UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

ESCOLA POLITÉCNICA

DEPARTAMENTO DE ELETRÔNICA E DE COMPUTAÇÃO

Estudo de códigos LDPC (Low-Density Parity Check): Matrizes Regulares e Irregulares

Autor: _____
Fernando Jorge Figueiredo Nazareth

Orientador: _____
Prof. Gelson Vieira Mendonça, Ph.D.

Examinador: _____
Prof. Mauros Campello Queiroz, M.Sc.

Examinador: _____
Prof. José Gabriel Rodriguez Carneiro Gomes, Ph.D.

DEL

Novembro de 2010

Sumário

Capítulo 1 - Introdução

A Teoria da informação trata dos aspectos quantitativos de armazenamento e transmissão das mensagens [1], [2]. Tem como um de seus objetivos principais garantir a integridade dos dados enviados através de algum tipo de canal. Na manipulação das mensagens, dois obstáculos são encontrados:

a) falta de capacidade no armazenamento ou transmissão das mensagens enviadas;

b) ruído na transmissão, ou seja, introdução aleatória de erros nas mensagens enviadas.

Ao contrário das teorias matemáticas, que surgiram nas universidades e geralmente após um longo período de tempo, migraram para as aplicações práticas em tecnologia e indústrias, a teoria de códigos corretores de erros surgiu nos laboratórios de empresas de telefonia e posteriormente se transformou em uma teoria matemática completa com aplicações em várias áreas como, por exemplo, geometria algébrica.

Um código corretor de erros visa recuperar informações que no processo de emissão tenham sofrido algum tipo de ruído. Pode-se afirmar que hoje praticamente todo sistema de envio de informações possui algum tipo de código corretor de erros. Como exemplos típicos, a telefonia digital, a transmissão de dados via satélite, a comunicação interna em computadores, armazenamento ótico de dados e armazenamento de dados em fitas ou disquetes magnéticos.

Um sistema de envio de mensagem pode ser esquematizado da forma especificada na figura 1.1:

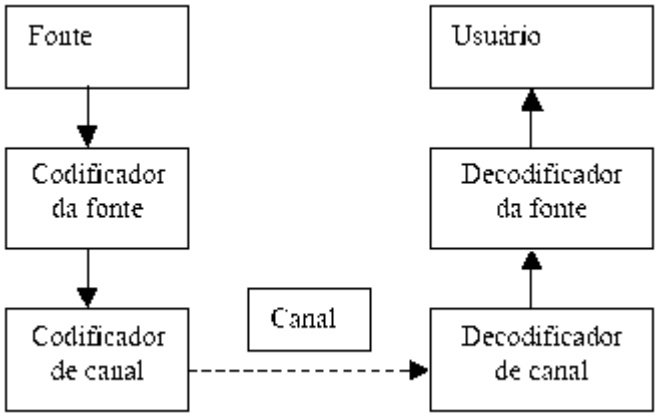


Figura 1.1: Esquema de transmissão

O canal pode ser, por exemplo, circuito integrado digital, disco de armazenamento, cabo, canal de microondas, canal de radiofreqüência, etc.

Com o aumento da confiabilidade nas comunicações digitais e a emergência do computador digital como ferramenta essencial na sociedade tecnológica, os códigos corretores de erros vêm conquistando uma posição proeminente. Para ilustrar a praticidade e importância do uso de códigos corretor de erros, tem:

a) uso do bit de paridade como um mecanismo detector de erro - é um dos esquemas mais simples e conhecidos na comunicação computacional;

b) armazenamento em discos – está sendo muito utilizado devido ao aumento da densidade. Quanto maior a densidade, a probabilidade de ocorrência de erros também aumenta;

c) transmissão de informação pelas naves espaciais:

- em 1972 a espaçonave Mariner transmitiu figuras de Marte para a Terra com 64 tonalidades de cinza. Atividade solar e outras condições atmosféricas podem introduzir erros em sinais fracos vindos do espaço. O código utilizado foi o de Reed-Muller [2],[17];
- em 1979 a espaçonave Voyager começou a enviar imagens com 4096 tonalidades de cores. O código utilizado foi o de Golay [2],[18];

d) áudio digital – o aumento da popularidade do áudio digital deve-se ao desenvolvimento dos códigos corretores de erros que facilita o processo de digitalização.

Ao inicializar a leitura do CD, o sistema corrige os erros produzidos por marcas de dedos, arranhões e outras imperfeições, para logo em seguida transformar em sinais sonoros. O código utilizado é o de Reed-Solomon. Este, criado em 1960 por Irving Reed e Gustave Solomon [3], é um código não binário cíclico - cada deslocamento cíclico das palavras-código é também uma palavra-código - e pode ser construído de forma a corrigir erros múltiplos, devido a sua boa propriedade de distância mínima e à existência de algoritmos eficientes de decodificação (os códigos são determinados por estruturas algébricas, como por exemplo, polinômios). No entanto, apesar da possibilidade de corrigir um elevado número de erros dentro de uma mesma palavra, não é recomendado para transmissão de uma grande quantidade de dados, limitado a palavra-código de tamanho $n \leq q + 1$ - sendo b o número de bits de uma palavra-código e $q = 2^b$ palavras-código possíveis. Para códigos mais longos, as propriedades de distância mínima e, consequentemente, a possibilidade de correção de múltiplos erros diminui.

Motivado pela importância dos códigos corretores de erros, conforme mencionado acima, este trabalho visa estudar o código LDPC (low-density parity-check) [4], pois se trata de um código amplamente utilizado hoje em dia devido a sua performance próxima a capacidade do canal para diversos tipos de canais e um algoritmo de decodificação de baixa complexidade. Atualmente, o sistema DVB-S2 (Digital Video Broadcasting - Satellite versão 2) para transmissão de vídeo por satélite já utiliza este código.

O objetivo do trabalho será realizar comparações entre algoritmos variados de codificação LDPC e observar as diferentes performances em um canal gaussiano. Para isso, foram escolhidos algoritmos regulares e irregulares (as diferenças serão explicadas no capítulo 3 e utilizado o decodificador SPA (Soma de Produto), que foi introduzido por Gallager e até hoje estudado e propostas diversas variações).

No capítulo 2, será explicada a teoria de códigos corretores de erros, abordando conceitos básicos que são necessários para o entendimento do estudo do LDPC. Sistema de comunicação, códigos de blocos lineares, distância de Hamming e decodificação síndroma serão explicitados.

No capítulo 3, os conceitos de LDPC serão introduzidos, fazendo um apanhado em alguns tipos de códigos regulares e irregulares, apresentando algumas propriedades deste código e sua representação com grafos de Tanner [5].

No capítulo 4, serão mostradas formas de construção de alguns tipos de códigos LDPC, a partir de sua matriz de paridade H. Estas diferentes matrizes serão o foco da análise comparativa que será realizada neste trabalho.

O capítulo 5 trará como são feitas a codificação e decodificação dos códigos LDPC, destacando o fato de, em alguns casos, haver uma dificuldade de se obter a matriz geradora do código, necessitando realizar uma solução aproximada. Além disso, dois métodos de decodificação serão detalhados.

No capítulo 6, os resultados das simulações serão mostrados, realizando observações nas comparações de BER dentre os variados algoritmos de codificação.

Por último, o capítulo 7 encerrará o trabalho apresentando as últimas conclusões baseadas na teoria e observações mostradas ao longo deste texto.

Capítulo 2 - Códigos Corretores de Erros

Claude E. Shannon, em 1948, publicou o artigo sobre a teoria matemática da comunicação [1] que se tornou clássico e até hoje é referência bibliográfica de qualquer texto que aborde esse assunto. Segundo Shannon, haveria um limite na quantidade de informação que poderia trafegar por um canal com ruído. Em se tratando de uma comunicação digital em um canal AWGN (Aditive White Gaussian Noise), foi demonstrado que a maior taxa de transmissão que pode trafegar nesse canal é dada por:

$$C = B \times \log_2\left(1 + \frac{P}{N}\right) \tag{2.1}$$

onde C é a capacidade do canal em bits por segundo (bps), P é a potência em Watts do sinal enviado através desse canal e N é a potência em Watts de ruído branco na largura de faixa disponível B em Hertz.

Shannon [1] demonstrou que, adicionando uma redundância controlada à informação, poder-se-ia reduzir a quantidade de erros na recepção induzidos pelo ruído a um patamar tão pequeno quanto se quisesse, desde que a taxa de transmissão estivesse abaixo do limite determinado pela equação (2.1).

Este processo, através do qual a redundância anteriormente citada é adicionada à informação de modo a permitir a detecção e correção de erros, é chamado de codificação de canal. O termo "redundância controlada" está relacionado à restrição das possíveis sequências de bits de informação na recepção. Tendo sido detectado um padrão diferente das possíveis sequências, o decodificador de canal "procura" dentre elas aquela que mais se assemelha à informação detectada. Essa semelhança é obtida através da correta utilização de critérios de decisão, sendo que os mais conhecidos são o critério do máximo a-posteriori - MAP (Maximum a-posteriori) e o de máxima verossimilhança - ML (Maximum Likelihood). Ambos têm como objetivo minimizar o erro de decisão sobre os bits transmitidos.

2.1 - Códigos de blocos e convolucionais

Existem duas grandes famílias de códigos detectores e corretores de erros: os códigos de bloco e os convolucionais [2]. A codificação de bloco atribui a cada bloco de k bits de informação uma palavra código com n bits codificados, n > k. Um código assim formado é descrito na literatura como código de bloco (n,k), sendo que a relação entre o número de bits de um bloco de informação e o número de bits da palavra código correspondente, k/n, é denominada taxa do código. De maneira geral, quanto menor a taxa de um código, maior a sua capacidade de detecção e correção de erros. Como este trabalho tem como objetivo estudar o código LDPC, que são códigos de blocos, aprofundaremos um pouco mais o assunto adiante.

A outra família de códigos se refere aos convolucionais. Nesse tipo de codificação uma seqüência contínua de bits de informação, com tamanho variável, é mapeada em uma seqüência também contínua de bits codificados. Um codificador convolucional é dito com memória, pois um determinado bit codificado depende de um ou mais bits de informação anteriores combinados linearmente.

Existem vários algoritmos de decodificação para códigos de bloco e convolucionais. Destaque maior é dado àqueles caracterizados como algoritmos de decodificação suave (soft decision decoding algorithms). Nesse tipo de decodificação não são utilizados os bits "0s" e "1s" detectados no receptor, como acontece nos algoritmos de decodificação abrupta (hard decision decoding algorithms), e sim os valores reais dos sinais recebidos. Esse processo apresenta consideráveis ganhos em relação ao processo de decodificação abrupta. Estes algoritmos serão melhor explicados no próximo capítulo.

Em vários sistemas de comunicação móvel são utilizadas codificações em cascata. Essa cascata pode conter somente codificadores de bloco, codificadores de bloco e convolucionais ou somente codificadores convolucionais – são os denominados códigos concatenados.

É importante nesse momento ressaltar a diferença entre os processos de codificação de fonte e de codificação de canal. O primeiro tem como objetivo reduzir a quantidade de bits necessários à representação da informação, ou seja, diminuir a redundância existente na informação. O segundo adiciona, de maneira controlada, outro tipo de redundância na

informação, objetivando a detecção e a correção de erros causados pelo canal. Esses dois processos normalmente estão presentes nos sistemas de comunicação digital.

2.1.1 - Conceitos Básicos

Suponha que todos os dados de interesse pudessem ser representados por uma informação binária, isto é, como uma seqüência de zeros e uns. Esta informação binária está para ser transmitida através de um canal que causa erros ocasionais. O propósito de um código é adicionar símbolos extras aos símbolos da informação de modo que os erros podem ser encontrados e corrigidos no receptor. Isto é, uma seqüência de símbolos de dados é representada por uma seqüência maior de símbolos com redundância suficiente para proteger os dados.

Um código binário de tamanho M e comprimento de bloco n é um conjunto de k palavras de comprimento n, chamadas palavras do código. Geralmente, $M = 2_k$ para um inteiro k e o código é denominado de código binário (n,k). Por exemplo, pode ser feito o seguinte código:

$$C = \{00000; 01011; 10110; 11101\}$$

Este é um código muito pobre e muito pequeno com $M = 4$ e $n = 5$, mas ele satisfaz os requisitos da definição. Pode-se usar este código para representar números binários com 2 bits, fazendo a seguinte correspondência arbitrária:

$$00 \rightarrow 00000$$

$$01 \rightarrow 01011$$

$$10 \rightarrow 10110$$

$$11 \rightarrow 11101$$

Suponha-se um robô que se move sobre um tabuleiro quadriculado, de modo que, ao se dar um dos comandos (para frente, para trás, para direita ou para esquerda), o robô se desloca do centro de uma casa para o centro de outra casa adjacente indicada pelo comando. Os quatro comandos acima podem ser codificados como elementos de {0,1} x {0,1}, como se segue:

Para frente $\rightarrow 00$

Para direita $\rightarrow 10$

Para trás $\rightarrow 01$

Para esquerda $\rightarrow 11$

O código acima é então usado como código da fonte. Suponha-se, agora, que esses pares ordenados devam ser transmitidos via rádio e que o sinal no caminho sofra interferências. Imagine-se que a mensagem 00 possa, na chegada ser recebida como 01, o que faria com que o robô, em vez de ir para frente, fosse para trás. O que se faz, então, é recodificar as palavras, de modo a introduzir redundâncias que permitam detectar e corrigir erros. Pode-se, por exemplo, modificar o nosso código da fonte como já foi feito anteriormente:

$00 \rightarrow 00000$

$01 \rightarrow 01011$

$10 \rightarrow 10110$

$11 \rightarrow 11101$

Nessa recodificação, as duas primeiras posições reproduzem o código da fonte, enquanto que as três posições restantes são redundâncias introduzidas. O novo código introduzido na recodificação é chamado de código do canal.

Suponha-se agora que se tenha introduzido um erro ao transmitir, por exemplo, a palavra 01011, de modo que a mensagem recebida seja 11011. Comparando essa mensagem com as palavras do código, é observado que não lhe pertence e, portanto, são detectados erros. A palavra do código mais próxima da referida mensagem (a que tem menor números de componentes diferentes) é 01011, que é precisamente a palavra transmitida.

Quando é recodificado o código fonte, de modo a introduzir redundâncias que permitam detectar e corrigir erros, esta recodificação não precisa ter obrigatoriamente o

código fonte inserido. Por exemplo, no código do robô poderia ter feito a seguinte recodificação:

$$C = \{10101; 10010; 01110; 11111\}$$

utilizando-se a seguinte correspondência:

$$00 \rightarrow 10101$$
$$01 \rightarrow 10010$$
$$10 \rightarrow 01110$$
$$11 \rightarrow 11111$$

Os dois códigos criados para o exemplo do robô não são códigos bons, pois eles não são capazes de corrigir muitos tipos de erros. Por exemplo:

Note que no primeiro código a escolha da palavra código 01011 para estimar a mensagem recebida 11011 é feita de maneira bem natural. De fato, em relação às palavras do código observa-se que:

11011

00000 → 4 diferenças
01011 → 1 diferença
10110 → 3 diferenças
11101 → 2 diferenças

Suponha-se que tivesse recebido a mensagem 01110. Nesse caso, em relação às palavras do código teria:

01110

00000 → 3 diferenças
01011 → 2 diferenças
10110 → 2 diferenças

11101 →3 diferenças

Nesse caso, não é possível estimar qual foi a palavra código transmitida.

O ponto de partida para a construção de um código corretor de erros é definir o alfabeto A com um número finito de *q* símbolos. No caso dos códigos binários $A = \{0, 1\}$.

Um código corretor de erros é um subconjunto próprio qualquer de $A_n = A \times A \times A \times \ldots \times A$, para algum número natural *n*. O número de elementos de um conjunto *A* será denotado por $|A|$. O código do robô é um subconjunto próprio de A_5, com $A = \{0, 1\}$, onde $A_5 = \{(00000);(00001);(00010);(00100);.....;(11111)\}$ e $|A_5| = 2^5 = 32$

Para que se possam identificar as palavras mais próximas de uma dada palavra recebida com erro e estimar qual foi a palavra do código transmitida, será apresentado um modo de medir a "distância"entre palavras em A_n.

2.1.2 - Distância de Hamming

Suponha-se que $\underline{c}_i$ e $\underline{c}_j$ sejam duas palavras-código quaisquer de um código binário (*n,k*). Uma medida da diferença entre as duas palavras-código é o número de bits em posições correspondentes que diferem entre si. Esta medida é denominada de **distância de Hamming** e é denotada por d_{ij}. Por exemplo, sejam $\underline{c}_i = [0\ 1\ 0\ 1\ 1]$ e $\underline{c}_j = [1\ 0\ 0\ 0\ 1]$. Então $d_{ij} = 3$.

Observe que d_{ij} sempre satisfaz a condição $0 < d_{ij} \leq n$, $i \neq j$, para duas palavras-código $\underline{c}_i$ e $\underline{c}_j$, ambas de *n* bits (por definição, em um código C(*n,k*), $\underline{c}_i \neq \underline{c}_j$ $\forall i$ e $\forall j$ com $i \neq j$).

O menor valor no conjunto $\{d_{ij}\}$, $i,j = 0,1,...,M-1$, $i \neq j$, $M = 2^k$ é denominado **distância mínima** (d_{min}). Por exemplo, para o código:

C = {10101; 10010; 01110; 11111}

as distâncias estão colocadas na tabela 2.1:

Tabela 2.1 - distância entre as palavras-código.		
c_i	c_j	d_{ij}
10101	10010	3
10101	01110	4
10101	11111	2
10010	01110	3
10010	11111	3
01110	11111	2

Portanto, $d_{min} = 2$.

A Distância de Hamming d_{ij} é uma medida do grau de separação entre duas palavras-código. Portanto, d_{min} está associado à capacidade do código em identificar palavras-código demoduladas no receptor quando estas são recebidas com erro como conseqüência do ruído e interferência presentes no canal. Em outras palavras, quanto maior o d_{min} maior a capacidade de um código detectar e corrigir erros.

Com isso, seja C(n,k) um código corretor binário, d o número máximo de erros que C(n,k) é capaz de detectar, t o número máximo de erros que C(n,k) é capaz de corrigir e d_{min} a distância mínima do código. Então:

$$d = \frac{d_{min}}{2} \tag{2.1}$$

$$t = \frac{d_{min} - 1}{2} \tag{2.2}$$

$$d_{min} \leq n - k + 1 \tag{2.3}$$

No exemplo da tabela 2.1, $d_{min} = 2$ para o código C. Portanto, pelas equações (2.1) e (2.2) temos que $d = 1$ e $t = 0$ (pegando apenas a parte inteira), ou seja, o código C detecta no máximo 1 erro e não tem capacidade de corrigi-lo. É um código simples que serve apenas como exemplo.

2.2 - Códigos de Blocos Lineares

Como o LDPC é o objetivo de estudo deste trabalho e por se tratar de um código de bloco linear, neste tópico será aprofundado este assunto.

Seja $\{m_0, m_1, ..., m_{k-1}\}$ um bloco arbitrário de k bits gerados pela fonte. Tipicamente, o codificador de bloco usa estes k bits para gerar uma palavra de código com n > k bits, acrescentando n-k bits de controle. A palavra-código assim construída tem a estrutura como:

$$[b_0 \ b_1 \ \cdots \ b_{n-k-1} \mid m_0 \ m_1 \ \cdots \ m_{k-1}]$$

Este é um código (n, k) e tem uma taxa de codificação definida por R = k/n.

No caso de um código de bloco linear, os bits de paridade $b_0, b_1, \cdots, b_{n-k-1}$ dependem linearmente (numa aritmética binária) dos bits da mensagem $m_0, m_1, \cdots, m_{k-1}$. Ou seja, definindo os vetores linha

$$b = [b_0 \ b_1 \ \cdots \ b_{n-k-1}]$$

e

$$m = [m_0 \ m_1 \ \cdots \ m_{k-1}]$$

podemos escrever

$$b = mP \tag{2.4}$$

onde P é uma matriz binária (k x n-k) que determina o código. Portanto, sendo

$$x = [b \mid m] \tag{2.5}$$

e substituindo (2.4) em (2.5), resulta

$$x = m[P \mid I_k] \tag{2.6}$$

onde I_k é a matriz identidade de dimensão (k x k). Definindo a matriz geradora do código:

$$G = [P \mid I_k] \tag{2.7}$$

de dimensão (k x n), usando (2.6) se tem a equação:

$$x = mG \tag{2.8}$$

É fácil verificar que o código formado pelas palavras x, geradas pela matriz G a partir das 2^k mensagens m, é um código linear. Com isso, sendo $x_i = m_iG$ e $x_j = m_jG$ palavras-código, então $x_i \oplus x_j = m_iG \oplus m_jG = (m_i \oplus m_j)$ G; como $m_i \oplus m_j$ é necessariamente uma mensagem, $x_i \oplus x_j$ é uma palavra do código.

Definindo a matriz de verificação de paridade

$$H = [I_{n-k} \mid P^T] \tag{2.9}$$

de dimensão (*n-k* x *n*), e usando (2.7) verifica-se que

$$HG^T = [I_{n-k} \mid P^T] \begin{bmatrix} P^T \\ \hline I_k \end{bmatrix} = P^T + P^T = 0_{n-k \times k} \tag{2.10}$$

Assim sendo, de (2.8) vem $x^T = G^T m^T$, o que, utilizando a equação (2.10), conduz a

$$Hx^T = 0_{n-k \times 1} \quad \text{ou} \quad xH^T = 0_{1 \times n-k} \tag{2.11}$$

Esta é uma condição necessária e suficiente para que *x* seja uma palavra do código (*n*, *k*) gerado pela matriz *G*. No entanto, a verificação de (2.11) à saída do canal ruidoso não significa necessariamente que não tenham ocorrido erros de transmissão. Com isso, se for transmitida a palavra-código *x*, então, a palavra *y* recebida é, em geral,

$$y = x + e \tag{2.12}$$

onde *e* é o vetor de erro $e = [e_0 \cdots e_i \cdots e_{n-1}]$, e_i é igual a 1 se houver erro no bit *i* e 0 caso não haja erro. Usando-se (2.12) em (2.11):

$yH^T = xH^T + eH^T = eH^T = 0$ se *e* for palavra-código

$yH^T = xH^T + eH^T = eH^T \neq 0$ caso contrário

o que significa que sendo o vetor de erro uma palavra-código, a palavra *y*, recebida com erros de transmissão, cumpre o teste de verificação de paridade (2.11), e os erros de transmissão não são detectados.

2.3 - Gráficos de Tanner

Os Gráficos de Tanner[5] são uma das formas mais simples de representar um código binário linear.

Sabemos que um código binário linear (n,k) pode ser definido por um sistema de (n-k) equações lineares homogêneas a n incógnitas (os bits das palavras-código). Podemos construir a partir deste sistema de equações, um gráfico bipartido formado por dois tipos de nós:

- (n-k) nós designados por nós de teste (CN's - check nodes), um por cada uma das equações lineares homogêneas do sistema.

- nós designados por nós das variáveis (BN's - bit nodes), um por cada uma das variáveis do sistema de equações.

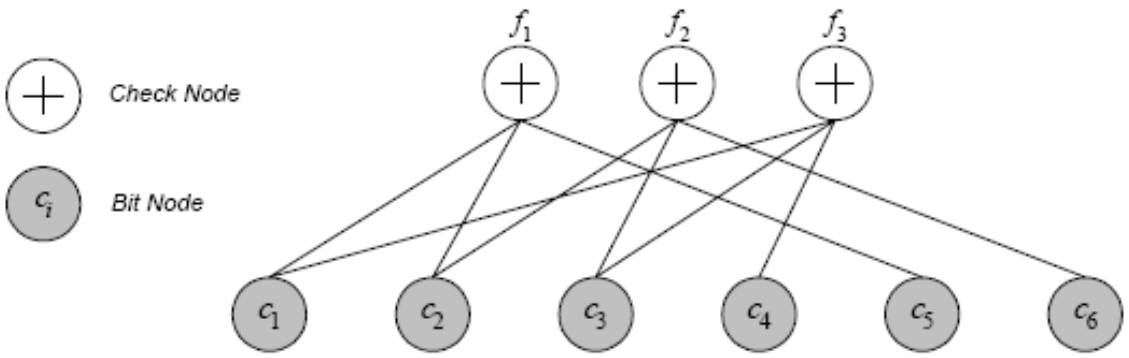
Cada CN é ligado a todos os BN's que intervêm na equação à qual o CN está associado. Por sua vez, cada BN associado a um dado bit da palavra-código, é ligado a todos os CN's correspondentes às equações de paridade na qual o bit intervém. Devido a este fato, apenas existem ligações entre BN's e CN's e nunca entre nós do mesmo tipo (gráfico bipartido). A um gráfico deste tipo dá-se o nome de **Gráfico de Tanner**.

Dada a matriz de teste de paridade H de um código binário linear (n,k), facilmente se pode obter o seu GT e vice-versa. Assim, basta atender a que existem (n-k) CN's associados a cada linha da matriz H, n BN's associados a cada coluna de H e que cada CN j é ligado ao BN i sempre que na matriz o elemento $h_{ij}=1$.

Exemplo - Considere-se o código binário linear (6,3), definido pela matriz de teste de paridade:

$$H = \begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \end{array} \begin{array}{c} \begin{array}{cccccc} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

O GT associado a este código é:



A importância dos GT como forma de representação de códigos binários lineares reside no fato de muitos dos algoritmos de decodificação destes códigos terem por base esta representação gráfica. Mais adiante, no capítulo de LDPC, será mostrada sua aplicação para este tipo de código estudado neste trabalho.

2.4 - Decodificação pela Síndrome

Para proceder à decodificação, isto é, a detecção e/ou correção de erros de transmissão, o decodificador começa por calcular a síndrome da palavra recebida *y*, isto é, o vetor binário

$$s = yH^T \tag{2.13}$$

de dimensão *n-k* . Levando-se em conta (2.11) e (2.12), verificamos que a síndrome só depende do padrão de erros, ou seja,

$$s = eH^T \tag{2.14}$$

Por outro lado, todos os padrões de erro que diferem entre si de uma palavra-código têm a mesma síndrome. Com isso, dado um vetor e que verifique (2.14), então, todos os vetores de erro

$$e_i = e \oplus x_i, \; i = 0,1,...,2^k\text{-}1 \tag{2.15}$$

onde x_i , i = 0,1,...,2^k-1 são todas as palavras-código, verificam também (2.14). Assim, define-se o *coset* do padrão de erros e como sendo o conjunto dos vetores de erro definido em (2.15) que têm a mesma síndrome (2.14). Uma vez que um código de bloco linear (*n*,*k*) tem 2^k palavras admissíveis, num total de 2^n palavras binárias de comprimento *n*, conclui-se que existem 2^{n-k} *cosets*, isto é, 2^{n-k} síndromes distintas.

A síndrome contém alguma informação sobre o correspondente padrão de erros, embora geralmente insuficiente para o identificar sem ambigüidade. Se assim fosse, qualquer padrão de erros poderia ser corrigido. De qualquer modo, o conhecimento da síndrome *s* reduz o espaço de busca de uma 2^n dimensão para 2^k. Uma vez calculado *s*, o decodificador deve escolher o elemento do respectivo *coset* que otimize um determinado critério. Por exemplo, a respectiva probabilidade de ocorrência. Para valores relativamente baixos da probabilidade de ocorrência de erros de transmissão, o padrão de erros mais provável corresponde àquele que tem menos 1's, isto é, aquele cujo peso

$$w(e) = \sum_{i=0}^{n-1} e_i \tag{2.16}$$

é mínimo.

Portanto, o algoritmo de decodificação seria da seguinte maneira.

Dada a palavra recebida y:

1. Calcular a síndrome $s = yH^T$
    a) $s = 0$, então, y deve ser uma palavra-código $y_0 = y$
    b) $s \neq 0$, então, executar próximo passo.

2. Calcular o *coset* de y $\{e_i = y \oplus x_i$, i = 0,1,...,2^k-1$\}$, escolher o padrão e_0 de menor peso e executar o próximo passo.
3. Construir a palavra corrigida $y_0 = y \oplus e_0$.

Observação: sendo $e_0 = y \oplus x_0$, com $x_0 H^T = 0$, tem-se

$$y_0 = y \oplus e_0 = y \oplus y \oplus x_0 = x_0$$

isto é, a saída do decodificador é a palavra-código que difere da palavra recebida num número mínimo de posições (correspondentes às posições dos 1's em e_0). Ainda por outras palavras, é a palavra-código para a qual $y \oplus x_0$ tem peso mínimo.

Capítulo 3 - Código Low-density parity-check (LDPC)

Os códigos LDPC foram introduzidos por R.G.Gallager [4] no início dos anos 60 e são capazes de atingir um desempenho próximo da capacidade em diversos modelos de canais assim como os códigos Turbo [6]. Nesta época, os computadores não eram capazes de simular o desempenho de códigos com comprimentos significativos e com baixas taxas de erro: na geração da matriz **H** que garantisse uma boa distância mínima do código, na codificação (normalmente não sistemático) e ainda na sua decodificação, tendo por base o algoritmo Soma de Produtos (SPA) proposto pelo próprio Gallager. Isso fez com que os códigos LDPC fossem deixados de lado pelos pesquisadores por muito tempo. Somente mais tarde, em 1981, R.M.Tanner [5] generalizou o trabalho de Gallager e introduziu a representação gráfica de códigos LDPC através de grafos bipartite conforme foi mostrado no capítulo anterior. Ainda não foi nesta época que os códigos LDPC passaram a ser utilizados. Mas, D.J.C. Mackay [7] em meados da década de 90; após o advento dos códigos Turbo e da decodificação iterativa, os redescobriu. Mackay mostrou que os códigos LDPC longos, quando decodificados com o algoritmo Soma-Produto (SPA), são capazes de atingir um desempenho muito próximo ao limite de Shannon [1] do Canal AWGN. Com isso, estes códigos passaram a ser intensamente estudados e utilizados para controle de erros em um grande número de sistemas de comunicação e armazenamento de dados.

Além disso, começaram a ser estudados os códigos LDPC irregulares, ao contrário dos códigos regulares introduzidos por Gallager [4]. Estes códigos se mostraram superiores aos regulares em alguns aspectos e a diferença entre estes códigos, que é o objetivo deste projeto, será explorada mais adiante.

3.1 - Códigos regulares e irregulares

Um código LDPC regular é definido como o espaço nulo de uma matriz de verificação de paridade **H** de M x N que possui a seguinte propriedade: cada linha e coluna possuem pesos constantes d_c *(linha)* e d_s *(coluna)* e estes pesos são muito pequenos quando comparados com o comprimento N do código e com o número de linhas M em **H**,

respectivamente. Por isso, o nome low-density parity-check, ou seja, uma matriz de verificação de paridade com uma densidade baixa (pesos nas linhas e colunas bem menores que seu tamanho).

Isto significa que em um código LDPC, cada símbolo pertencente a uma palavra-código é envolvido em d_s equações de paridade e cada equação de paridade envolve d_c símbolos de uma palavra-código. Um código com estas características é chamado de código LDPC (d_s,d_c)-regular, o número de elementos não nulos em **H** é

$$M \times d_c = N \times d_s \tag{3.1}$$

Como o número total de elementos na matriz é M x N, a densidade de **H** (porcentagem de 1's na matriz), denotada por r, é dada por

$$r = \frac{M \times d_c}{M \times N} = \frac{d_c}{N} = \frac{N \times d_s}{M \times N} = \frac{d_s}{M} \tag{3.2}$$

Geralmente, por se tratar de uma matriz com poucos valores não-nulos se comparado com o seu tamanho, as técnicas de projeto de códigos LDPC não garantem que **H** irá possuir apenas linhas linearmente independentes. Ou seja, normalmente, para um código LDPC, o posto(**H**) é menor ou igual à M. Portanto, o número de símbolos de verificação de paridade satisfaz $N - K \leq M$, com igualdade apenas quando todas as linhas de **H** forem linearmente independentes. É possível eliminar as linhas linearmente independentes para encontrar uma matriz de verificação de paridade $N - K$ x M de posto completo, entretanto, a matriz resultante provavelmente não possuiria colunas e linhas de peso constante. Normalmente, as possíveis dependências entre as linhas de **H** são ignoradas e assume-se que a taxa do código - a razão entre a quantidade de informação e o tamanho da palavra-código - é igual a:

$$R = \frac{N - M}{N} = 1 - \frac{M}{N} = 1 - \frac{d_s}{d_c} \tag{3.3}$$

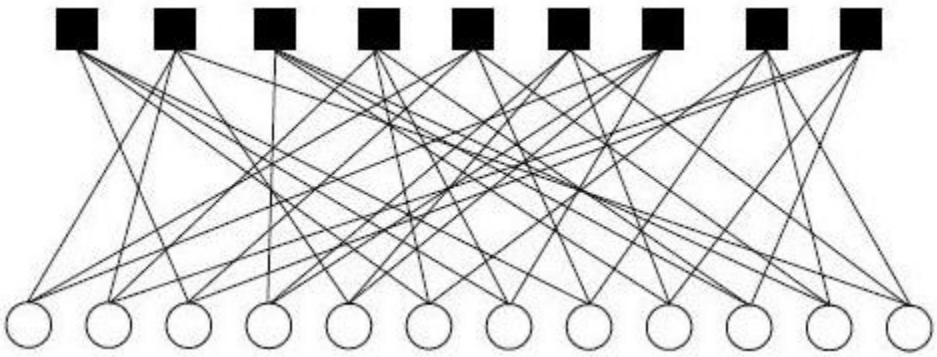
Gallager, em sua dissertação, provou ainda que fazendo $d_s \geq 3$, os códigos LDPC que podem ser obtidos possuem, em sua maioria, uma distância mínima elevada, bastando para tal seguir algumas regras simples de construção como, por exemplo, garantir que quaisquer duas colunas da matriz **H** possuam quanto muito um só 1 em comum. Devido à

baixa densidade da matriz **H**, é garantido que o número mínimo de colunas de **H** que são necessárias somar de forma a obter o vetor nulo é elevado e, logo, que o código possua uma distância mínima elevada de acordo com o que foi explicado no capítulo anterior.

O grafo fator, que nada mais é do que a generalização dos grafos de Tanner já vistos acima, de um código LDPC (d_s,d_c)-regular é constituído de N nós de variável (BN's) de grau d_s e M nós de verificação (CN's) de grau d_c. Uma vez que os graus dos nós são determinados, ainda é possível escolher quais as conexões em particular serão realizadas. O par (d_s,d_c), juntamente com o comprimento N do código, especifica um conjunto de códigos LDPC. Para ilustrar a estrutura dos códigos LDPC (d_s,d_c)-regular, a matriz de verificação e o grafo fator de um código LDPC (3,4)-regular de comprimento 12 são mostrados abaixo:

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

(a)



(b)

figura 3.1:(a) matriz de paridade e (b)grafo fator para um código LDPC (3,4)-regular de comprimento 12

Neste caso, temos:

$$M = 9 \quad N = 12 \quad d_c = 4 \quad d_s = 3$$

Este código possui taxa $R = 1 - ¾ = ¼$ e densidade $r = 4/12 \approx 0.33$.

As matrizes de verificação de paridade dos códigos LDPC também podem apresentar linhas e colunas com pesos variáveis. Neste caso, os códigos LDPC são chamados *irregulares*. Os códigos LDPC irregulares também possuem matrizes de verificação de paridade esparsas e são representados por grafos nos quais os BN's e CN's não possuem graus constantes. Um conjunto de códigos LDPC irregulares é definido através de *distribuições de grau*. Uma distribuição de grau $\gamma(x)$ é um polinômio da forma:

$$\gamma(x) = \sum_i \gamma_i x^{i-1} \qquad i \geq 1$$

(3.4)

em que os coeficientes são números reais não-negativos, de modo que $\gamma(1) = 1$. Dado um par $(\lambda(x),\rho(x))$ de distribuições de grau e um número natural N, define-se o conjunto de códigos LDPC $(\lambda(x),\rho(x))$-irregulares de comprimento N, no qual, a partir de (3.4) temos $\lambda(x) = \sum_i \lambda_i x^{i-1}$ como a distribuição de grau dos BN's, e $\rho(x) = \sum_i \rho_i x^{i-1}$ como distribuição de grau dos CN's. Os coeficientes de λ_i (ou ρ_i) denotam a fração de ramos no grafo que estão conectados a BN's (ou CN's) de grau i. A taxa de um código LDPC $(\lambda(x),\rho(x))$-irregular é dada pela seguinte expressão:

$$R = 1 - \frac{\int_0^1 \rho(x)dx}{\int_0^1 \lambda(x)dx}$$

(3.5)

Exemplo de código LDPC-irregular: Seja um conjunto de códigos LDPC $(\lambda(x),\rho(x))$-irregulares de comprimento 12, definido pelas distribuições de grau:

$$\lambda(x) = \frac{1}{2}x + \frac{1}{2}x^2$$

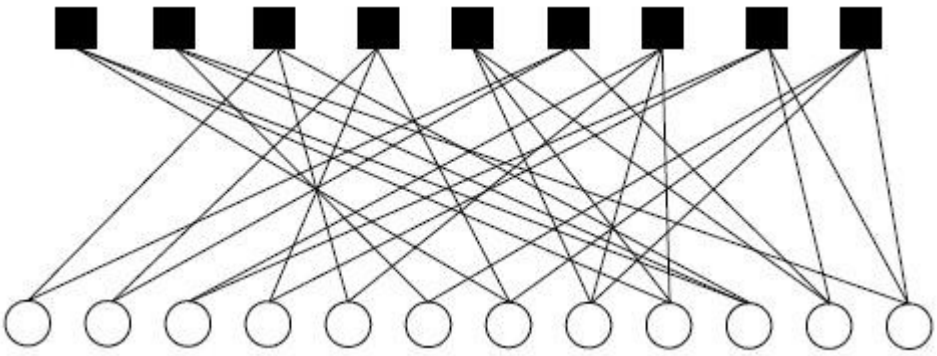
e

$$\rho(x) = \frac{2}{3}x^2 + \frac{1}{3}x^3$$

Um elemento deste conjunto é um código definido a partir de 9 equações de verificação de paridade. De acordo com as distribuições de grau, o grafo fator que representa este código possui 6 BN's de grau 2, 6 BN's de grau 3, 6 CN's de grau 3 e 3 CN's de grau 4. Uma matriz de verificação de paridade e seu respectivo grafo fator para este código são mostrados a seguir.

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(a)



(b)

figura 3.2: (a) matriz de paridade e (b) grafo fator para um código LDPC (λ(x),ρ(x))-irregular de comprimento 12

A densidade da matriz **H** é $r = 30/108 \approx 0.2777$ e a taxa deste código é dada por

$$R = 1 - \frac{\int_0^1 \frac{2}{3}x^2 + \frac{1}{3}x^3 dx}{\int_0^1 \frac{1}{2}x + \frac{1}{2}x^2 dx} \approx 0.2677$$

3.2 - Conceito de Giro

Um dos mais importantes conceitos relativos aos GT (grafos de Tanner) é a definição de *giro* (girth) de comprimento *l* como sendo um percurso fechado formado por *l* caminhos. Por exemplo, tendo por base a figura 3.3 podemos observar um ciclo de comprimento 6 que se encontra assinalado em negrito. O menor comprimento de todos os ciclos existentes num GT é designado por giro.

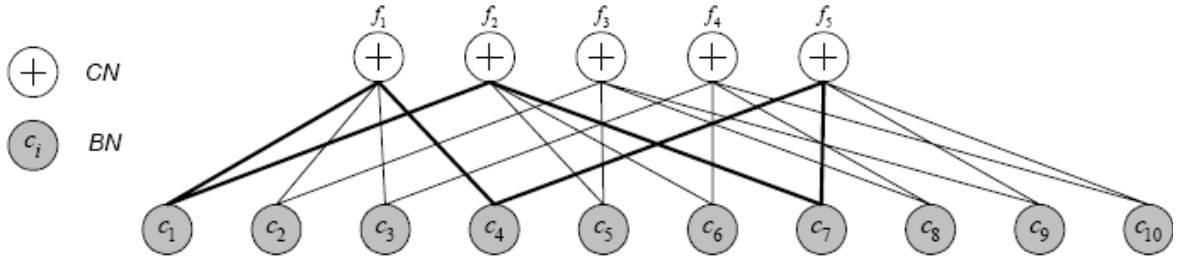


figura 3.3: grafo de Tanner para um código LDPC regular.

Na prática, ao projetar um código LDPC procura-se evitar a existência de ciclos de pequeno comprimento no seu GT de forma a melhorar o desempenho do algoritmo SPA. De fato, prova-se que o algoritmo SPA tem desempenho ótimo quando aplicado a gráficos sem ciclos. Na presença de gráficos com ciclos, a sua eficiência diminui, sendo inferior para códigos com um baixo giro. Por outro lado, prova-se também que GT sem ciclos não suportam bons códigos, pelo que é necessário obedecer a algumas regras na construção dos códigos LDPC por forma a que estes possuam boas propriedades (distância mínima e um giro elevados).

3.2.1 - Identificação de ciclos e determinação do giro de um código LDPC

Qualquer ciclo de um GT tem, necessariamente, um comprimento par e o seu valor mínimo é 4 correspondendo a uma matriz de teste de paridade em que existem duas colunas com dois 1's em comum (ver figura 3.4). Uma das regras de construção da matriz **H**, para evitar a existência de ciclos de dimensão 4, consiste em garantir que quaisquer duas colunas da matriz , possuam quanto muito um só 1 em comum.

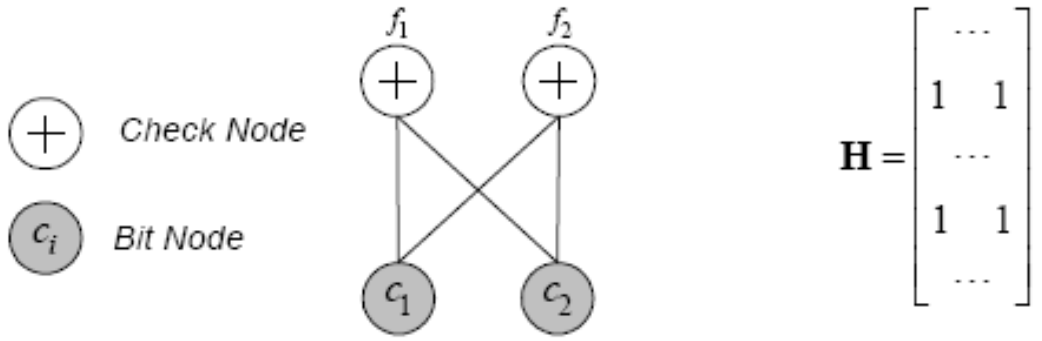


figura 3.4: Exemplo de ciclo de comprimento 4 e sua respectiva matriz de verificação de paridade.

Já os ciclos de comprimento 6 são mais difíceis de identificar na matriz **H**. Na figura 3.5 apresentamos um exemplo.

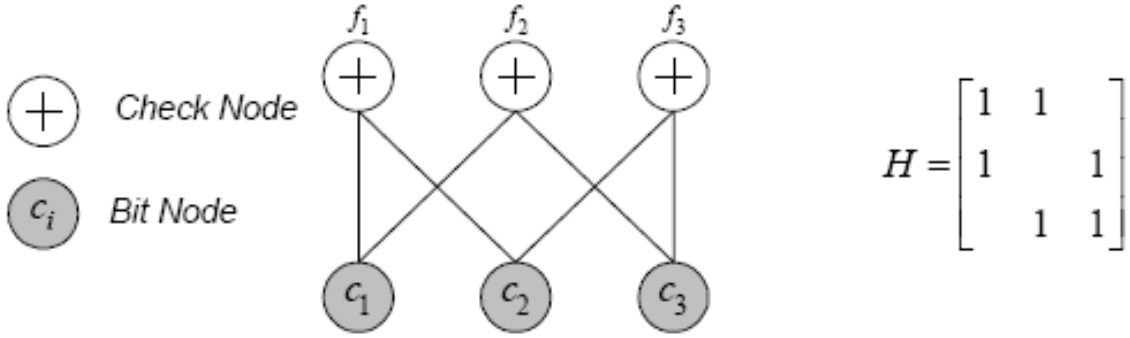


figura 3.5: Exemplo de um ciclo de comprimento 6 num GT e a sua respectiva matriz de verificação de paridade.

Mao e Banihashemi [8] apresentam uma forma simples de determinar o giro associado a cada BN, isto é, o ciclo mais curto que passa por cada BN. O método consiste em construir para cada BN uma árvore a partir do GT ou da matriz **H** do código. Assim, considera-se como raiz da árvore o BN_i cujo giro pretendemos determinar. A árvore é então construída passo a passo. No nível *k* de descendência da árvore são incluídos todos os nós a uma distância *k* do BN_i. O procedimento é repetido até o nível de descendência *n* em que é incluído um nó que se encontra ligado no GT do código a pelo menos dois nós já incluídos

no nível de descendência. Isto identifica a formação do primeiro ciclo, sendo $2n$ o giro do BN_i.

Em resumo, o método consiste em adicionar como descendentes, a cada nó da árvore, todos os nós que a ele se encontram ligados no GT, com exceção do nó pai, terminando o algoritmo quando é encontrado um nó que é descendente de mais do que um nó diferente.

Na figura 3.6, pode-se observar a aplicação do método anteriormente descrito na determinação do giro do BN do código descrito pelo GT da figura 3.3. Como pode ser observado, só no 3º nível de descendência da árvore encontramos um nó descendente simultaneamente de mais do que um nó, pelo que se conclui que o giro do BN é 6. Assinalado em negrito encontra-se identificado um dos ciclos de comprimento 6 que contém o BN .

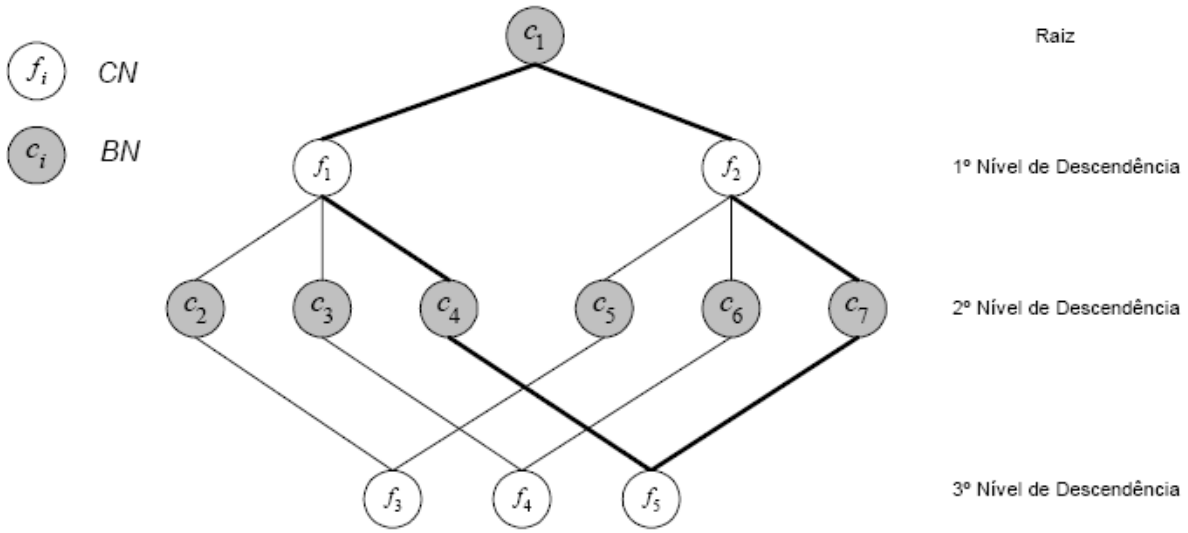


figura 3.6: Determinação do giro C_i do código LDPC descrito na figura 3.3.

McGowan e Williamson [9] apresentam um método algébrico de determinação do giro de cada BN, baseado no conceito de *matriz adjacente*. Dado um código LDPC descrito por uma matriz de verificação de paridade **H**, a sua matriz adjacente é definida por,

$$A = \begin{bmatrix} 0 & H \\ H^T & 0 \end{bmatrix} \tag{3.6}$$

pelo que cada nó do GT do código (BN ou CN) é representado por uma linha e por uma coluna da matriz **A**.

Definindo os nós GT como $v_1, v_2, \ldots, v_p$, com a matriz A = [a_{ij}] de tamanho $p \times p$, então:

$a_{ij} = 1$ se v_i está conectado ao v_j,

$a_{ij} = 0$ se o contrário.

Considerando:

$$A^2 = \begin{bmatrix} HH^T & 0 \\ 0 & H^T H \end{bmatrix}$$

os elementos a_{ij}^2 da matriz A^2 são calculados pela fórmula:

$$a_{ij}^2 = \sum_{k=1}^{p} a_{ik} a_{kj}$$

percebe-se que este também é o caminho de tamanho 2 entre v_i e v_j, pois cada caminho contém seu nó intermediário v_k cujas duas ligações são $a_{ik} = 1$ e $a_{kj} = 1$.

Portanto, considerando um gráfico com giro de tamanho 4, observa-se nestes ciclos que cada nó v_i é conectado ao nó v_j, com dois caminhos de tamanho 2 entre eles, ou seja, os giros de tamanho 4 são observados quando $a_{ij}^2 = 2$.

Com isso, provaram que o elemento (i,j) da matriz A^n é o número de percursos com comprimento n entre os nós i e j do código. Neste sentido, cada elemento não-diagonal da matriz A^n, a_{ij}^n, representa o número de ciclos de dimensão n que contém os nós i e j. O método descrito permite determinar rapidamente o giro de cada BN, bem como, o giro do código.

Capítulo 4 – Construção de códigos LDPC

A forma mais simples de projetar um código LDPC consiste na construção da matriz de teste de paridade que cumpra um conjunto de requisitos pretendidos, como sejam, definir um código regular (caracterizado por um dado peso para cada coluna e linha) ou irregular (podendo ser especificado o grau de cada BN e CN), ou ainda, garantir um dado giro mínimo. Este projeto também pode ser feito a partir do GT que a matriz de teste de paridade irá representar.

Obviamente, para qualquer conjunto de restrições existe um conjunto imenso de códigos que cumprem as referidas especificações. A construção da matriz **H** é feita de forma quase aleatória, mas seguindo algumas regras que maximizam a probabilidade do código obtido possuir um bom desempenho. No entanto, essas regras não nos dão qualquer garantia de que tal aconteça. Poderão inclusive suceder situações em que a matriz **H** obtida não será de característica máxima, ou seja, algumas linhas ou colunas não serão linearmente independentes e, portanto, definirá um código LDPC (n,k') com $k' <$ k.

O primeiro método utilizado para obter matrizes de códigos LDPC regulares foi originalmente proposto por Gallager [4]. Para construir a matriz de verificação de paridade, denominada $\mathbf{H}_{GA}$; é preciso primeiro construir uma submatriz $\mathbf{H}_1$ de dimensões k x (k **.** d_c), na qual as colunas apresentam peso 1 e as linhas apresentam peso d_c. A constante k é determinada a partir do comprimento N e do valor de d_c, de modo que $N = k$ **.** d, conforme mostrado na figura 4.1:

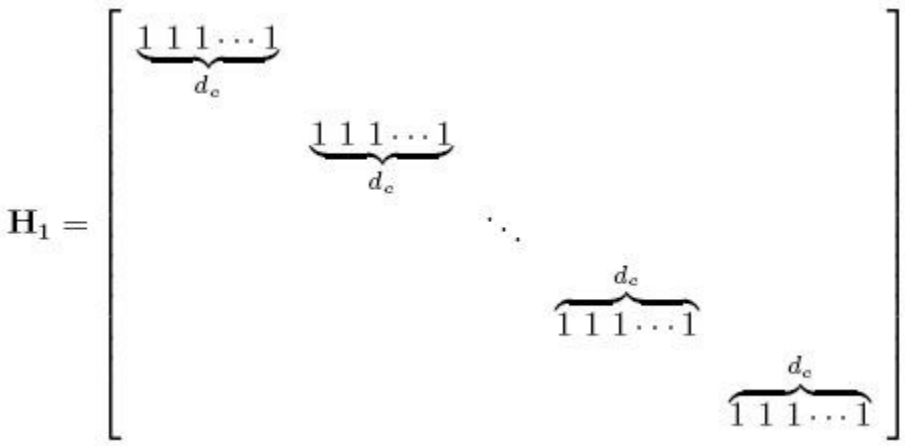


figura 4.1: matriz de paridade proposta por Gallager.

Em seguida, são realizadas $d_s - 1$ permutações das colunas de $\mathbf{H_1}$ para formar outras $d_s - 1$ submatrizes $\mathbf{H_2}$, $\mathbf{H_3}$, $\cdots$, $\mathbf{H_{ds}}$ de dimensões k x k **.** d_c. Com estas d_s submatrizes forma-se a matriz $\mathbf{H_{GA}}$ da seguinte maneira:

$$\mathbf{H}_{GA} = \begin{bmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \\ \vdots \\ \mathbf{H}_{d_s} \end{bmatrix}$$

Os pesos das colunas e das linhas da matriz resultante $\mathbf{H_{GA}}$ são d_s e d_c, respectivamente. A densidade $r = 1/k$. Portanto, a matriz $\mathbf{H_{GA}}$ é esparsa para $k >> 1$.

As $d_s - 1$ permutações devem ser escolhidas de modo que o código gerado pela matriz $\mathbf{H_{GA}}$ possua uma boa distância mínima e seu GT não possua ciclos curtos, especialmente, ciclos de comprimento 4. Gallager [4] não sugeriu nenhum método específico para encontrar estas permutações e não existe nenhum algoritmo conhecido para encontrar permutações que garantam a não existência de ciclos curtos. Normalmente, os códigos LDPC de Gallager são construídos através de buscas computacionais exaustivas por permutações que garantam um código de bom desempenho. No entanto, é possível utilizar permutações aleatórias na construção de códigos LDPC. Tal construção não garante

nenhuma propriedade de distância mínima ou ciclo mínimo, porém, para comprimentos longos é muito provável que o código construído desta forma possua um bom desempenho.

Exemplo de Código de Gallager: A matriz $\mathbf{H}_{\mathbf{GA}}$ mostrada na figura 4.2 foi construída a partir da técnica de Gallager, com os parâmetros $N = 20$, $d_s = 3$ e $d_c = 4$.

$$\mathbf{H}_{GA} = \left[\begin{array}{cccccccccccccccccccc}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\hline
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
\hline
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{array}\right]$$

figura 4.2: matriz de Gallager construída com $N = 20$, $d_s = 3$ e $d_c = 4$.

Makay e Neal [7] apresentam um conjunto de estratégias para gerar códigos LDPC. Estas estratégias são apresentadas de forma numerada, sendo convicção dos autores que as de ordem superior maximizam a probabilidade do código obtido possuir um melhor desempenho. No entanto, eles próprios reconhecem não possuir qualquer prova deste fato.

- Estratégias apresentadas por Mackay e Neal para construção de códigos LDPC:

I. A matriz **H** é gerada partindo de uma matriz de zeros de dimensões $(n - k)$ x n e aleatoriamente colocando d_s bits em cada coluna (o código assim gerado poderá ser irregular);

II. A matriz **H** é gerada criando aleatoriamente colunas de peso de Hamming d_s;

III. A matriz **H** é gerada criando aleatoriamente colunas de peso de Hamming d_s e procurando uniformizar ao máximo o peso de Hamming d_c de cada linha.

IV. A matriz **H** é gerada com colunas de peso de Hamming d_s, linhas de peso de Hamming d_c, e não possuindo quaisquer duas colunas com mais de um 1 em comum;

V. A matriz **H** é gerada de acordo com o procedimento do ponto anterior mas tendo como objetivo a maximização do giro do código (apresentado no capítulo anterior);

VI. A matriz **H** é gerada de acordo com o procedimento referido em IV, procurando obter uma matriz **H** de característica máxima, de preferência na forma $\mathbf{H} = [\mathbf{H_1} \mid \mathbf{H_2}]$ com $\mathbf{H_1}$ ou $\mathbf{H_2}$ inversível.

Mackay e Neal [7] apresentam ainda em um conjunto de técnicas que fazem uso de algumas das estratégias antes referidas. Essas técnicas são vulgarmente conhecidas por 1A, 2A, 1B e 2B, sendo usadas por muitos investigadores na classificação dos códigos LDPC estudados.

A estratégia 1A diz respeito ao ponto III com $d_s = 3$, e em que é introduzida a restrição de o código possuir um giro superior a 4. O código resultante possui taxa $R = ½$. A figura 4.3 ilustra a construção 1A. O número 3 denota a sobreposição de 3 matrizes de

permutação e a seta circular representa permutações aleatórias de todas as colunas naquele bloco.

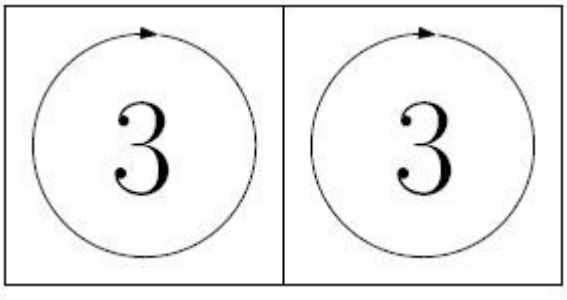


figura 4.3: Construção 1A para um código de Mackay (3,6)-regular.

A estratégia 2A baseia-se na construção da matriz **H** com *M*/2 colunas (com *M* sendo o número de linhas da matriz **H**) de peso 2, sem que exista qualquer 1 em comum entre elas, e em que as restantes colunas têm peso 3. Simultaneamente, procura-se que o peso d_c seja o mais uniforme possível, impondo como restrição o fato de quaisquer duas colunas de **H** não terem mais do que um 1 em comum. Consiste, pois numa implementação da estratégia IV anteriormente mencionada. O código resultante possui taxa $R = 1/3$. A construção irregular foi introduzida por apresentar bons resultados empíricos. No próximo capítulo, a diferença de performance entre códigos regulares e irregulares será mostrada. A figura 4.4 mostra a construção 2A. As linhas diagonais representam matrizes de identidade.

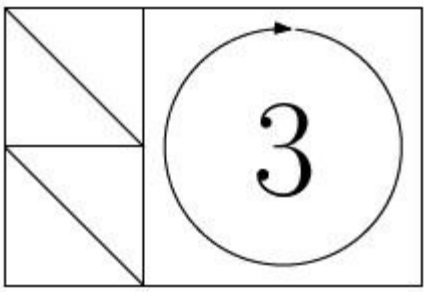


figura 4.4:Construção 2A para um código de Mackay irregular.

As estratégias 1B e 2B resultam da eliminação de um pequeno número de colunas das matrizes **H** obtidas segundo as estratégias 1A e 2A, respectivamente, de forma a eliminar todos os ciclos do código de tamanho inferior a um dado valor *l* estipulado, procurando, desta forma, maximizar o giro de acordo com a estratégia V.

Exemplo de Códigos de Mackay: Considere o código de Mackay (3,6)-regular de comprimento 96 e taxa $R = ½$ construído a partir da técnica 1A. A sua matriz de verificação de paridade é mostrada na figura 4.5, na qual os pontos pretos representam os elementos não-nulos e os elementos nulos são deixados em branco.

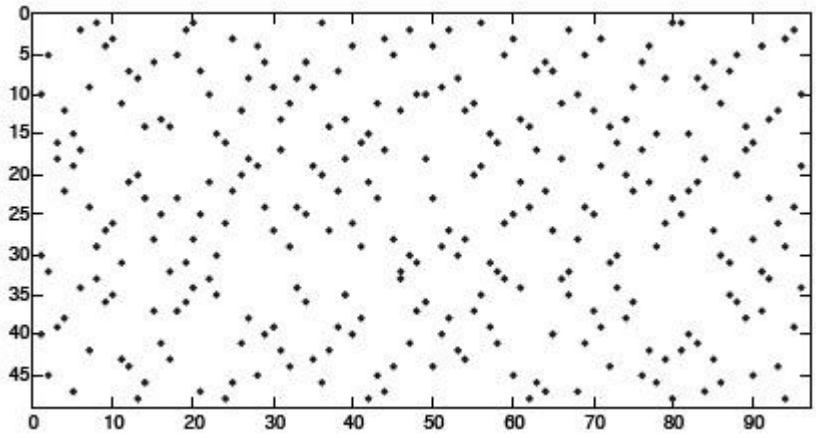


figura 4.5: Matriz H de um código de Mackay (3,6)-regular

E um exemplo de uma matriz de verificação de paridade, construída com o uso da técnica 2A, é mostrada na figura 4.6. Este código possui comprimento 96 e taxa $R = 1/3$.

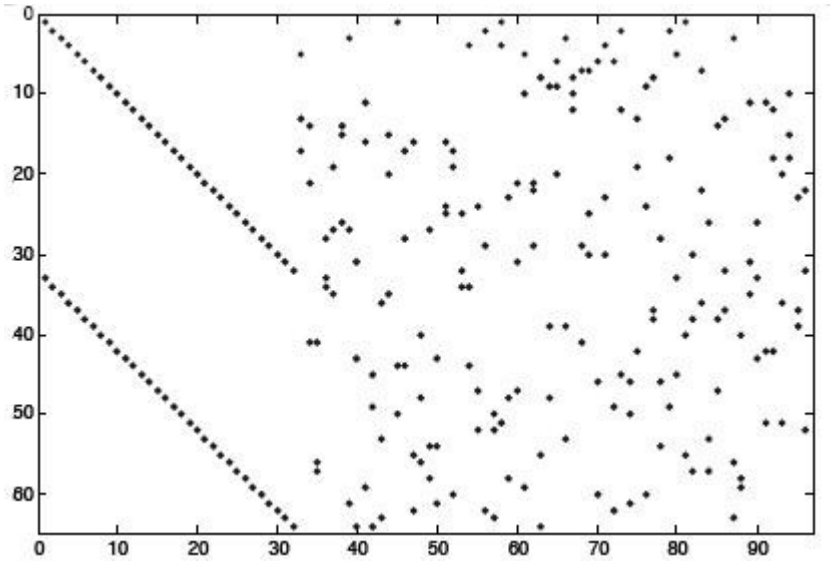


figura 4.6: Matriz H de um código de Mackay irregular

Um dos problemas que é colocado é exatamente a forma de eliminar os ciclos de comprimento inferior a um dado valor. McGowan e Williamson [9] baseados no conceito de matriz adjacente, definida anteriormente, apresentam um método para remoção desses ciclos por alteração da matriz **H**, garantindo, simultaneamente, que não são gerados novos ciclos do mesmo tamanho ou inferior.

Como já foi referido, seguindo qualquer uma destas estratégias, o conjunto de códigos admissíveis é muito grande, tornando-se necessário dispor de alguns critérios de seleção por forma a escolher os susceptíveis de apresentarem um melhor desempenho, que é o principal objetivo deste trabalho. Para isso, comparações entre alguns códigos propostos serão realizadas a fim de observar o de melhor performance.

Mao e Banihashemi [8] sugerem, como critério de seleção a escolha do código que o apresente o giro mais elevado. Este método implica a determinação do giro de cada um dos códigos o que coloca, desde logo, limitações à sua aplicação em comprimentos de bloco elevados. Estes autores mostraram que para códigos obtidos segundo o método de construção 2A, o critério de seleção por eles propostos, conduzia à escolha de um código com uma menor probabilidade de erro de decodificação.

Um outro critério de seleção consiste na escolha do código que possui uma maior distância mínima (sem dúvida, um bom parâmetro de avaliação do desempenho de um código linear). A determinação da distância mínima de um código linear é, no entanto, uma tarefa complexa para códigos com comprimentos de bloco elevados. Berrou, Vaton, Jézéquel e Douillard [10] apresentam um método iterativo para a determinação da distância mínima de um código linear, que designam por *Método do Erro Impulsivo*. O método baseia-se na resposta de um de codificador iterativo de máxima verossimilhança do tipo *Soft-In / Soft-Out* a um erro do tipo impulsivo (A palavra-código transmitida é apenas corrompida num bit com ruído impulsivo de amplitude A_i) num canal Gaussiano. Os autores provaram que a amplitude mínima de ruído que provoca um erro de decodificação é igual à distância mínima do código em causa.

Além destes, Richardson e Urbanke [11] introduziram um método amplamente utilizado atualmente que leva em consideração a análise das densidades de probabilidade das mensagens iteração por iteração, que são funções do nível de ruído do canal e dos graus dos nós de verificação e de bits (símbolos). Este método chama-se *Density Evolution*, que

busca definir o limite de nível de ruído de um determinado conjunto de códigos, chamado de *limiar*, a partir do qual a decodificação iterativa SP não converge, ou seja, até este valor o código terá um desempenho livre de erros. O tamanho do bloco influencia na distância entre o desempenho do código e o limiar do seu conjunto de códigos, pois, conforme diminui o seu tamanho, essa distância aumenta. No final, através desse método, concluíram que aumentando os graus dos nós de bits (símbolos), o limiar de decodificação também aumenta, mostrando que com códigos irregulares é possível alcançar o limite do canal de Shannon.

As técnicas de construção apresentadas até agora se baseiam diretamente no projeto das matrizes de verificação de paridade. No entanto, um código LDPC também pode ser construído através do projeto do GT que o descreve, conforme foi mencionado anteriormente. Para se obter um bom código, é sempre desejável construir um GT que possua um ciclo mínimo grande. A construção de GTs com o maior ciclo-mínimo possível é um difícil problema de difícil análise combinatorial. Entretanto, é possível projetar GTs com um ciclo mínimo de comprimento relativamente grande através de algoritmos sub-ótimos. O algoritmo Progressive Edge Growth (PEG) constrói grafos colocando ramos progressivamente entre BN's e CN's em um GT de modo a maximizar os ciclos mínimos locais de cada um dos BN's.

Dados os parâmetros do grafo, isto é, o número de BN's N, o número de CN's M e a distribuição de grau dos BN's $\lambda(x)$, um procedimento que adiciona ramos é realizado de modo que cada novo ramo adicionado ao grafo possua o menor impacto possível no ciclo mínimo. A idéia do algoritmo é, para cada BN, encontrar os CN's mais distantes e então conecta-los através de ramos. Para encontrar os CN's mais distantes de um determinado BN x_n, é construído um subgrafo partindo de x_n e dos ramos nele incidentes (x_n, c_{m1}),(x_n, c_{m2}), $\cdots$, $(x_n, c_{md(xn)})$ são incluídos no subgrafo, juntamente com todos os ramos incidentes neles, excluindo (x_n, c_{m1}),(x_n, c_{m2}), $\cdots$, $(x_n, c_{md(xn)})$. O processo continua até que uma determinada *profundidade* seja alcançada.

Neste procedimento, o subgrafo pode conter vários vértices e ramos iguais. Para um BN x_n, a sua *vizinhança de profundidade l* é definida como o conjunto de CN's alcançados pelo subgrafo de profundidade l expandido como mostrado na figura 2.13 e é denotada por

$N^l(x_n)$. O conjunto complementar $\overline{N}^l(x_n)$ é definido como $V_c \setminus N^l(x_n)$. Na figura 4.7, qualquer BN que aparece pela primeira vez na profundidade *l* está a uma distância 2*l* em relação ao nó x_n e qualquer CN que aparece pela primeira vez na profundidade *l* está a uma distância de 2*l* + 1 em relação ao nó x_n.

Ao expandir progressivamente o subgrafo a partir do BN x_n duas situações podem ocorrer:

1) a cardinalidade do conjunto $N^l(x_n)$ para de aumentar e é menor do que *M*;

2) $N^l(x_n) \neq \emptyset$ e $\overline{N}^{l+1}(x_n) = \emptyset$.

No primeiro caso, nem todos os CN's podem ser alcançados a partir de x_n, então conectar x_n a um dos elementos de $N^l(x_n)$ não cria nenhum ciclo adicional. No segundo caso, todos os CN's são alcançados em uma profundidade *l* + 1, então a conexão de x_n a um elemento de $\overline{N}^l(x_n)$ cria um ciclo de comprimento 2(*l* + 2).

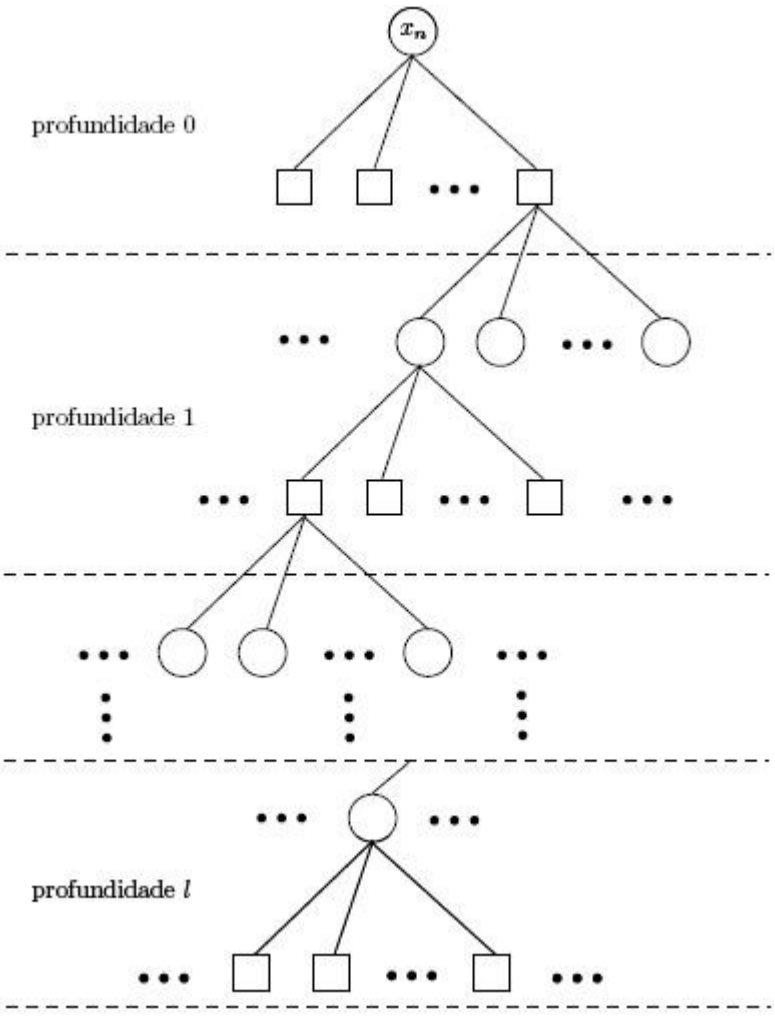


figura 4.7: subgrafo expandido a partir do nó x.

Quando o algoritmo encontra mais de um CN para conectar ao BN x_n, isto é, $\overline{N}^{l}(x_n)$ possui mais de um elemento, o nó com o menor grau é selecionado. Tal procedimento faz com que a distribuição de graus dos CN's seja tão uniforme quanto possível. Se houverem múltiplas escolhas de nós de menor grau, é possível escolher aleatoriamente um dos nós do conjunto (com probabilidade uniforme) ou escolher o CN com o menor índice do conjunto.

Exemplo de código construído com o algoritmo PEG: Usando o algoritmo PEG para construir um código LDPC de comprimento N = 504, M = 252 equações de paridade e distribuição de graus $\lambda(x) = x^2$, obtém-se a matriz de verificação de paridade mostrada na figura 4.8.

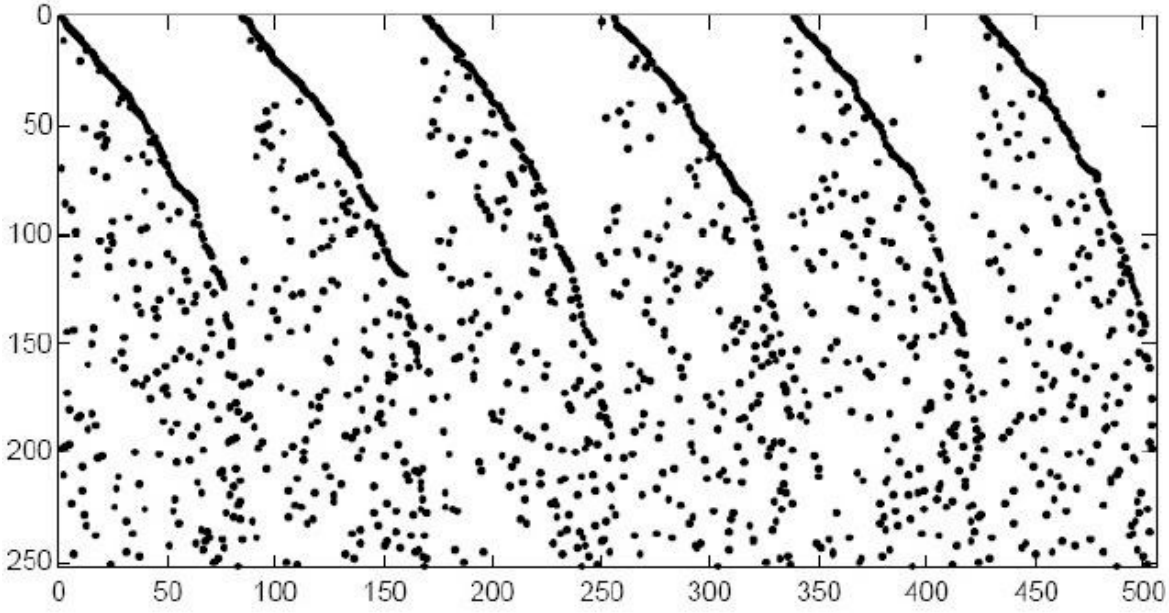


figura 4.8: Matriz H de um código LDPC construído com o algoritmo PEG.

O algoritmo PEG possui a vantagem de ser bastante flexível e possibilita obter códigos com uma estrutura que permite um codificador de complexidade linear. Além disso, o algoritmo PEG permite obter códigos regulares e irregulares de qualquer taxa, com um bom ciclo mínimo. Mesmo os códigos regulares podem possuir uma leve irregularidade nos graus de CN's. No entanto, isto não influencia significativamente o seu desempenho.

- *Abordagens alternativas para construção de códigos LDPC.*

Existem outras formas de construção de códigos LDPC. Por exemplo, Prabhakar e Narayanan [14] apresentam um método algébrico de construção de códigos LDPC regulares. Este apresenta como grande vantagem o fato de a estrutura de o GT poder ser gerada algebricamente usando um procedimento recursivo.

Este método algébrico pretende construir um código LDPC regular de dimensões (n,k), sendo d_s o peso de cada BN e o peso de d_c *de* cada CN. Existem n x d_s ligações a BN's e a CN's em que cada caminho do GT liga uma conexão BN a uma conexão CN. Designando por A_i a conexão CN à qual liga a conexão BN_i, com A_i pertencente ao BN

[i/d_s], então, Prabhakar e Narayanan propuseram que a seqüência A_0, A_1, ⋯ , A_{nxds-1} fosse obtida da seguinte forma recursiva:

$$A_{i+1} = (a.A_i + b) \bmod M \quad \text{com} \quad M = n \text{ x } d_s \quad \text{e} \quad 0 \leq i \leq M \,, \tag{4.1}$$

com a e b números inteiros satisfazendo um conjunto de condições por eles deduzidas e A_0 um número inteiro entre 0 e M.

Ao todo são nove as restrições colocadas por Prabhakar e Narayanan que garantem que o código gerado por este método é LDPC e não possui ciclos de comprimento inferior a seis. Estas condições não são muito restritivas, permitindo que seja gerado um vasto conjunto de códigos LDPC com diferentes comprimentos e taxas de informação. O desempenho do código gerado depende também dos valores escolhidos para a, b e A_0.

Uma abordagem alternativa para a construção de bons códigos LDPC irregulares é apresentada por Richardson, Shokrollahi e Urbanke [12]; e Chung, Forney, Richardson e Urbanke [13] que se baseia num estudo sobre a distribuição dos pesos de cada BN que otimiza o desempenho do código.

Capítulo 5 – Codificação e Decodificação de LDPC

5.1 - Codificação

Agora que foi introduzida a teoria de códigos LDPC, será mostrada como é realizada a codificação a partir da matriz esparsa gerada.

Sendo os códigos LDPC, a forma imediata de realizar a codificação seria partindo do conhecimento da matriz geradora determinar as palavras-código de acordo com a equação (2.8), ou seja, fazendo $x = mG$. No entanto, os métodos utilizados na construção de códigos LDPC assentam na obtenção da sua matriz de teste de paridade **H** ou no equivalente GT. Na grande maioria das vezes a matriz **H** obtida não é sistemática nem de característica máxima. A matriz pode, no entanto, ser expressa na forma sistemática (2.9) usando o método de Gauss. A obtenção da matriz **G** é depois imediata. Podem, no entanto, surgir situações em que para obter **H** na forma (2.9) se torne necessário efetuar troca de colunas, obtendo-se desta forma um código LDPC diferente mas com o mesmo desempenho do original.

Embora de fácil implementação, o método anterior é extremamente dispendioso em termo do número de operações a realizar na codificação de cada palavra de código. Isto se deve ao fato de a sistematização da matriz **H** não conduzir, necessariamente, à obtenção de uma matriz **G** com baixa densidade de 1's, pelo que a codificação por (2.8) exige a realização de um número extremamente elevado de operações.

Uma aproximação alternativa para a solução deste problema consiste na obtenção de códigos LDPC por métodos algébricos e geométricos em que a codificação possa ser realizada por circuitos simples baseados em registros de deslocamento. É o caso dos códigos apresentados por Johnson e Weller [15], que sugerem uma família de códigos LDPC quase cíclicos com giro não inferior a 6. É também o caso de Kou, Lin e Fossorier [16] que apresentam uma abordagem para a construção de códigos LDPC baseada em linhas e pontos de uma geometria finita, como a geometria Euclidiana e a geometria projetiva em campos finitos. Os códigos construídos por este método são também cíclicos ou quase cíclicos.

5.2 - Decodificação "Hard e Soft decision"

Uma das principais vantagens dos códigos LDPC é o fato de poder ser decodificado usando algoritmo iterativo, que tem a complexidade crescente linearmente com o tamanho do código.

O algoritmo usado para decodificar os códigos LDPC foi descoberto independentemente várias vezes e aparece com diferentes nomes. Os mais comuns são "belief propagation" (BA), "message passing algorithm" (MPA) e "sum-product algorithm" (SPA), que é o mais utilizado em teoria de comunicação.

Para este tipo de algoritmo, existem duas formas possíveis: "hard decision" (HD) e soft decision" (SD). O primeiro considera que o número de símbolos possíveis que chegam ao decodificador é finito, e o segundo trabalha com a distribuição probabilística dos símbolos recebidos. Apesar do HD ser mais simples, o SD apresenta melhor desempenho por levar em consideração uma distribuição de probabilidades.

Com o objetivo de explicar este algoritmo, uma simples variação, que funciona com HD será introduzido primeiro. Em seguida, o algoritmo será estendido para trabalhar com SD, que geralmente obtém melhores resultados na decodificação. Apesar de ser utilizado o canal AWGN para as simulações mais adiante, para este algoritmo ser mais bem entendido, será considerado o canal binário simétrico.

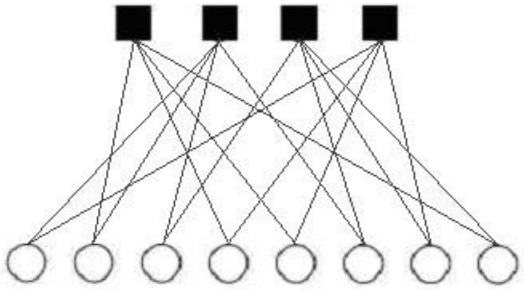
"Hard decision":

O algoritmo será explicado baseado na seguinte matriz de verificação de paridade. A matriz mostrada abaixo tem dimensão 4 x 8 com $d_s = 4$ e $d_c = 2$.

$$H: \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

representada pelo seu GT:

Sendo c = [1 0 0 1 0 1 0 1] uma palavra-código livre de erros para a matriz H acima (satisfaz a condição $Hc^T = 0$), supõe-se sua transmissão em um canal binário simétrico e a recepção com um erro, por exemplo, de $c_1 = 1$, ou seja, c' = [1 1 0 1 0 1 0 1].



1. No primeiro passo, todos os nós de variáveis (BN's) c_i enviam uma "mensagem" para seus nós de verificação (CN') f_j (sempre 2 neste exemplo) contendo o bit que acreditam ser o correto para si. Neste estágio, a única informação que um BN c_i tem, é o correspondente i-ésimo bit recebido de c, y_i. Isso significa, por exemplo, que c_0 envia uma mensagem contendo 1 para f_1 e f_3, nó c_1 envia mensagem contendo $y_1(1)$ para f_0 e f_1, e por aí vai.

2. No segundo passo, todo CN f_j calcula uma resposta para todos os BN's conectados. A resposta contem o bit que f_j acredita ser o correto para o BN c_i assumindo que os outros BN's conectados a si (f_j) estão corretos. Em outras palavras: Se verificar o exemplo, todo CN f_j está conectado a 4 BN's. Então, um CN f_j olha para a mensagem recebida de três BN's e calcula o bit que o quarto BN deve ter para satisfazer a equação de verificação de paridade. A tabela 5.1 mostra este passo.

Tabela 5.1: lista das mensagens recebidas e enviadas pelos nós de verificação no passo 2 do algoritmo de decodificação hard decision.

c-node	received/sent				
f_0	received:	$c_1 \to 1$	$c_3 \to 1$	$c_4 \to 0$	$c_7 \to 1$
	sent:	$0 \to c_1$	$0 \to c_3$	$1 \to c_4$	$0 \to c_7$
f_1	received:	$c_0 \to 1$	$c_1 \to 1$	$c_2 \to 0$	$c_5 \to 1$
	sent:	$0 \to c_0$	$0 \to c_1$	$1 \to c_2$	$0 \to c_5$
f_2	received:	$c_2 \to 0$	$c_5 \to 1$	$c_6 \to 0$	$c_7 \to 1$
	sent:	$0 \to c_2$	$1 \to c_5$	$0 \to c_6$	$1 \to c_7$
f_3	received:	$c_0 \to 1$	$c_3 \to 1$	$c_4 \to 0$	$c_6 \to 0$
	sent:	$1 \to c_0$	$1 \to c_3$	$0 \to c_4$	$0 \to c_6$

Importante é que o algoritmo pode terminar neste ponto. Acontecerá isto se todas as equações de verificação de paridade forem satisfeitas. Será observado mais tarde que o algoritmo contem um loop, sendo outra possibilidade de término do algoritmo, alcançar um número máximo de loops.

3. Próxima fase: os BN's recebem as mensagens dos CN's e usam como informação adicional para decidir se seu bit originalmente recebido está certo. Uma forma simples de fazer isto é escolher a maioria. Voltando ao exemplo, cada BN tem três fontes de informação a respeito de seu bit. O bit original recebido e duas sugestões dos CN's. A tabela 5.2 ilustra este passo. Agora, os BN's podem enviar uma outra mensagem que eles decidiram ser o correto pelos CN's.
4. Vá para o passo 2.

Tabela 5.2:Descreve o passo 3. Os BN's usam as mensagens de resposta dos CN's para escolher pela maioria o valor de seu bit.

v-node	y_i received	messages from check nodes		decision
c_0	1	$f_1 \rightarrow 0$	$f_3 \rightarrow 1$	1
c_1	1	$f_0 \rightarrow 0$	$f_1 \rightarrow 0$	0
c_2	0	$f_1 \rightarrow 1$	$f_2 \rightarrow 0$	0
c_3	1	$f_0 \rightarrow 0$	$f_3 \rightarrow 1$	1
c_4	0	$f_0 \rightarrow 1$	$f_3 \rightarrow 0$	0
c_5	1	$f_1 \rightarrow 0$	$f_2 \rightarrow 1$	1
c_6	0	$f_2 \rightarrow 0$	$f_3 \rightarrow 0$	0
c_7	1	$f_0 \rightarrow 1$	$f_2 \rightarrow 1$	1

Neste exemplo, a segunda execução do passo 2 encerraria o processo de decodificação, pois c_1 escolheu o valor 0 no último passo. Com isso, corrige o erro de transmissão e satisfaz todas as equações de verificação.

"Soft decision":

A decodificação descrita acima foi apenas uma maneira didática de explicar o algoritmo. A decodificação "soft decision" dos códigos LDPC, que é baseada no conceito de "envio de mensagens", tem uma performance melhor e é, por isso, o método mais

utilizado. A idéia é a mesma do "hard decision". Antes de apresentar o algoritmo, serão apresentadas as notações:

- $P_i = Pr(c_i = 1|y_i)$
- q_{ij} é a mensagem enviada por um BN c_i para o CN f_j. Toda mensagem contem sempre um par $q_{ij}(0)$ e $q_{ij}(1)$ que carrega o quanto se acredita y_i ser 0 ou 1.
- r_{ji} é a mensagem enviada por um CN f_j para o BN c_i. Novamente, existe um $r_{ji}(0)$ e $r_{ji}(1)$ que indica o quanto se acredita y_i ser 0 ou 1.

O número de passos na descrição a seguir corresponde ao caso do "hard decision".

1. Todos os BN's enviam suas mensagens q_{ij}. Como nenhuma informação está disponível neste passo, $q_{ij}(1) = P_i$ e $q_{ij}(0) = 1 - P_i$.
2. Os CN's calculam suas mensagens de resposta r_{ji}[1]:

$$r_{ji}(0) = \frac{1}{2} + \frac{1}{2}\prod_{i' \in V_j \setminus i} (1 - 2q_{i'j}(1)) \qquad (5.1)$$

e

$$r_{ji}(1) = 1 - r_{ji}(0) \qquad (5.2)$$

Então, é calculada a probabilidade de existir um número par de 1's entre os BN's, exceto c_i (isto é exatamente o que $V_j \setminus i$ significa). Esta probabilidade é igual à probabilidade de $r_{ji}(0)$, ou seja, de c_i ser 0. Este passo e a informação usada para calcular as resposta estão ilustrados na figura 5.1.

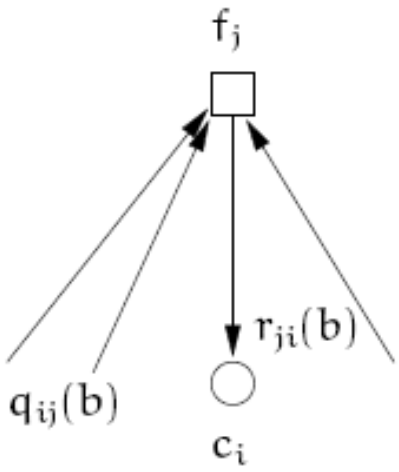


figura 5.1: ilustra o cálculo de $r_{ji}(b)$

---

[1]Equação (5.1) usa o seguinte resultado de Gallager: para uma seqüência de M dígitos binários independentes a_i com uma probabilidade de p_i para $a_i = 1$, a probabilidade de toda a seqüência conter um número par de 1's é

$$\frac{1}{2} + \frac{1}{2}\prod_{i=1} (1 - 2p_i)$$

as constantes Kij são escolhidas de uma forma que assegure qij(0) + qij(1) = 1. Ci\j agora significa todos os CN's exceto fj. A figura 5.1 ilustra o cálculo neste passo.

3. Os BN's atualizam as suas mensagens de resposta para os CN's. Isto é feito de acordo com as seguintes equações:

$$q_{ji}(0) = K_{ij}(1 - P_i)\prod_{j' \in Ci\setminus j} r_{j'i}(0)$$

(5.2)

e

$$q_{ji}(1) = K_{ij}(P_i)\prod_{j' \in Ci\setminus j} r_{j'i}(1) \qquad (5.3)$$

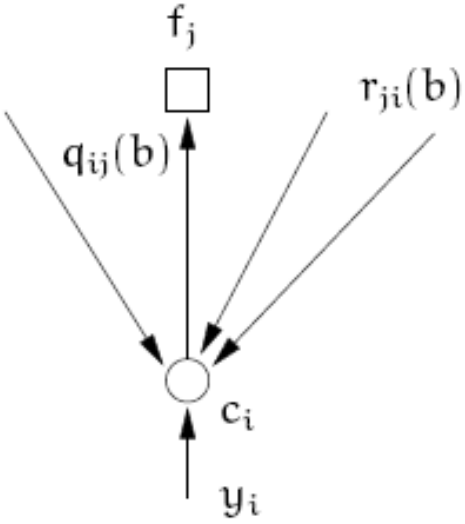


figura 5.2: ilustra o cálculo de $q_{ij}(b)$

Neste ponto, os BN's também atualizam sua estimação corrente $\hat{c}_i$ de sua variável c_i. Isto é feito calculando as probabilidades de ser 0 e 1 e escolhendo a maior delas. As equações usadas são

$$Q_i(0) = K_{ij}(1 - P_i)\prod_{j \in Ci} r_{ji}(0) \tag{5.4}$$

e

$$Q_i(1) = K_{ij}(P_i)\prod_{j \in Ci} r_{ji}(1) \tag{5.5}$$

são quase as mesmas usadas para calcular $q_{ij}(b)$, mas agora a informação de cada CN é usada.

$$\hat{c}_i : \begin{cases} 1 & \text{se } Q_i(1) > Q_i(0), \\ 0 & \text{contrário} \end{cases} \tag{5.6}$$

Se a palavra-código estimada satisfizer as equações de verificação de paridade o algoritmo é encerrado.

4. Voltar para o passo 2.

Este algoritmo de decodificação “soft-decision” explicado é uma variante muito simples, utilizada em canais binários simétricos e pode ser modificado com melhorias de performance. Como não é o objetivo deste trabalho discutir os algoritmos de decodificação, foi citada esta simplificação apenas como ilustração. Além da performance, existem numerosos problemas de estabilidade devido a muitas multiplicações de probabilidades. Os resultados virão muito próximos de 0 para um comprimento de bloco muito grande. Para prevenir isto, é possível mudar para o domínio logarítmico e fazer adições ao invés de multiplicações. O resultado é um algoritmo mais estável e uma melhor performance, pois adições são mais simples que multiplicações.

Capítulo 6 - Simulações

Neste capítulo, serão mostrados os resultados de simulações dentre variados algoritmos de código LDPC, pois, o objetivo deste trabalho é observar a diferença de performance de alguns tipos de algoritmos de codificação LDPC. Para isto, foram estudados diferentes tipos de algoritmos e escolhidos 5 deles: 3 regulares e 2 irregulares, que foram explicados no capítulo anterior. Os algoritmos foram escolhidos para mostrar a evolução dos tipos de códigos LDPC sugeridos ao longo do tempo. Seriam os algoritmos regulares: o introduzido pelo Gallager [4] na década de 60, o apresentado pelo Makay e Neal [7] nos anos 90 (o número VI, comentado no capítulo anterior) e o construído através do algoritmo PEG; e os irregulares: o 2A apresentado por Makay e Neal [7] e o PEG irregular.

Para poder visualizar a diferença de desempenho, todos os códigos foram introduzidos em um canal RAGB e decodificados com o mesmo algoritmo iterativo SP (também apresentado no capítulo anterior), pois é um dos mais utilizados atualmente. Para a decodificação, foram utilizadas 20 iterações, por que se verificou que o desempenho pouco se altera em quantidades acima disso. Além disso, foram utilizados 4 tamanhos de matrizes H para verificar o impacto no desempenho de cada código. Os tamanhos escolhidos são: 48x96, 252x504, 504x1008 e 1008x2016.

O primeiro gráfico apresentado é com as comparações de performance dos códigos com a matriz de tamanho 48x96.

A partir dos resultados, pode-se observar que os algoritmos regulares atingiram melhor performance em comparação aos códigos irregulares. Conforme foi citado em capítulo anterior, os códigos regulares funcionam melhor que os irregulares em blocos curtos por apresentarem, em geral, ciclos mais longos e distância mínima maior. Além disso, percebe-se que o código PEG regular mostrou uma superioridade sobre os demais códigos também regulares, pois a matriz de paridade é construída evitando ciclos de tamanho 4. Os demais códigos regulares, o de Gallager e Mackay, apresentam ciclos curtos, pois não houve a preocupação de eliminá-los.

Por outro lado, o código PEG irregular, apesar de também ter sido construído evitando ciclos curtos, não obteve performance melhor que os regulares de Gallager e Mackay, provavelmente devido ao grau de densidade elevado para o tamanho da matriz. Por último, o código de Mackay irregular que foi a pior performance, por que, assim como seu código regular, não foram eliminados os ciclos curtos. Fora isso, Mackay, quando introduziu os códigos irregulares, não apresentou uma forma ótima de construção da matriz, chegando a melhor resposta de forma empírica, realizando simulações de diversas matrizes.

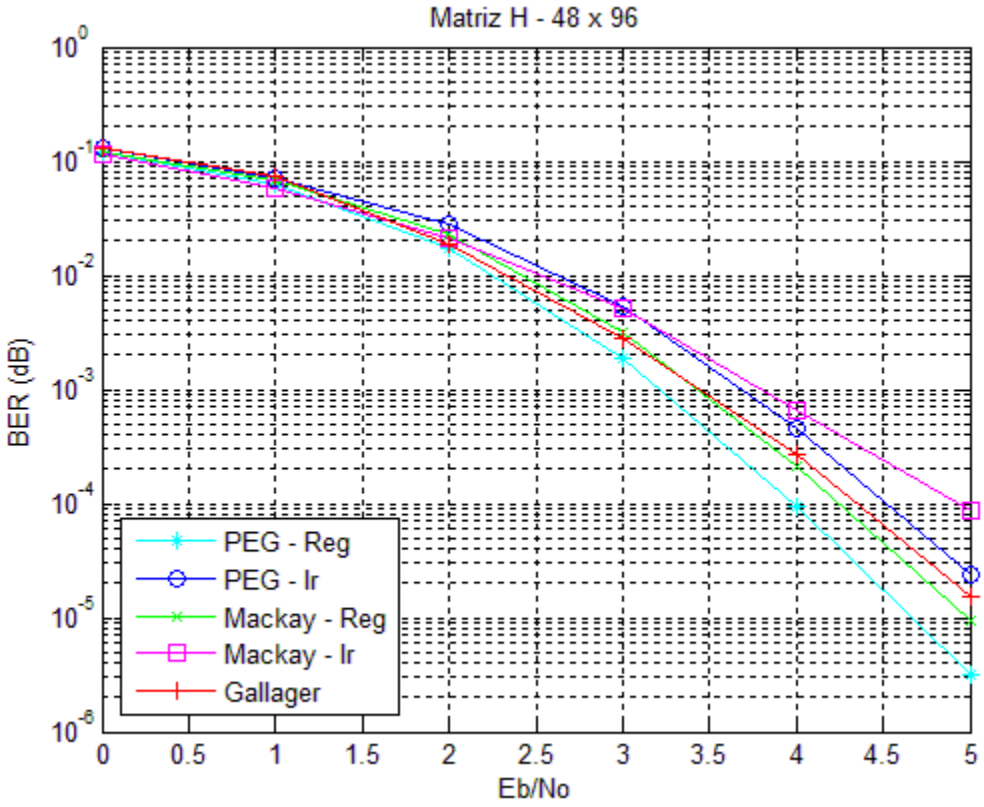


Gráfico 6.1 – Simulações com matrizes de tamanho 48x96.

Nos códigos de tamanho 252x504, novamente o de melhor performance foi o PEG. Percebe-se que os códigos regulares para este tamanho de matriz apresentam superioridade em relação aos irregulares. No caso do PEG irregular, a performance fica bem próxima dos códigos regulares de Gallager e de Mackay, sendo superior quando estes últimos apresentam ciclo de tamanho 4. Mais uma vez, os códigos irregulares de Mackay foram os piores, confirmando o argumento de que a forma de construção da matriz H tem bastante

importância para seu desempenho, sendo ainda reforçado pelo fato da eliminação dos ciclos de tamanho 4 do código, que praticamente não influencia no resultado.

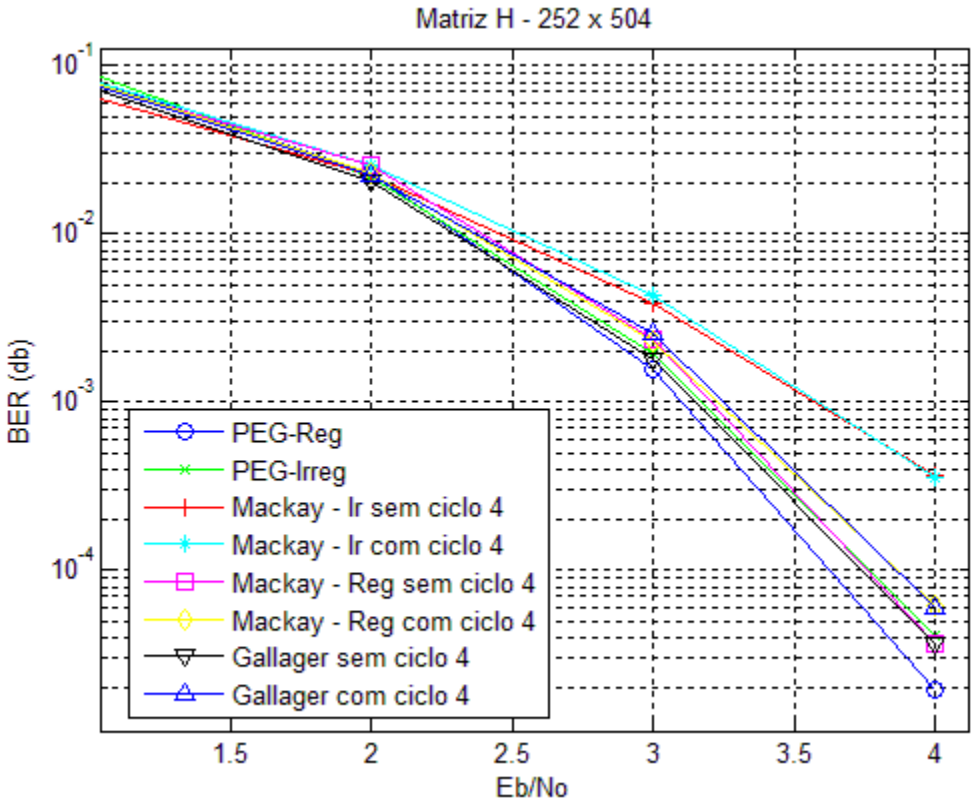


Gráfico 6.2 – Simulações com matrizes de tamanho 252x504.

Em seguida, os códigos de 504x1008, que tiveram os resultados semelhantes aos apresentados anteriormente (tamanho 252x504). Observa-se que, para este tamanho de matriz, a regularidade dos códigos ainda apresenta superioridade sobre os irregulares. De qualquer forma, o código de PEG irregular apresenta uma melhora significativa em comparação aos tamanhos de matrizes anteriores. Isto mostra que, aumentando-se o tamanho do bloco de informação, os códigos irregulares tendem a ser mais vantajosos do que os regulares.

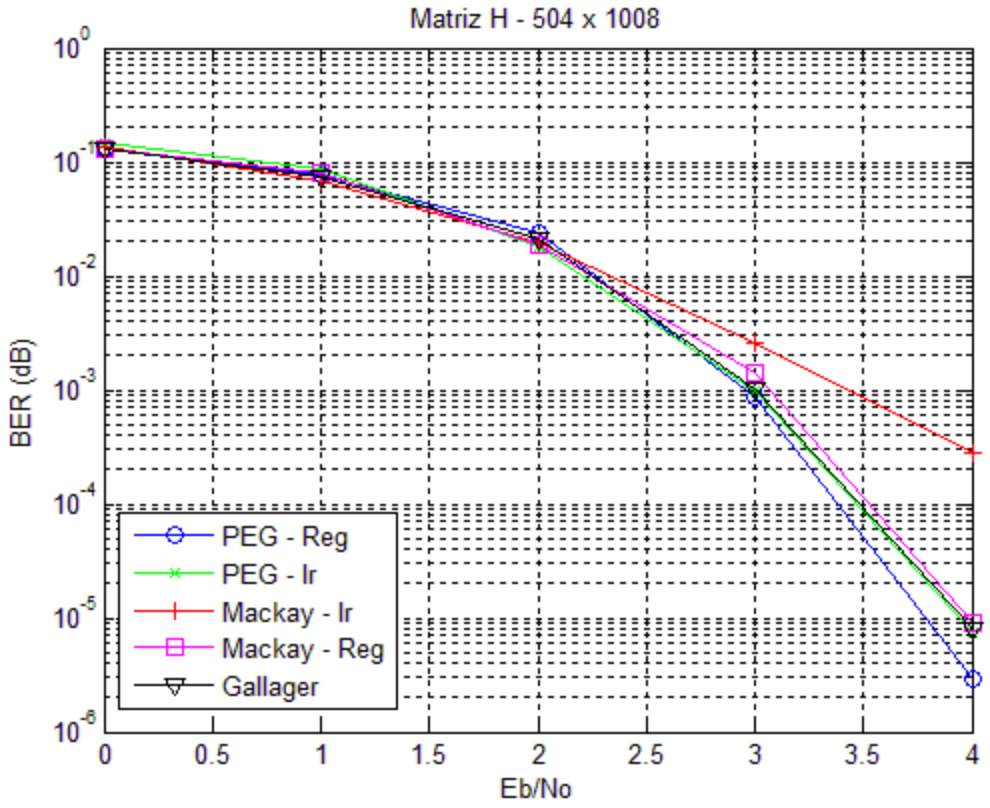


Gráfico 6.3 – Simulações com matrizes de tamanho 504x1008.

Por último, os códigos maiores, de tamanho 1008x2016, que apresentaram uma peculiaridade: os algoritmos regulares não funcionam satisfatoriamente. Os três algoritmos obtiveram uma degradação muito grande se comparados com os tamanhos de matrizes anteriores. Por outro lado, os algoritmos irregulares melhoraram a performance e foram satisfatórios, confirmando a ideia de que, para blocos cada vez mais longos, os códigos irregulares melhoram ainda mais a sua performance por apresentarem os graus dos nós de variáveis, devido a isso, se conseguir diminuir a distância entre o limiar de decodificação e o limite do canal.

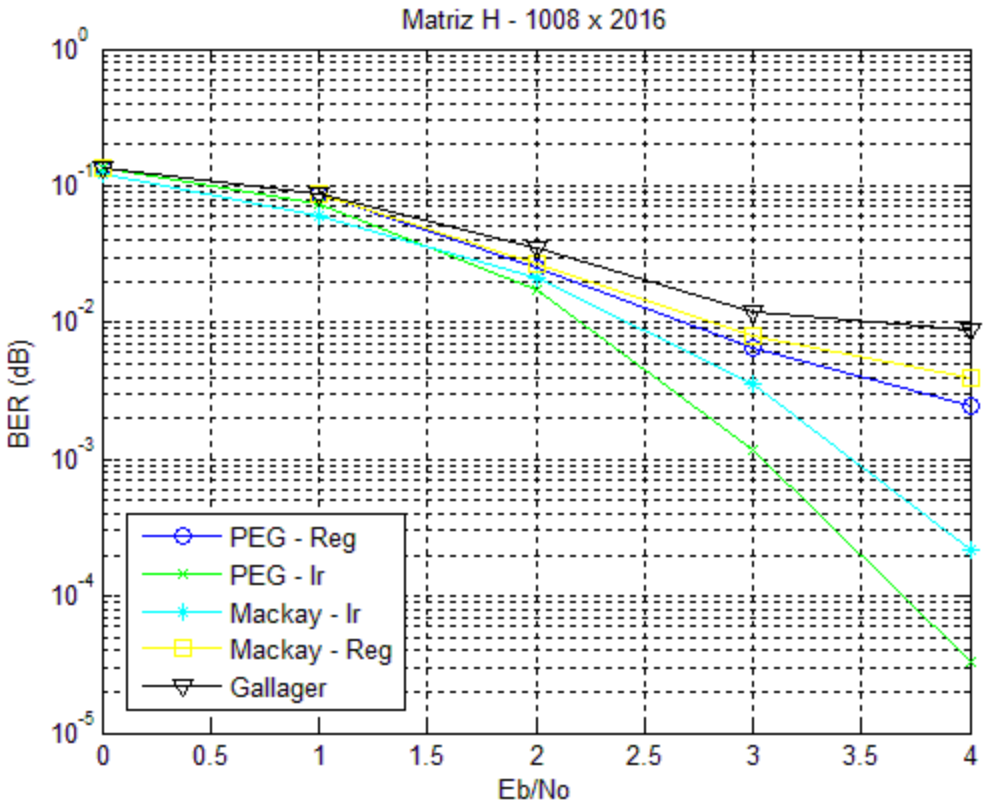


Gráfico 6.4 – Simulações com matrizes de tamanho 1008x2016.

Capítulo 7 - Conclusões

A motivação deste trabalho foi estudar um dos códigos corretores de erro mais utilizados atualmente, o LDPC, que, apesar de apresentado desde a década de 60, somente passou a ser largamente utilizado depois dos anos 90.

Dentro deste tema, por haver uma variedade de assuntos, aplicações e tipos de códigos LDPCs apresentados ao longo dos anos, o trabalho se restringiu a realizar uma comparação de desempenho entre alguns códigos regulares e irregulares, a partir da observação das taxa de erros de bits recebidos para 4 diferentes tamanhos de matrizes.

Antes das simulações, realizadas através de rotinas implementadas ou adaptadas em MatLab, foram apresentadas as teorias de códigos corretores de erros e de LDPC especificamente, além de citados algoritmos sugeridos por alguns autores, dos quais 5 foram escolhidos para a comparação.

Baseado nos resultados mostrados no capítulo anterior, pode-se confirmar o que havia sido citado na teoria, na qual os códigos regulares costumam apresentar desempenhos melhores em tamanhos menores (distância mínima maior), mas que, conforme se vai aumentando o código, os códigos irregulares se tornam mais interessantes. Além disso, vale destacar o fato de que na tentativa e erro da construção das matrizes de paridade irregulares (destacado por Mackay), é possível se alcançar distribuições de graus cada vez melhores, o que passa a ser o desafio para os profissionais que estudam este código.

Referências Bibliográficas

[1] C. E. Shannon; *A mathematical theory of communication*, Bell System Technical Journal, v. 27, p. 379-423 e 623-656, Julho e Outubro, 1948.

[2] S. Haykin; *Communications Systems*. 4.ed. John Wiley & Sons. EUA, 1994.

[3] Irving Reed e Solomon Golomb; *Polynomial codes over certain finite fields*, Joint Society of Industrial and Applied Mathematics Journal 8 (1960), no. 2, 300–304.

[4] R. G. Gallager; *Low-Density Parity-Check Codes*. Tese de PhD, Massachusetts Institute of Technology. EUA, 1963.

[5] R. M. Tanner; *A Recursive Approach to Low Complexity Codes*. IEEE Transactions on Information Theory, vol., IT-27, no. 25. 1981.

[6] Berrou, A. Glavieux, e P. Thitimajshima; *Near Shannon limit error correcting coding and decoding: Turbo codes*. In Proceedings of the IEEE International Conference on Communications, Geneva, Switzerland, May 2003.

[7] D.J.C. Mackay e R.M. Neal; *Near Shannon Limit performance of low density parity check codes*, Eletronic Letters, v.32, p. 1645-1646, 1996.

[8] Y. Mao e A. H. Banihashemi; *A Heuristic Search for Good Low-Density Parity-Check Codes at Short Block Lenghts*, *IEEE ICC 2001*, Filândia, Junho 2001.

[9] J. A. McGowan e R. C. Williamson; *Loop Removal from LDPC Codes*, IEEE Information Theory Workshop 2003, Paris, 2003.

[10] C. Berrou, S. Vaton, M. Jézéquel e C. Douillard; *Computing the Monimum Distance of Linear Codes by the Error Impulse Method*, IEEE GLOBECOM ‘02, Taipei, Taiwan, Novembro 2002.

[11] T. Richardson e R. Urbanke; *The Capacity of Low-Density Parity-Check Codes Under Message-Passing Decoding*, IEEE Transactions on Information Theory, vol. 47, nº 2, pp. 599-618, Fevereiro 2001.

[12] T. Richardson, M. Shokrollahi e R. Urbanke; *Design of Capacity Approaching Irregular Low-Density Parity-Check Codes*, IEEE Transactions on Information Theory, vol. 47, nº 2, pp. 619-637, Fevereiro 2001.

[13] S. Chung, G.Forney, T. Richardson e R. Urbanke; *On the Design of Low-Density Parity-Check Codes within 0.0045 dB of the Shannon Limit*, IEEE Communications Letters, vol. 5, nº 2, pp. 58-60, Fevereiro 2001.

[14] A. Prabhakar e K.R. Narayanan; *Pseudorandom Construction of LDPC codes using Linear Congruential Sequences*, IEEE Transactions on Communications, vol. 50, nº 9, pp. 1389-1396, Setembro 2002.

[15] S. Johnson e S. R. Weller; *Quasi-cyclic LDPC codes from difference families*, 3rd Australian Communications Theory Workshop, Canberra, February 4-5, 2002.

[16] Y. Kou, S. Lin e M. P. C. Fossorier; *Low-Density Parity-Check Codes Based on Finite Geometries: A Rediscovery and New Results*, IEEE Transactions on Information Theory, vol. 47, nº 7, pp. 2711-2736, Novembro 2001.

[17] Reed, Irving S.; *A Class of Multiple-Error-Correcting Codes and Decoding Scheme*, MIT Lincoln Laboratory, 1953.

[18] Golay, M. J. E; *Notes on Digital Coding*, Proc. *IRE* **37**, 657, 1949.

Anexo – Códigos em MatLab utilizados nas simulações

- Script de geração da matriz regular de Gallager:

```
function [H_nocycle, H]=gen_ldpc_gallager(rows,dc,ds)

clear parity_check_aux;
%construct H1 matrix
for i=1:rows
   for j=1+dc*(i-1):dc*i
      parity_check(i,j)=1;
   end
end

%add Hds submatrixes
for k=1:ds-1
   for i=rows*k+1:rows*(k+1)
      for j=1:rows*dc
            parity_check(i,j)=parity_check(i-rows,j);
      end
      end
end

%create a auxiliar matrix
for j=1:rows*dc
   for i=1:rows*ds
      parity_check_aux(i,j)=parity_check(i,j);
   end
end

%permute ds-1 lines on matrix H
for k=1:ds-1
   a=randperm(rows*dc);
   for j=1:rows*dc
      for i=rows*k+1:rows*(k+1)
         parity_check(i,j) = parity_check_aux(i,a(j));
      end
   end
end

H=parity_check;

for i = 1:rows*ds
   % Look for pair of row - column
   for j = (i + 1):rows*ds
       w = and(parity_check(i, :), parity_check(j, :));
       c1 = find(w);
       lc = length(c1);
       if lc > 1

          % If found, flip one 1 to 0 in the row with less number of 1s
          if length(find(parity_check(i, :))) <
length(find(parity_check(j, :)))
```

```
            % Repeat the process until only one column left
            for cc = 1:lc - 1
               parity_check(j, c1(cc)) = 0;
            end
         else
            for cc = 1:lc - 1
               parity_check(i, c1(cc)) = 0;
            end
         end % if

      end % if
   end % for j
end % for i

H_nocycle = parity_check;

save HGal756x1512.mat H
save HGal756x1512no.mat H_nocycle
```

- Script de geração da matriz irregular:

```
function [H] = make_irreg_matrix(N, M, lambda, ro)

H=zeros(M,N);

onesInCols=0;
onesInRows=0;

lambdaAux=lambda;
roAux=ro;

for i=1:length(lambda)
   lambdaAux(2,i) = round(lambda(2,i)*N);
   onesInCols=onesInCols+lambda(1,i)*lambdaAux(2,i);
end
for i=1:length(ro)
   roAux(2,i) = round(ro(2,i)*M);
   onesInRows=onesInRows+ro(1,i)*roAux(2,i);

end

if onesInCols ~= onesInRows
   fprintf('The total number of ones in columns and rows are
differents\n')
   onesInCols
   onesInRows
   break;
else
   cols = randperm(N);
   i=1;
   for j=1:length(lambdaAux)
      while lambdaAux(2,j) ~= 0
```

```
         cols(2,i)=lambdaAux(1,j);
         lambdaAux(2,j)=lambdaAux(2,j)-1;
         i=i+1;
      end

   end
   rows = randperm(M);
   i=1;
   for j=1:length(roAux)
      while roAux(2,j) ~= 0
         rows(2,i)=roAux(1,j);
         roAux(2,j)=roAux(2,j)-1;
         i=i+1;
      end

   end


   colsAux=cols;
   rowsAux=rows;
   count1=0;
   while (size(colsAux,2) ~= 0 & count1<20)
      x=randperm(size(colsAux,2));
      y=randperm(size(rowsAux,2));


      if(H(rowsAux(1,y(1)),colsAux(1,x(1)))~=1)
         count1=0;
         H(rowsAux(1,y(1)),colsAux(1,x(1)))=1;
         colsAux(2,x(1))=colsAux(2,x(1))-1;
         if colsAux(2,x(1))==0
            colsMem=colsAux;
            clear colsAux;
            colsAux(:,1:x(1)-1)=colsMem(:,1:x(1)-1);
            colsAux(:,x(1):size(colsMem,2)-1)=colsMem(:,x(1)+1:size(cols-
Mem,2));

         end


         rowsAux(2,y(1))=rowsAux(2,y(1))-1;
         if rowsAux(2,y(1))==0
            rowsMem=rowsAux;
            clear rowsAux;
            rowsAux(:,1:y(1)-1)=rowsMem(:,1:y(1)-1);
            rowsAux(:,y(1):size(rowsMem,2)-1)=rowsMem(:,y(1)+1:size(rows-
Mem,2));

         end
      else
         count1=count1+1;

            end
      end

end
```

```
while size(colsAux,2)~=0

      x=randperm(size(cols,2));
   y=randperm(size(rowsAux,2));
   if H(rowsAux(y(1)),x(1)) ~= 1
      H(rowsAux(y(1)),x(1))=1;
      changeCol=0;
      for j=1:M
         if (H(j,x(1))==1 & changeCol==0)
             H(j,x(1))=0;
             H(j,colsAux(1,1))=1;
             changeCol=1;
             colsAux(2,1)=colsAux(2,1)-1;
             rowsAux(2,y(1))=rowsAux(2,y(1))-1;
         end
      end
      if colsAux(2,1) == 0
            colsMem=colsAux;
            clear colsAux;
         colsAux(:,1:size(colsMem,2)-1)=colsMem(:,2:size(colsMem,2));
      end
      if rowsAux(2,y(1))==0
            rowsMem=rowsAux;
            clear rowsAux;
            rowsAux(:,1:y(1)-1)=rowsMem(:,1:y(1)-1);
            rowsAux(:,y(1):size(rowsMem,2)-1)=rowsMem(:,y(1)+1:size(rows-
Mem,2));

      end
      end
end
```

- Script de codificação da matriz de paridade:

```
function [c, u, newH] = makedSource(numSource, H, strategy)
% Generate parity check vector bases on LDPC matrix H using sparse LU de-
composition
%
%  dSource : Binary source (0/1)
%  H       : LDPC matrix
%  strategy: Strategy for finding the next non-zero diagonal elements
%            {0} First  : First non-zero found by column search
%            {1} Mincol : Minimum number of non-zeros in later columns
%            {2} Minprod: Minimum product of:
%                         - Number of non-zeros its column minus 1
%                         - Number of non-zeros its row minus 1
%
%  c       : Check bits
%
%
% Copyright Bagawan S. Nugroho, 2007
% http://bsnugroho.googlepages.com

% Get the matric dimension
```

```
[M, N] = size(H);
% Set a new matrix F for LU decomposition
F = H;
% LU matrices
L = zeros(M, N - M);
U = zeros(M, N - M);

% Re-order the M x (N - M) submatrix
for i = 1:M
    if i == 1000
        i1=i
    end
    if i == 2000
            i1=i
    end
   % strategy {0 = First; 1 = Mincol; 2 = Minprod}
   switch strategy

      % Create diagonally structured matrix using 'First' strategy
      case {0}

         % Find non-zero elements (1s) for the diagonal
         [r, c] = find(F(:, i:end));

         % Find non-zero diagonal element candidates
         rowIndex = find(r == i);

         % Find the first non-zero column
         chosenCol = c(rowIndex(1)) + (i - 1);

      % Create diagonally structured matrix using 'Mincol' strategy
      case {1}

         % Find non-zero elements (1s) for the diagonal
         [r, c] = find(F(:, i:end));
         colWeight = sum(F(:, i:end), 1);

         % Find non-zero diagonal element candidates
         rowIndex = find(r == i);

         % Find the minimum column weight
         [x, ix] = min(colWeight(c(rowIndex)));
         % Add offset to the chosen row index to match the dimension of
the...
         % original matrix F
         chosenCol = c(rowIndex(ix)) + (i - 1);

      % Create diagonally structured matrix using 'Minprod' strategy
      case {2}

         % Find non-zero elements (1s) for the diagonal
         [r, c] = find(F(:, i:end));
         colWeight = sum(F(:, i:end), 1) - 1;
         rowWeight = sum(F(i, :), 2) - 1;

         % Find non-zero diagonal element candidates
         rowIndex = find(r == i);
```

```
         % Find the minimum product
         [x, ix] = min(colWeight(c(rowIndex))*rowWeight);
         % Add offset to the chosen row index to match the dimension of
the...
         % original matrix F
         chosenCol = c(rowIndex(ix)) + (i - 1);

      otherwise
         fprintf('Please select columns re-ordering strategy!\n');

   end % switch

   % Re-ordering columns of both H and F
   tmp1 = F(:, i);
   tmp2 = H(:, i);
   F(:, i) = F(:, chosenCol);
   H(:, i) = H(:, chosenCol);
   F(:, chosenCol) = tmp1;
   H(:, chosenCol) = tmp2;

   % Fill the LU matrices column by column
   L(i:end, i) = F(i:end, i);
   U(1:i, i) = F(1:i, i);

   % There will be no rows operation at the last row
   if i < M

      % Find the later rows with non-zero elements in column i
      [r2, c2] = find(F((i + 1):end, i));
      % Add current row to the later rows which have a 1 in column i
      F((i + r2), :) = mod(F((i + r2), :) + repmat(F(i, :), length(r2),
1), 2);

   end % if

end % for i
c=zeros(M,numSource);
u=zeros(M*2,numSource);
for i=1:numSource

    dSource = round(rand(M, 1));

    % Find B.dsource
    z = mod(H(:, (N - M) + 1:end)*dSource, 2);

    % Parity check vector found by solving sparse LU
    c(:,i) = mod(U\(L\z), 2);
    u(:,i) = [c(:,i);dSource];
    % Return the rearrange H
    newH = H;
end
%fprintf('Message encoded.\n');
save dSourcePEGir252x504_0.mat c u newH
```

- Script de transmissão e decodificação:

```
% general script for non-systematic LDPC simulation

% Things to Remember - Steps to follow ALWAYS !!!!!!!!
% 0. This file is good for non-systematic H only (errors calculated for
all n)
% 1. Load H
% 2. Define SNR Range
% 3. Set Maximum number of iterations
% 4. Set Maximum number of codeword-errors for which to run simulation
% 5. Select decoder - Matlab or C-based



clear all
close all
clc
tic

load dSourcePEGreg252x504.mat
H=newH;
dSource = u;

ind=find(H==1);
[r,c]=ind2sub(size(H),ind);
[rows,cols]=size(H);
h=sparse(H);                          % for use with Matlab-based LDPC De-
coder
n=cols;
k=n-rows;


% Find
% 1: maximum check degree
% 2: column indeces in each row which contain '1'
% 3: maximum variable degree
% 4: find column indeces in each row which contain '1'
[max_check_degree,check_node_ones,BIGVALUE_COLS,max_variable_degree,vari-
able_node_ones,BIGVALUE_ROWS]=one_finder(H);



rand('seed',584);
randn('seed',843);

dB=[0:4];                             % range of SNR values in dB
SNRpbit=10.^(dB/10);                  % Eb/No conversion from dB to decimal
No_uncoded=1./SNRpbit;                % since Eb=1
R=k/n;                                % code rate
No=No_uncoded./R;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
max_iter=5;                    %maximum number of decoder iterations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
maximum_blockerror=30;          % maximum blockerrors per SNR point
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
max_errors = 500;

blockerrors=0;
biterrors=0;
block=0;
FER=zeros(1,length(dB));        % array for Frame Error Rate
BER=zeros(1,length(dB));        % array for Channel Error Rate
block_array=zeros(1,length(dB));

for z=1:length(SNRpbit), % loop for testing over range of SNR values

    biterrors=0;
    blockerrors=0;
    block=0;
    i=1;
    while(blockerrors<maximum_blockerror)  %while loop
    %while(biterrors<max_errors)
        maximum_blockerror-blockerrors
        %max_errors-biterrors
        %%%%%%%%%%%%%%% u is the codeword to be transmitted %%%%%%%%%%%%%
%%%%%%
        %u=zeros(1,cols);  %%all zero codeword
        %u = round(rand(cols, 1))';

        %[max_check_degree,check_node_ones,BIGVALUE_COLS,max_variable_de-
gree,variable_node_ones,BIGVALUE_ROWS]=one_finder(H);

        tx_waveform=bpsk(dSource(:,i));

        sigma=sqrt(No(z)/2);
        rx_waveform=tx_waveform + sigma*randn(1,length(tx_waveform));

        gamma_n=(4/No(z))*rx_waveform;
        vhat=decode_ldpc_new(max_iter,gamma_n,check_node_ones,max_check_d
egree,BIGVALUE_COLS-
1,variable_node_ones,max_variable_degree,BIGVALUE_ROWS-1,rows,cols);
        %vhat=decodeLogDomain(rx_waveform', H, gamma_n', max_iter);

        uhat=vhat;

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        Errors=zeros(1,length(vhat));
        Errors(find(dSource(:,i)'~=vhat))=1;
        if i==size(dSource,2)
             i=1;
        else
             i=i+1;
        end
        if sum(Errors)~=0
             blockerrors=blockerrors+1;
```

```
        end

        biterrors=biterrors+sum(Errors);
        block=block+1;

   end     %while loop

   block_array(z)=block; %counting blocks per SNR point
   BER(z)=biterrors/(block*n);
   FER(z)=blockerrors/block;
   fprintf(1,'\n\n Simulation finished for SNR: %d \n',dB(z))

%     save results.mat dB BER FER block_array

end      % loop for testing over range of SNR values


    save resultsPEGir252x504_5_30b.mat BER FER dB


toc
```