $LDGM$ com $N = 306$ bits codificados e $K = 204$ bits de informação e a taxa do código $R_c = 0.6667$. A matriz P da matriz sistemática foi obtida do site de Mackay [14], com um $d_v^* = 5$ e um $d_c^* = 10$. Para todos os algoritmos BF se usou uma quantidade máxima de iterações $IT = 3$.

A Figura 4.21 mostra que o melhor desempenho foi da decodificação $PHBF$, e as decodificações $SHBF$ e WBF ficam longe deste valor, para uma aproximação de um 1% dos bits errados ($IT = 3$) como máximo por vetor codificado. Isto é igual ao caso da Figura 4.19 onde também se tem uma

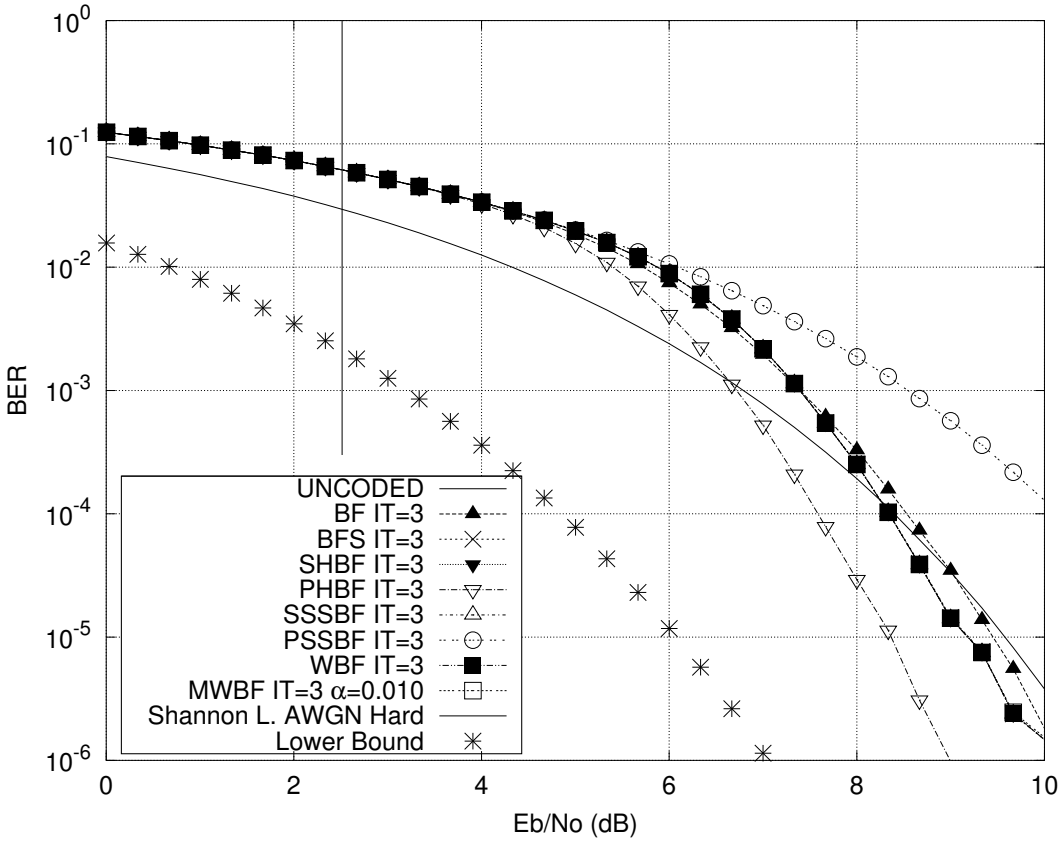

Fig. 4.21: Simulação dos algoritmos bit-flipping num canal $BI-AWGN$ com uma $LDGM$ de K=204 bits de informação, N=306 bits codificados, 3 iterações como máximo, $d_v^*=5$ e $d_c^*=10$.

estimação que um 1% dos bits errados ($IT=300$). A Figura 4.22 é a mesma que a Figura 4.21 com a diferença que se usa uma estimação de 15% dos bits estão errados como máximo em cada vetor codificado.

Pode-se ver que como na Figura 4.19 a tendência de vantagem do algoritmo $PHBF$, para um baixo número de iterações máximas, se conserva. A diferença fundamental é que por causa da diminuição do número de bits por palavra código, então o desempenho do algoritmo na curva do BER diminui também.

Um ponto de estudo importante é o comportamento dos algoritmos $bit-flipping$ num canal $BI-AWGN$, usando na codificação e decodificação códigos de bloco com matrizes $EG-LDPC$ de tipo I. Como já se tem visto na equação (4.35), é de esperar que os algoritmos BF e $PHBF$ tenham o mesmo desempenho. Isto pode ver-se na Figura 4.23, onde se usou um código de bloco gerado por uma $EG^*(m,2^s)$, com $m=2$, $s=2$ e um polinômio primitivo $p(X)=X^4+X+1$. O código de bloco assim obtido terá $K=175$ bits de informação, $N=255$ bits codificados e um

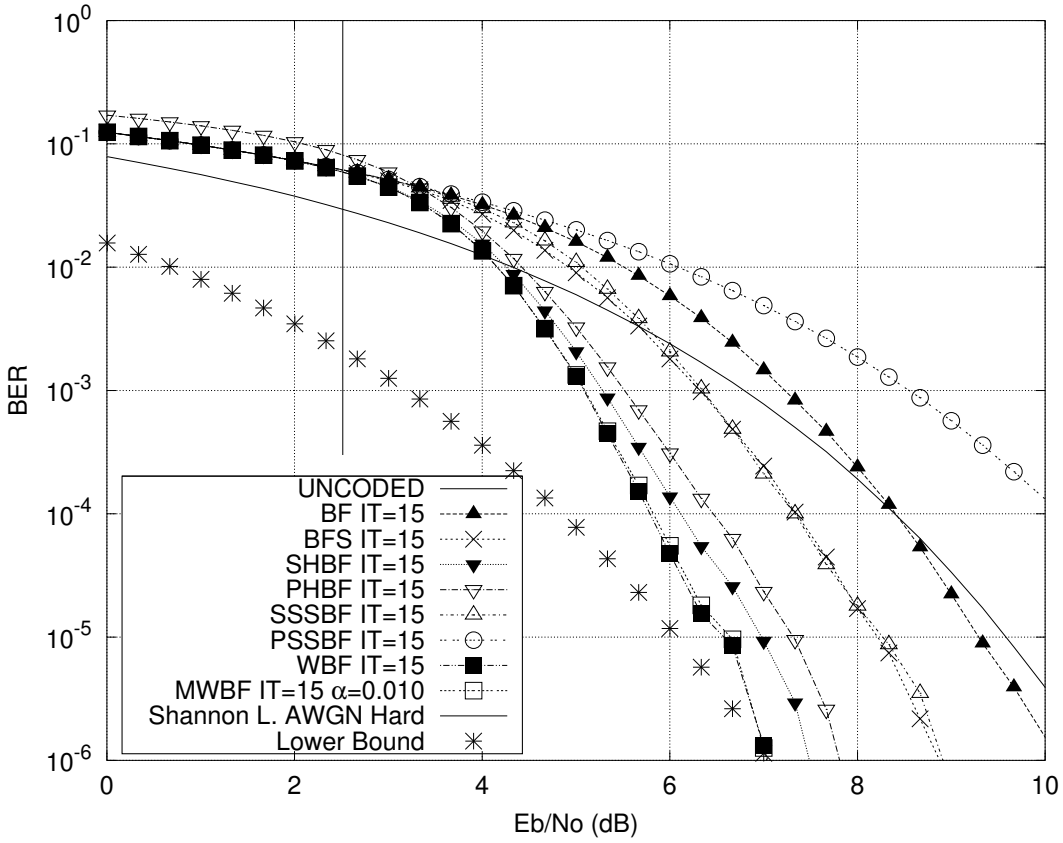

Fig. 4.22: Simulação dos algoritmos bit-flipping num canal $BI - AWGN$ com uma $LDGM$ de K=204 bits de informação, N=306 bits codificados, 15 iterações como máximo, $d_v^* = 5$ e $d_c^* = 10$.

$d_v = d_c = 16$. Para esta simulação se usaram uma quantidade $IT = 12$ iterações como máximo, correspondente a um 5% de N bits errados como máximo. Também pode-se ver como os algoritmos $SHBF$, WBF e $MWBF$ tem uma pequena melhora quanto comparados como BF e $PHBF$. Os algoritmos $SSSBF$ e $PSSBF$ tem o pior desempenho. Como se tem visto até agora os algoritmos seriais só são superiores aos algoritmos paralelos como BF, $PHBF$ e $PSSBF$, quando o número de iterações máximas é suficientemente grande, pelo qual pode-se intuir que é possível diminuir este número e conseguir um resultado ainda ótimo. Também se tem um desempenho melhor da curva do BER das decodificações bit-flipping, quanto comparados com os códigos de bloco $LDGM$ simulados até agora. Isto é, por causa de que a quantidade de uns por linha é quase o duplo que os códigos $LDGM$ já vistos. Esta característica é tipica das $EG - LDPC$

O gráfico da Figura 4.24 mostra o desempenho dos algoritmos bit-flipping para um código de bloco com uma matriz geradora e de verificação de paridade criada a partir de uma $EG(m, 2^s)$, com $m = 2$, $s = 5$ e usando o polinômio primitivo $p(X) = X^{10} + X^3 + 1$. O código terá $K = 781$,

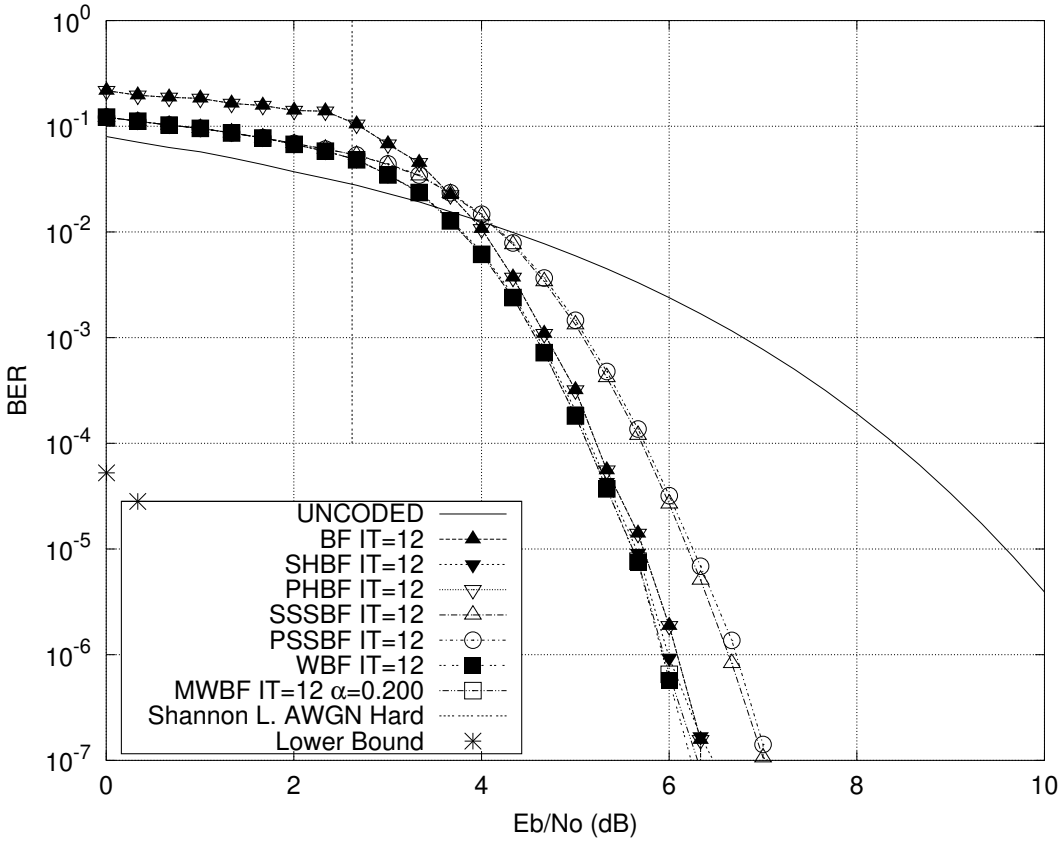

Fig. 4.23: Simulação dos algoritmos bit-flipping num canal $BI-AWGN$ com uma $EG-LDPC$ de tipo I, com K=175 bits de informação, N=255 bits codificados, 12 iterações como máximo, $d_v = 16$ e $d_c = 16$.

$N = 1023$ e $J = 1023$. Para a simulação se usa como máximo $IT = 50$ iterações, equivalente ao 5% dos N bits codificados. Este código tem um $d_v = d_c = 32$, número muito superior ao valor de 9 que foi usado nos códigos $LDGM$, isto se vê refletido no melhor desempenho de todas as curvas na simulação. É importante ressaltar que quanto maior seja o valor de d_v melhor desempenho terão os algoritmos de decodificação, mas também terão maior complexidade na decodificação.

Pelo visto até agora os algoritmos $PHBF$ e $SHBF$ tem um bom desempenho em matrizes $LDGM$ com um baixo grau de linha (d_v), superando em algumas situações a algoritmos com decisão suave. Isto não se conserva em matrizes $EG - LDPC$ por causa da regularidade na quantidade de uns por linha na matriz de verificação de paridade, o que origina que seu desempenho seja igual ao algoritmo BF, isto é o incremento na complexidade de $PHBF$ e $SHBF$ não outorga nenhuma ventagem.

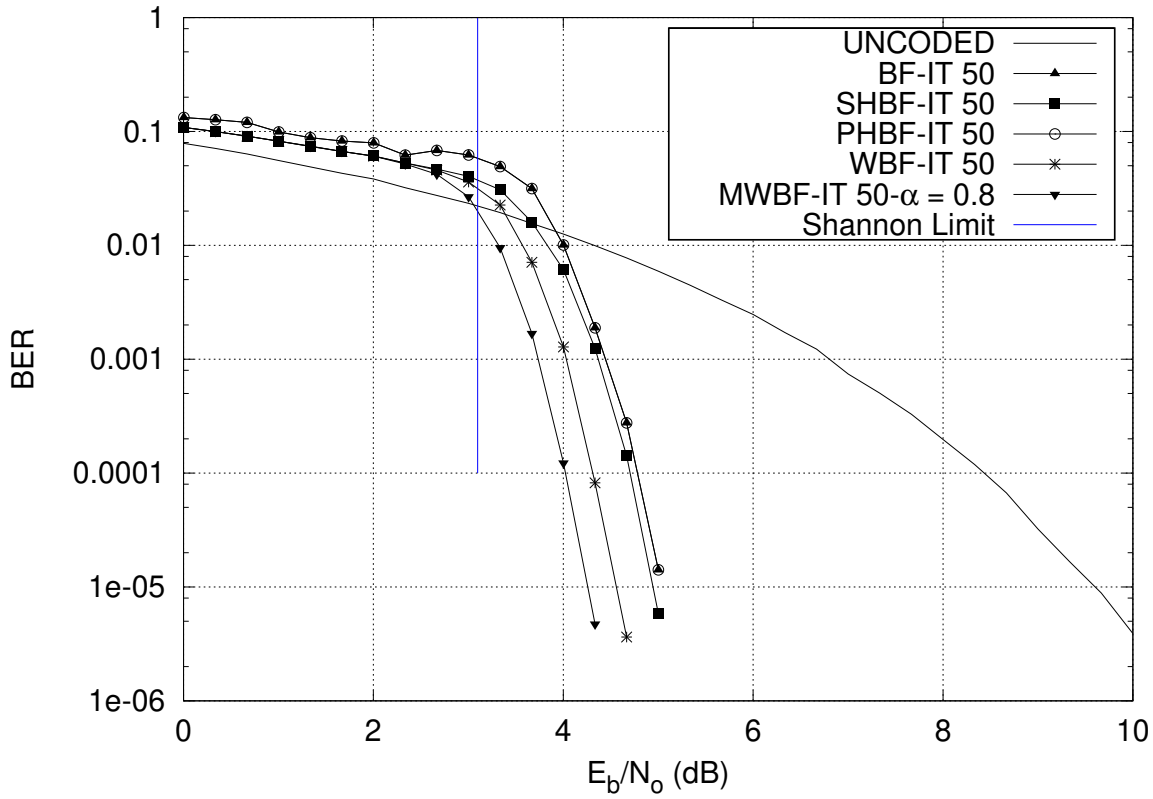

Fig. 4.24: Simulação dos algoritmos bit-flipping num canal $BI-AWGN$ com uma $EG-LDPC$ de tipo I, com K=781 bits de informação, N=1023 bits codificados, 50 iterações como máximo, $d_v = 32$ e $d_c = 32$.

Capítulo 5

Conclusões

Esta tese apresentou os algoritmos de decodificação $PHBF$ e $SHBF$ como modificações nas formas paralela e sequencial do algoritmo de decodificação BF, proposto por Gallager. Estas modificações também estão baseadas no método de geração de confiabilidade do algoritmo modificado BF proposto por Sipser e Spielman.

Para um comprimento moderado dos códigos $LDGM$ se observa que o algoritmo de decodificação $MWBF$ não apresenta nenhuma melhora significativa no comportamento quanto comparado com a decodificação $PHBF$. Isto pode ser atribuído ao fato de que estes códigos tem uma distribuição irregular no peso das colunas da matriz de verificação de paridade H, com $N - K$ colunas de peso 1 e K colunas de peso d_v, e o fator de ponderação da decodificação $MWBF$ atua com o mesmo valor sobre confiabilidades de distintos pesos por coluna de H.

Outro ponto importante é que os algoritmos de decodificação $SHBF$ e $PHBF$ apresentam um desempenho semelhante com diminuição da taxa de erro quando comparados ao algoritmo WBF, este último sendo mais custoso em complexidade computacional por causa de seus cálculos em ponto flutuante.

É interessante ressaltar que se apresentou uma forma de predizer a "queda de água" na curva da taxa de erro para os algoritmos de decodificação sequencial.

Fica para futuros trabalhos investigar se existe uma maneira de decodificação $MWBF$ com melhor desempenho quando comparado ao WBF e $PHBF$ para matrizes $LDGM$ sistemáticas.

Referências Bibliográficas

[1] R. Gallager. Low-Density Parity-Check Codes. *Cambridge,Massachusetts: MIT Press*, 1963.

[2] Michael Sipser and Daniel A. Spielman. Expander Codes. *IEEE Trans. Inform. Theory*, 42(6):1710–1722, Novembro 1996.

[3] Yu Kou, Shu Lin, and Marc P. C. Fossorier. Low-density parity-check codes based on finite geometries: A rediscovery and new results. *IEEE Trans. Inform. Theory*, 47(7):2711–2736, Novembro 2001.

[4] Juntan Zhang and Marc P. C. Fossorier. A Modified Weighted Bit-Flipping Decoding of Low-Density Parity-Check Codes. *IEEE Commun. Lett.*, 8(3):165–167, Março 2004.

[5] Shu Lin, Daniel J. Costello. *Error Control Coding*. Pearson Prentice Hall, 2004.

[6] D. J. C. MacKay. Good error correcting codes based on very sparse matrices. *IEEE Trans. Info. Theory*, 45:399–431, Março 1999.

[7] Jung-fu Cheng and Robert J. Mceliece. Frequency-Efficient Coding with Low-Density Generator Matrices. *IEEE International Symposium on Information Theory,Ulm. Germany*, Julho 1997.

[8] J. F. Garcia and W. Zhong. Approaching Shannon performance by iterative decoding of linear codes with low-density generator matrix. *IEEE Commun. Lett.*, 7(6):266–268, Junho 2003.

[9] William E. Ryan, Shu Lin. *Channel Codes Classical and Modern*. Cambridge University Press, 2009.

[10] D. J. C. MacKay and R. M. Neal. Near Shannon limit performance of low density parity check codes. *Electronics Letters*, 32(18):1645–1646, Agosto 1996. Reprinted *Electronics Letters*, **33**(6):457–458, Março 1997.

[11] D. J. C. MacKay and R. M. Neal. Good codes based on very sparse matrices. In Colin Boyd, editor, *Cryptography and Coding. 5th IMA Conference*, number 1025 in Lecture Notes in Computer Science, pages 100–111. Springer, Berlin, 1995.

[12] Brendan J. Frey. *Graphical models for machine learning and digital communication*. MIT Press, 1998.

[13] Ming Jiang, Chunming Zhao, Zhihua Shi, and Yu Chen. An Improvement on the Modified Weighted Bit-Flipping Decoding Algorithm for LDPC Codes. *IEEE Commun. Lett.*, 9(9):814–816, Setembro 2005.

[14] D. J. C. MacKay. MacKay's Homepage. Página na internet, University of Cambridge., Março 2006. `http://www.inference.phy.cam.ac.uk/mackay/codes/`.

[15] Fernando Pujaico Rivera. Pagina pessoal. Página na internet, Universidade Estadual de Campinas., Março 2011. `http://www.decom.fee.unicamp.br/~fpujaico/descargas.html`.

Apêndice A

Apêndice A: Representação de um $GF(2^m)$

A.1 Campos de Galois

Um campo finito ou um campo de Galois ou simplesmente GF (do inglês $Galois\ Field$), é um campo que tem um número finito de elementos. Um campo está definido como:

DEFINIÇÃO A.1.1 *Um conjunto $\wp$ onde as operações binárias de soma $\boxplus$ e multiplicação $\boxdot$ são definidas, é chamado de $campo$ se as seguintes condições foram satisfeitas para a, b e c que pertencem a $\wp$.*

i) As operações binárias $\boxplus$ e $\boxdot$ são associativas:

$$a \boxplus (b \boxplus c) = (a \boxplus b) \boxplus c \quad \wedge \quad a \boxdot (b \boxdot c) = (a \boxdot b) \boxdot c$$

ii) As operações binárias $\boxplus$ e $\boxdot$ são comutativas:

$$a \boxplus b = b \boxplus a \quad \wedge \quad a \boxdot b = b \boxdot a$$

iii) Tem um elemento identidade para as operações binárias $\boxplus$ e $\boxdot$

$$a \boxplus e_{\boxplus} = e_{\boxplus} \boxplus a = a \quad \wedge \quad b \boxdot e_{\boxdot} = e_{\boxdot} \boxdot b = b$$

iv) Tem um elemento inverso a' e b' para as operações binárias $\boxplus$ e $\boxdot$

$$a \boxplus a' = a' \boxplus a = e_{\boxplus} \quad \wedge \quad b \boxdot b' = b' \boxdot b = e_{\boxdot}$$

Para cada $b \neq 0$.

v) Existe distributividade de $\boxdot$ com relação a $\boxplus$:

$$a \boxdot (b \boxplus c) = (a \boxdot b) \ \boxplus \ (a \boxdot c)$$

EXEMPLO A.1.1 *Dado o conjunto finito $\wp = \{0, 1\}$ e definidas as operações binárias $\oplus$ (OR exclusivo) e $\cdot$ (AND) obtemos as Tabelas 3.1 e 3.2, vemos que elas cumprem os pontos (i), (ii), (iii) (iv) e (v) da definição A.1.1 pelo qual o conjunto $\wp$ e os operadores $\oplus$ e $\cdot$ formam um campo de ordem 2 (finito), pelo qual é um campo de Galois ($GF(2)$).*

EXEMPLO A.1.2 *Dado o conjunto finito $\wp = \{0, 1, 2, 3, 4\}$ e definidas as operações binárias $\boxplus$ (soma módulo 5) e $\boxdot$ (multiplicação módulo 5) obtemos as Tabelas 3.3 e 3.4, vemos que elas cumprem os pontos (i), (ii), (iii) (iv) e (v) da definição A.1.1 pelo qual o conjunto $\wp$ e os operadores $\boxplus$ e $\boxdot$ formam um campo de ordem 5 (finito), pelo qual é um campo de Galois ($GF(5)$).*

Assim vemos que existe um $GF(p)$ para qualquer primo p, onde p representa o ordem do campo, um caso particular de estudo é o $GF(2)$ que é o campo de Galois do exemplo A.1.1, onde as operações de soma e multiplicação são binarias. Alem dos campos do tipo $GF(2)$, também são motivo de estudo os campos do tipo $GF(2^m)$, que é o campo de extensão de $GF(2)$. Uma $GF(2^m)$ consta dos seguintes elementos:

$$GF(2^m) = \left\{0, 1, \alpha, \alpha^2, \alpha^3, \cdots, \alpha^{2^m-2}\right\}. \tag{A.1}$$

No $GF(2^m)$ as operações binárias de soma e multiplicação são em módulo $p(x)$, onde $p(x)$ é um polinômio irredutível com coeficientes em $GF(2)$ e α é uma raiz de $p(x)$ é dizer $p(\alpha) = 0$, $p(x)$ recebe o nome de polinômio primitivo.

TEOREMA A.1.1 *Um polinômio $p(x)$ de grau m é irredutível se não é divisível por nenhum polinômio de grau major que 0 e menor do que m.*

TEOREMA A.1.2 *Um polinômio $p(x)$ em $GF(2)$ de grau m é chamado de $primitivo$ se o menor dos inteiros positivos n para $p(x)$ que divide a $x^n + 1$ é $2^m - 1$, como se vê na seguinte equação*

$$x^{2^m-1} + 1 = q(x)p(x). \tag{A.2}$$

O fato de que α seja uma raiz de $p(x)$ é consequência de que $p(x)$ seja um polinômio primitivo e que o campo $GF(2^m)$ seja finito e fechado para a multiplicação. Avaluando α em (A.2) obtemos

$$\alpha^{2^m-1} + 1 = q(\alpha)0, \tag{A.3}$$

$$\alpha^{2^m-1} = 1. \tag{A.4}$$

EXEMPLO A.1.3 *Dado o conjunto finito $\wp = \{0, 1, \alpha, \alpha^2\}$ e definidas as operações binárias soma $\boxplus$ e multiplicação $\boxdot$ em módulo $p(x)$, para um $p(x) = x^2 + x + 1$ obtemos as Tabelas A.1 e A.2, vemos que elas cumprem a definição A.1.1, pelo qual o conjunto $\wp$ e os operadores $\boxplus$ e $\boxdot$ formam um campo de Galois $GF(2^2)$.*

$\boxplus$	0	1	α	α^2
0	0	1	α	1+α
1	1	0	1+α	α
α	α	1+α	0	1
α^2	1+α	α	1	0

Tab. A.1: Operação binária soma $\boxplus$ em módulo $p(x)$ sobre o conjunto $\wp = \{0, 1, \alpha, \alpha^2\}$.

$\boxdot$	1	α	α^2
1	1	α	1+α
α	α	1+α	1
α^2	1+α	1	α

Tab. A.2: Operação binária multiplicação $\boxdot$ em módulo $p(x)$ sobre o conjunto $\wp = \{1, \alpha, \alpha^2\}$.

Os cálculos do resíduo feitos nas Tabelas A.1 e A.2 podem ser facilitados se usamos distintas representações para cada um dos elementos de $GF(2^2)$, por exemplo se usamos uma representação polinomial. Representando a cada α^i de $GF(2^m)$, $i \leq 2^m - 2$, mediante um polinômio $f_i(\alpha)$ de grau $m - 1$ da forma $f_i(\alpha) = a_{i0} + a_{i1}\alpha + a_{i2}\alpha^2 + a_{i3}\alpha^3 + \cdots + a_{i(m-1)}\alpha^{m-1}$, onde $a_{ij} \in GF(2)$ e $j \leq m - 1$, obtemos a Tabela A.3.

Representando os α^i como $f_i(\alpha)$ bastaria com realizar somas e multiplicações sobre os polinômios $f_i(\alpha)$ mediante operações em $GF(2)$, também ajuda muito à simplificação se ajudar usando a propriedade $p(\alpha) = 0$.

Representação em potências	Representação polinomial	Representação vetorial
0	0	$(0, 0)$
1	1	$(1, 0)$
α	α	$(0, 1)$
α^2	$1 + \alpha$	$(1, 1)$

Tab. A.3: Três representações para os elementos de $GF(2^2)$ gerado por $p(x) = x^2 + x + 1$.

Alem da representação polinomial pode-se optar por trabalhar com uma representação vetorial, que não é outra coisa que puxar os coeficientes a_{ij} de $f_i(\alpha)$ e gerar uma $m-upla$, $\alpha^i = (a_{i0}, a_{i1}, a_{i2}, \cdots, a_{i(m-1)})$, como se vê nas Tabelas A.3 e A.4.

Representação em potências	Representação polinomial	Representação vetorial
0	0	$(0,0,0,0)$
1	1	$(1,0,0,0)$
α	α	$(0,1,0,0)$
α^2	α^2	$(0,0,1,0)$
α^3	α^3	$(0,0,0,1)$
α^4	1+α	$(1,1,0,0)$
α^5	α+α^2	$(0,1,1,0)$
α^6	α^2+α^3	$(0,0,1,1)$
α^7	1+α+α^3	$(1,1,0,1)$
α^8	1+α^2	$(1,0,1,0)$
α^9	α+α^3	$(0,1,0,1)$
α^{10}	1+α+α^2	$(1,1,1,0)$
α^{11}	α+α^2+α^3	$(0,1,1,1)$
α^{12}	1+α+α^2+α^3	$(1,1,1,1)$
α^{13}	1+α^2+α^3	$(1,0,1,1)$
α^{14}	1+α^3	$(1,0,0,1)$

Tab. A.4: Três representações para os elementos de $GF(2^4)$ gerado por $p(x) = x^4 + x + 1$.

A.2 Algoritmo para a representação de $GF(2^m)$ na forma polinomial

Usando o Algoritmo 6 é possível gerar as Tabelas A.3 e A.4, estas tem uma representação polinomial com coeficientes em $GF(2)$. Em geral este algoritmo serve para procurar a representação polinomial $f_i(\alpha) = a_{i0} + a_{i1}\alpha + a_{i2}\alpha^2 + a_{i3}\alpha^3 + \cdots + a_{i(m-1)}\alpha^{m-1}$, $a_{ij} \in GF(2)$, de cada elemento $\alpha^i \in GF(2^m)$ com polinômio primitivo $p(x)$ de grau m.

Algoritmo 6 Algoritmo para a representação de $GF(2^m)$ na forma polinomial em $GF(2)$.

Entrada: $i = 0$, $f_0(\alpha) = 1$, $p(\alpha)$ e $m \leftarrow Grau\{p(\alpha)\}$.
Saida: $f_i(\alpha)\ \forall\ 0 \leq i \leq 2^m - 2$.

```
repeat
    i ← i + 1
    m_f ← Grau{f_i(α)α}
    f_i(α) ← f_i(α)α
    if m_f == m then
        f_i(α) ← f_i(α) ⊕ p(α)
    end if
until i ≠ 2^m − 2
return f_i(α) ∀ 0 ≤ i ≤ 2^m − 2
```

A.3 Algoritmo para a representação de $GF(2^m)$ na forma vetorial

Usando o Algoritmo 7 é possível gerar as Tabelas A.3 e A.4 na sua representação vetorial com elementos em $GF(2)$, em geral este algoritmo serve para procurar a representação vetorial

$$A = (a_{i0}, a_{i1}, a_{i2}, a_{i3}, \cdots, a_{i(m-1)}),\ a_{ij} \in GF(2) \tag{A.5}$$

de cada elemento $\alpha^i \in GF(2^m)$ com polinômio primitivo $p(x)$ de grau m. Para isso é importante definir A_i^* como um vetor de grau $m + 1$.

$$A_i^* = (a_{i0}^*, a_{i1}^*, a_{i2}^*, a_{i3}^*, \cdots, a_{i(m-1)}^*, a_{im}^*),\ a_{ij}^* \in GF(2). \tag{A.6}$$

P é a representação vetorial do polinômio $p(\alpha)$

$$p(\alpha) = b_{i0} + b_{i1}\alpha + b_{i2}\alpha^2 + b_{i3}\alpha^3 + \cdots + b_{im}\alpha^m,\ b_{ij} \in GF(2), \tag{A.7}$$

$$P = (b_{i0}, b_{i1}, b_{i2}, b_{i3}, \cdots, b_{im}). \tag{A.8}$$

A função $Corrimiento1BitDireita\{X\}$ faz um corrimento de bits à direita de X, e o último bit da direita é perdido neste processo. A função $PrimeirosBits\{X\}$ copia os primeiros m bits de X a outro vetor.

Algoritmo 7 Algoritmo para a representação de $GF(2^m)$ na forma vetorial em $GF(2)$.

Entrada: $i = 0$, $A_0^* = (1, 0, 0, \cdots, 0)$, P .
Saida: $A_i \ \forall \ 0 \leq i \leq 2^m - 2$.

1: **repeat**
2: $i \leftarrow i + 1$
3: $A_i^* \leftarrow Corrimiento1BitDireita\,\{A_i^*\}$
4: **if** $a_{im}^* == 1$ **then**
5: $A_i^* \leftarrow A_i^* \oplus P$
6: **end if**
7: $A_i \leftarrow PrimeirosBits\,\{A_i^*\}$
8: **until** $i \neq 2^m - 2$
9: **return** $A_i \, \forall \ 0 \leq i \leq 2^m - 2$.

Apêndice B

Apêndice B: Programas e formatos usados nas simulações

B.1 Formato Alist

O formato $Alist$ foi desenvolvido por David MacKay, Matthew Davey e John Lafferty para descrever mediante um arquivo de texto plano uma matriz de verificação de paridade de baixa densidade ($LDPC$). A Definição B.1.1 descreve linha a linha um arquivo em formato $Alist$.

DEFINIÇÃO B.1.1 (FORMATO ALIST) .

- *A primeira linha do arquivo contem o número de linhas (L) e colunas (C) da matriz $LDPC$.*
- *A segunda linha do arquivo contem o peso máximo de todas as linhas (P_{lmax}) e todas as colunas (P_{cmax}) da matriz. Para obter o peso máximo de todas as linhas, se calcula primeiro o peso de cada linha, contando a quantidade de uns por linha, e logo se procura o máximo peso achado em todas as linhas. Para obter o peso máximo de todas as colunas se procede da mesma maneira.*
- *A terceira linha do arquivo tem o peso (quantidade de uns) de todas as L linhas da matriz, iniciando pela esquerda com a primeira linha.*
- *A quarta linha do arquivo tem o peso (quantidade de uns) de todas as C colunas da matriz, iniciando pela esquerda com a primeira coluna.*
- *As seguintes L linhas do arquivo contem as posições dos uns das L linhas da matriz, sendo que cada linha do arquivo descreverá as posições dos uns de cada linha da matriz. Cada uma das linhas do arquivo sempre terão P_{lmax} elementos. Se a quantidade de uns por linha for*

inferior a P_{lmax} se enche com zeros para obter sempre P_{lmax} elementos por linha do arquivo. O primeiro bit "1" da esquerda da linha da matriz tem a posição 1, o ultimo bit "1" da direita da linha da matriz tem a posição C.

- *As seguintes C linhas do arquivo contem as posições dos uns das C colunas da matriz, sendo que cada linha do arquivo descreverá as posições dos uns de cada coluna da matriz. Cada uma das linhas do arquivo sempre terão P_{cmax} elementos, se a quantidade de uns por coluna for inferior a P_{cmax} se enche com zeros para obter sempre P_{cmax} elementos por linha do arquivo. O primeiro bit "1" da parte superior da coluna da matriz tem a posição 1, o ultimo bit "1" da parte inferior da coluna da matriz tem a posição L.*

EXEMPLO B.1.1 *Seguindo os passos descritos na Definição B.1.1 pode-se converter a matriz (B.1) no formato $Alist$, obtendo assim o Arquivo B.1.1.*

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{B.1}$$

ARQUIVO B.1.1 (MATRIZ B.1 EM FORMATO ALIST.) .

```
12 16
4 3
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 8 10 13
4 7 9 13
2 5 7 10
4 6 11 14
3 9 15 16
```

```
1 6 9 10
4 8 12 15
2 6 12 16
1 7 14 16
3 5 12 14
2 11 13 15
1 5 8 11
6 9 12
3 8 11
1 5 10
2 4 7
3 10 12
4 6 8
2 3 9
1 7 12
2 5 6
1 3 6
4 11 12
7 8 10
1 2 11
4 9 10
5 7 11
5 8 9
```

B.2 bf-tests-awgn-ldgm

O *bf-tests-awgn-ldgm* é um programa para a simulação dos algoritmos de decodificação: *bit-flipping*, *serial bit-flipping*, *serial hard bit-flipping*, *parallel hard bit-flipping*, *sequencial Sipser Spielman bit-flipping*, *parallel Sipser Spielman bit-flipping*, *weighted bit-flipping* e *modified weighted bit-flipping*. O programa usa como modelo de canal um canal $BI - AWGN$. Usando o canal $BI - AWGN$ de decisão abrupta para simular os algoritmos *bit-flipping*, *serial bit-flipping*, *serial hard bit-flipping*, *parallel hard bit-flipping*, *sequencial Sipser Spielman bit-flipping* e *parallel Sipser Spielman bit-flipping*, e um canal $BI-AWGN$ de decisão suave para simular os algoritmos *weighted bit-flipping* e *modified weighted bit-flipping*. O programa simula os algoritmos de decodificação usando para a codificação e decodificação códigos com matrizes $LDGM$.

Durante toda a simulação o programa *bf-tests-awgn-ldgm* escreve sobre dois arquivos de saída, o primeiro é "ArquivoOut" (–out ArquivoOut) que é o arquivo com o registro da taxa de erro do bit (BER) para todos o algoritmos de decodificação habilitados na simulação. As primeiras 4 colunas de "ArquivoOut" correspondem à :

- Probabilidade de erro do bit do canal (P_b), este valor é ideal pois dependendo de que tão bem esteja implementada a função aleatória usada para gerar o canal, melhor aproximação terá a simulação.
- Probabilidade de erro do bit do canal (BER não codificado), este valor é o mesmo que o anterior coma diferença que foi obtido experimentalmente contando a quantidade de bits errados nas iterações.
- Probabilidade de erro do pacote, este valor é ideal e se obteve mediante formula usando P_b.
- Probabilidade de erro do pacote, este valor é experimental e se obteve contando a quantidade de pacotes errados nas iterações.

As seguintes colunas são ocupadas pelo BER apos aplicar os algoritmos *bit-flipping*, respeitando o seguinte ordem de esquerda a direita: *bit-flipping*, *serial bit-flipping*, *serial hard bit-flipping*, *parallel hard bit-flipping*, *sequencial Sipser Spielman bit-flipping*, *parallel Sipser Spielman bit-flipping*, *weighted bit-flipping* e *modified weighted bit-flipping*. Se algum dos algoritmos não está habilitado então a coluna desaparece e o algoritmo da direita ocupa seu lugar. O segundo arquivo que é escrito constantemente pelo programa *bf-tests-awgn-ldgm* é "ArquivoLog" (–log ArquivoLog), que é um arquivo onde se registra o desempenho do simulador e seu progresso em cada iteração. Alem de estos dois arquivos o programa gera no inicio todos os arquivos necessários para gerar um gráfico em $OCTAVE$, e com um pouco de retoque pode ser usado em $MATLAB$. Algo interessante a tomar em conta na simulação é definir o BER mínimo (–ber-min BERminimo) que cancelará a execução do algoritmo (não do programa) quando atingir um valore do BER inferior. Outro parâmetro importante na simulação é a quantidade de pacotes enviados (–pack-max Pacotes), esta quantidade de pacotes estabelece o valor do BER confiável, para o qual se considera que qualquer valor do BER superior a $BER_{confiável}$, corresponde a um valor com uma simulação confiável. Se consideramos que o código tem N bits codificados e são enviados M_{pack} pacotes como máximo, o $BER_{confiável}$ fica expressado como segue

$$BER_{confiável} = \frac{100}{N\ M_{pack}} \tag{B.2}$$

O Programa ao inicio mostra o BER mínimo e o BER confiável. Deve-se verificar antes de continuar com a simulação que o BER mínimo seja maior ou igual ao BER confiável.

PROGRAMA B.2.1 (USO DO PROGRAMA BF-TESTS-AWGN-LDGM)

bf−tests−awgn−ldgm *[−v][−−file Arquivo] [−−log ArquivoLog] [−−out ArquivoOut] [−−it−por porcentagem]*

[−−bf] [−−bfs] [−−shbf] [−−phbf] [−−sssbf]
[−−pssbf] [−−wbf] [−−mwbf] [−−alpha fator]
[−−ber−min BERminimo] [−−pack−max Paquetes]

-v *Retorna a versão atual do programa.*

–file Arquivo *Onde "Arquivo"é o nome do arquivo que contem à matriz de paridade P, no formato AList de MacKay. http://www.inference.phy.cam.ac.uk/mackay/codes/alist.html*

–log ArquivoLog *Onde "ArquivoLog"é o nome do arquivo do log que contem os dados estatísticos das iterações, por defeito se não se específica este parâmetro se agrega um ".log" ao nome dado em –file.*

–out ArquivoOut *Onde "ArquivoOut"é o nome do arquivo de saída que contém os dados das iterações na simulação, por defeito se não se específica este parâmetro se agrega um ".bf" ao nome dado em –file.*

–it-por porcentagem *Onde "porcentagem"é o porcentagem de N que será usado como quantidade de iterações máximas (IT). Se não se especifica o valor é: 1.0e-02 equivalente ao 1% dos N bits da palavra código codificada.*

–bf *Ativo o algoritmo bit-flipping.*

–bfs *Ativo o algoritmo bit-flipping serial.*

–shbf *Ativo o algoritmo serial hard bit-flipping.*

–phbf *Ativo o algoritmo parallel hard bit-flipping.*

–sssbf *Ativo o algoritmo sequencial Sipser Spielman bit-flipping.*

–pssbf *Ativo o algoritmo parallel Sipser Spielman bit-flipping.*

–wbf *Ativo o algoritmo weighted bit-flipping.*

–mwbf *Ativo o algoritmo modified weighted bit-flipping.*

–alpha fator *O "fator"é o α usado no algoritmo mwbf. Se não se especifica, o valor é: 1.0e-01*

–ber-min BERminimo *Onde "BERminimo"é o valor mínimo que tomará o BER dos algoritmos BF. Depois deste valor o algoritmo se desativa. O valor por defeito é: 1.0e-07*

–pack-max Pacotes *Onde "Pacotes"é a quantidade máxima de pacotes enviados numa iteração, este dado está ligado à confiabilidade do BER mínimo achado na iteração. Todo valor de BER menor que 100/(N*Pacotes) é desconfiável dado que no teve a suficiente quantidade de iterações, o critério é que para uma probabilidade P_r se tem que fazer 100/P_r probas para obter uma probabilidade experimental confiável, o valor de por defeito "Pacotes" é: 70000*

EXEMPLO B.2.1 *O seguinte comando simula os algoritmos bit-flipping (–bf), parallel hard bit-flipping (–phbf) e modified weighted bit-flipping (–mwbf), a partir da matriz de paridade P em MatrizEmAlist.dat. O algoritmo weighted bit-flipping usa um $\alpha = 0.1$ e a matriz P tem K linhas e M colunas, a partir dela obtemos nossa matriz H de N colunas. Todos os algoritmos bit-flipping usam como máximo $IT = 0.03333 * N$ iterações, os algoritmos são desativados sim se obtém na sua saída um BER menor que $1.0e-7$ e como máximo para cada P_b para sua simulação se envia 520833 pacotes (vetores codificados de N bits por palavra código).*

```
bf-tests-awgn-ldgm --file MatrizEmAlist.dat --it-por 0.03333
--bf --phbf --mwbf --alpha 0.1 --ber-min 1.0e-7 --pack-max 520833
```

Se pode seguir em todo momento a simulação usando $OCTAVE$ aplicando o comando:

```
simula1
```

B.3 parity-matrix-alist-format

O programa *parity-matrix-alist-format* gera uma matriz de K linhas e $M = N - K$ colunas com vários métodos, tem que se lhe indicar ao programa a quantidade de uns em cada linha de K, esta matriz pode ser usada como matriz $P = P_{KxM}$ de uma matriz G ($LDGM$), $G = [P_{KxM} I_{KxK}]$, $V_{1xN} = U_{1xK} * G_{KxN}$.

PROGRAMA B.3.1 (USO DO PROGRAMA PARITY-MATRIX-ALIST-FORMAT) .

parity−matrix−alist−format *[−v][−k NumK][−m NumNK][−−ones NumUns]*
[−−semilla Semente][−−intentos Intentos]
[−−out Nome][−−modelo Modelo]

-v *Retorna o nome e versão atual do programa.*

-k NumK *Onde "NumK" é a quantidade de bits do vector de informação, ou que é o mesmo a quantidade de linhas da matriz gerada.*

-m NumNK *Onde "NumNK" é a quantidade de bits de verificação de paridade, ou que é o mesmo a quantidade de colunas da matriz gerada.*

–ones NumUns *Onde "NumUns" é a quantidade de uns em cada uma das K linhas. A quantidade de uns por coluna dependerá do parâmetro $--modelo$.*

–semilla Semente *Onde "Semente" é o número que se usará como semente na geração da matriz aleatória P. Por defeito: 0*

–intentos Intentos *Dado que a matriz se gera aleatoriamente, "Intentos" é a quantidade de intentos que se farão em cada linha para conseguir NumUns*K/M uns por coluna na matriz. Por defeito: 100*

–out Nome *Onde "Nome" é o nome do arquivo que contem à matriz de paridade P no formato AList de Mackay. http://www.inference.phy.cam.ac.uk/mackay/codes/alist.html. por defeito usa-se os dados de configuração para gerar o Nome.*

–modelo Modelo *Onde "Modelo" é o número de modelo para a geração da matriz, o modelo por defeito é 0. Os modelos podem ser:*

0 *Gera uma matriz com "NumUns" uns em posições aleatórias uniformemente distribuídas em cada fila, e intenta manter constante a quantidade de uns nas colunas. Os uns em cada coluna são aproximadamente: NumUns*K/M.*

1 *Gera uma matriz da forma:*
111000...0
011100...0
001110...0

2 *Gera uma matriz de "NumUns" uns em posições aleatórias da fila pero não se interessa em manter constante a quantidade de uns nas colunas.*

3 *Igual que el modelo 0 solo que la quantidade de unos por fila vai variando entre "NumUns" e "NumUns-1".*

EXEMPLO B.3.1 *O seguinte exemplo gera uma matriz de $K = 256$ linhas e $M = 128$ colunas, cada linha tem $d_v^* = 9$ uns repartidos aleatoriamente com uma distribuição uniforme, se realizarão 77 tentativas em cada linha antes de desistir de obter $d_c^* = 18 = d_v^* K/M$ uns por coluna. A matriz gerada será salvada no arquivo MatrizEmAlist.dat.*

```
parity-matrix-alist-format -k 256 -m 128 --ones 9 --intentos 77
--out MatrizEmAlist.dat --modelo 0
```

EXEMPLO B.3.2 *O seguinte exemplo gera uma matriz de $K = 16$ linhas e $M = 16$ colunas. Cada linha tem $d_v^* = 5$ uns repartidos aleatoriamente com uma distribuição uniforme, se realizarão 77 tentativas em cada linha antes de desistir de obter $d_c^* = 5 = d_v^* K/M$ uns por coluna. A matriz gerada será salvada no arquivo K16-NK16-5-5-S0-I77-M0.dat.*

```
parity-matrix-alist-format -k 16 -m 16 --ones 5 --intentos 77
```

$$\begin{pmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0
\end{pmatrix} \tag{B.3}$$

ARQUIVO B.3.1 (K16-NK16-5-5-S0-I77-M0.DAT : MATRIZ B.3 EM FORMATO ALIST) .

```
16 16
5 6
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 6 5 5 5 5 5 5 5 4 5 5 5 5
4 7 13 14 15
5 6 8 9 13
6 9 11 15 16
1 3 10 11 12
```

```
2 3 4 7 13
4 9 10 14 16
5 8 9 11 16
5 7 9 13 15
2 5 6 13 15
2 4 11 15 16
1 2 4 6 8
5 7 11 14 16
1 3 6 8 12
3 7 8 10 14
1 2 3 10 12
1 4 10 12 14
4 11 13 15 16 0
5 9 10 11 15 0
4 5 13 14 15 0
1 5 6 10 11 16
2 7 8 9 12 0
2 3 9 11 13 0
1 5 8 12 14 0
2 7 11 13 14 0
2 3 6 7 8 0
4 6 14 15 16 0
3 4 7 10 12 0
4 13 15 16 0 0
1 2 5 8 9 0
1 6 12 14 16 0
1 3 8 9 10 0
3 6 7 10 12 0
```

B.4 bf-tests-awgn-egldpc

O *bf-tests-awgn-egldpc* trabalha igual que o programa *bf-tests-awgn-ldgm* visto na seção B.2, com a diferença que usa para a codificação e decodificação, nos algoritmos bit-flipping, matrizes $EG-LDPC$ de tipo I. Os parâmetros de entrada do programa são iguais que no programa *bf-tests-awgn-ldgm*, com a diferença que a opção "$--fileArquivo$" indica o endereço e nome do arquivo que contem a matriz $EG-LPDC$ de tipo I gerado por o programa *generator-matrix-egldpc*. O programa *generator-matrix-egldpc* é descrito na seção B.5.

PROGRAMA B.4.1 (USO DO PROGRAMA BF-TESTS-AWGN-EGLDPC)

bf−tests−awgn−egldpc [−v][−−file Arquivo][−−log ArquivoLog]

[−−out ArquivoOut][−−it−por porcentagem]
[−−bf][−−bfs][−−shbf][−−phbf][−−sssbf]
[−−pssbf][−−wbf][−−mwbf][−−alpha fator]
[−−ber−min BERminimo][−−pack−max Paquetes]

EXEMPLO B.4.1 *O seguinte comando simula os algoritmos bit-flipping (–bf), serial hard bit-flipping (–shbf) e modified weighted bit-flipping (–mwbf), a partir da matriz geradora G e de paridade H achadas no arquivo MatrizEmAlist.dat. Estas matrizes são $EG-LDPC$ de tipo I. O algoritmo weighted bit-flipping usa um $\alpha = 0.5$. Todos os algoritmos bit-flipping usa como máximo $IT = 0.1 * N$ iterações, os algoritmos são desativados sim se obtém na sua saída um BER menor que $1.0e-6$ e como máximo para cada P_b se envia 600000 pacotes (vetores codificados).*

```
bf-tests-awgn-egldpc --file MatrizEGLDPC1.dat --it-por 0.1
--bf --shbf --mwbf --alpha 0.5 --ber-min 1.0e-6 --pack-max 600000
```

Se pode seguir em todo momento a simulação usando $OCTAVE$ aplicando o comando:

```
simula1
```

B.5 generator-matrix-egldpc

O programa *generator-matrix-egldpc* gera um código de bloco do tipo $EG-LDPC$ tipo I com uma matriz de verificação de paridade H de N colunas e J linhas, e uma matriz geradora G de K linhas e N colunas. Polinômios representarão às linhas. Estas matrizes são criadas a partir de una $EG(m, 2^s)$, que é uma geometria euclidiana de dimensão "m", e "2^s" pontos por dimensão. É necessário entregar como dados o valor "m" ou "s", e o polinômio primitivo gerador da $EG(m, 2^s)$. O polinômio primitivo tem que ter um grau $n = m * s$, para gerar uma matriz de verificação de paridade com N colunas, para N igual a $N = 2^n - 1$.

PROGRAMA B.5.1 (USO DO PROGRAMA GENERATOR-MATRIX-EGLDPC) .

generator−matrix−egldpc [−v][−m NumM][−s NumS][−−pos−one Pos]
[−−out Nome][−−gf2n ArchivoGF2n]
[−−help]

-v *Retorna o nome e versão atual do programa.*

-m NumM *Onde "NumM" é o valor "m" da $EG(m, 2^s)$. Se só indicas "m", então "s" calcula-se como n/m onde "n" é o grau do polinômio primitivo.*

-s NumS *Onde "NumS" é o valor "s" da $EG(m, 2^s)$. Se só indicas "s", então "m" calcula-se como n/s onde "n" é o grau do polinômio primitivo.*

–pos-one Pos *Onde "Pos" é a posição de um "1"(X^{Pos}) do polinômio primitivo. $m * s$ tem que coincidir com "n" o grau do polinômio.*

–out Nome *Onde "Nome" é o nome do arquivo que contem os dados da matriz EG-LDPC. Por defeito usa-se os dados de configuração para gerar o nome.*

–gf2n ArchivoGF2n *Onde "ArchivoGF2n" é o nome do arquivo onde se escreve a $GF * (2^{ms})$ na primeira coluna e na segunda a $EG * (ms, 2)$ (equivalente $EG * (m, 2^s)$), para o polinômio gerador indicado com $--pos-one$. O * indica que o elemento zero no está incluído.*

–help *Mostra esta ajuda.*

EXEMPLO B.5.1 *O seguinte exemplo gera uma matriz $EG - LDPC$ de tipo I, a partir de uma $EG(m, 2^s)$, com $m = 2$. Em cada linha tem-se $d_v = 2^s$ uns. O polinômio primitivo usado é $p(X) = X^4 + X + 1$. O grau do polinômio $p(X)$ é $n = 4$. A matriz gerada será salvada no arquivo mateg.dat.*

```
generator-matrix-egldpc -m 2 --out mateg.dat --pos-one 0
--pos-one 1 --pos-one 4
```

Obtendo como resultado o seguinte arquivo:

ARQUIVO B.5.1 (MATEG.DAT : MATRIZES $EG - LDPC$ DE TIPO I) .

```
N= 15
K= 7
Rc= 0.4667
J= 15
LINHAS-PADRAO: 1
LINHAS: 4
0 1 3 7
POLI-GERA: 5
0 4 6 7 8
```

Onde "LINHAS-PADRAO: 1" indica que só uma linha padrão gera toda a matriz de verificação de paridade H. "LINHAS: 4" indica que a matriz H usa 4 pontos por linha. $\{0, 1, 3, 7\}$ *é a representação da linha padrão* $l_0(X) = 1 + X + X^4 + X^7$ *usada para gerar* H.

$$H = \begin{pmatrix}
1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix} \tag{B.4}$$

O texto "POLI-GERA: 5" indica que o polinômio gerador G *tem 5 pontos por linha, e* $\{0, 4, 6, 7, 8\}$ *é a representação da linha geradora* $l_g(X) = 1 + X^4 + X^6 + X^7 + X^8$.

$$G = \begin{pmatrix}
1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1
\end{pmatrix} \tag{B.5}$$