UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
ESPECIALIZAÇÃO EM TELEINFORMÁTICA E REDES DE COMPUTADORES

WILLIAN DANIEL DE MATTOS

CÓDIGOS DE VERIFICAÇÃO DE ERROS DE PARIDADE DE BAIXA DENSIDADE (*LDPC*)

MONOGRAFIA DE ESPECIALIZAÇÃO

CURITIBA
2012

WILLIAN DANIEL DE MATTOS

CÓDIGOS DE VERIFICAÇÃO DE ERROS DE PARIDADE DE BAIXA DENSIDADE (*LDPC*)

Monografia de especialização apresentada ao Programa de Pós-Graduação da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de “Especialista em Teleinformática e Redes de Computadores”- Área de Concentração: Telemática.

Orientador: Prof. Dr. Walter Godoy Júnior.

CURITIBA

2012

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
ESPECIALIZAÇÃO EM TELEINFORMÁTICA E REDES DE COMPUTADORES

WILLIAN DANIEL DE MATTOS

CÓDIGOS DE VERIFICAÇÃO DE ERROS DE PARIDADE DE BAIXA DENSIDADE (*LDPC*)

Monografia aprovada como requisito parcial para obtenção do título de Especialista em Teleinformática e Redes de Computadores, da Universidade Tecnológica Federal do Paraná, pela banca formada pelos seguintes professores: NOTA: DEZ INTEIROS (10,0)

Banca Examinadora:

Prof. Dr. Walter Godoy Júnior - Orientador

Prof. MSc. Antonio Gortan

CURITIBA, 26 de JANEIRO de 2012

RESUMO

Esta monografia trata de abordar uma classe de códigos controladores de erros conhecidos como LDPC (*Low Density Parity Check Codes)* ou códigos de verificação de paridade de baixa densidade. Estes códigos foram desenvolvidos na década de 70 e apresentam um ótimo desempenho em canais ruidosos. Diferentemente de outras classes de códigos, eles têm um algoritmo de codificação e decodificação extremamente rápido o que os tornam perfeitos para aplicação prática em sistemas de telecomunicações. O nome veio da característica da matriz de verificação de paridade de possuir uma baixa densidade de elementos não nulos. Para que seja possível discutir estes códigos, alguns fundamentos sobre sistemas de comunicação digital e códigos de bloco lineares são tratados nos capítulos iniciais. Os códigos LDPC serão definidos por meio de sua representação matricial e utilizando grafos bipartidos, que são uma representação gráfica da matriz de verificação de paridade e que trouxeram uma maior facilidade na compreensão do processo de decodificação iterativa. Além disso, o processo de codificação e decodificação utilizando diferentes algoritmos será discutido, nos quais diferentes métodos visam sempre atingir uma taxa que se aproxime o máximo possível da capacidade do canal estabelecida por Shannon.

ABSTRACT

This monograph presents an ensemble of error control codes known as LDPC (Low Density Parity Check Codes). These codes were developed in the 70's decade and have a great performance in a channel with a big amount of noise. Unlike other classes of codes, they are equipped with a very fast encoding and decoding algorithms, which makes them perfect for practical applications in telecommunication systems. The name came from the behavior of the parity check matrix, which has a low density of non-null elements. In order to allow the discussion of theses codes, some elements about digital communications systems e linear block codes are presented in the initial chapters. The LDPC codes will be defined in terms of the matrix representation and via bipartite graphs which are a graphical representation of the parity check matrix which make easier the comprehension of the iterative decoding process. Furthermore, the coding and decoding algorithms using different methods will be discussed, which the approach the capacity of the channel defined by Shannon is the main goal.

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO

1.1 DELIMITAÇÃO DO TEMA

Em um sistema de telecomunicações, um canal de comunicação nunca estará livre de ruídos e interferências no sinal contendo a informação. Na prática, um dos métodos utilizados para reduzir os erros produzidos por estes efeitos é utilizar um sistema codificação de canal. Isto proporciona uma maior confiabilidade e desempenho na comunicação no que diz respeito à taxa de erros de bit na recepção dos dados. Diversos métodos de codificação de canal foram desenvolvidos, e dependem basicamente das características do canal onde a informação será transmitida. As aplicações destes códigos controladores de erro são as mais diversas, tais como: sistemas de comunicação via satélite, redes de computadores e sistemas de radio fusão (*broadcasting),* como a TV e rádio digital.

Os códigos que serão objetos de estudo são conhecidos como Códigos de Verificação de Paridade de Baixa Densidade (*Low Density Parity Check - LDPC)* e constituem uma classe de códigos de bloco lineares que apresentam um ótimo desempenho em canais ruidosos.

Estes códigos foram originalmente criados por Robert Gallager [1] em 1962 em sua tese de PhD. No entanto, estes códigos ficaram esquecidos por décadas devido à inviabilidade de implementação computacional de seu algoritmo de decodificação. Cerca de trinta anos mais tarde, estes códigos foram “reinventados” por Mackay e Neal [2], que demonstraram a possibilidade de se alcançar uma taxa de transmissão codificada próxima ao limite estabelecido por Shannon [3]. Em 1981, R.M Tanner [4] generalizou o trabalho proposto por Gallager, e apresentou uma representação gráfica do LDPC através de grafos bipartidos. Na década de 90, em consequência dos trabalhos de Mackay [5], Neal [6] e Luby [7] os códigos LDPC sofreram um considerável avanço. Atualmente, estes códigos têm sido objetos de novas pesquisas que visam a sua aplicação em sistemas de comunicação modernos.

1.2 OBJETIVOS

O desenvolvimento da monografia visa principalmente obter um conhecimento geral sobre os Códigos de Verificação de Paridade de Baixa Densidade (LDPC). Com a estruturação do conhecimento de maneira sistemática e objetiva, será possível entender e conceituar o problema de maneira precisa e bem delimitada. O estudo terá um caráter de pesquisa, no entanto, será feita uma análise de uma simulação computacional em Matlab, que visa comparar dois algoritmos decodificação. Além disso, a avaliação do desempenho e a possibilidade de otimização dos códigos LDPC serão baseados nos trabalhos previamente publicados, citados nas referências bibliográficas deste documento. Ao final do estudo, além de obter-se o entendimento conceitual e prático do assunto, será possível avaliar a necessidade de realização de novas pesquisas relacionadas ao tema, no que diz respeito à utilização dos códigos LDPC nos sistemas de comunicação digital modernos, tais como a TV e a Rádio digital.

1.3 JUSTIFICATIVA

Os códigos LDPC são um dos temas mais tratados hoje em dia no que diz respeito à teoria de codificação. Diferentemente de outras classes de códigos, eles possuem um algoritmo extremamente rápido de codificação/decodificação. Além disso, o fato de novas ferramentas analíticas e combinatórias permitirem a recuperação da informação em um canal com altas taxas de ruído, torna os códigos LDPC uma alternativa perfeita para aplicações práticas. A principal justificativa para a realização de uma monografia sobre este tema é a necessidade de pesquisas na área da teoria da informação relacionadas à codificação de canal. Por meio do desenvolvimento contínuo destes códigos, é possível aumentar consideravelmente o desempenho de um sistema de comunicação como um todo. Isto reduz custo do sistema e permite a viabilidade de implantação de novas tecnologias para o oferecimento de serviços confiáveis.

CAPÍTULO 2

FUNDAMENTOS DOS SISTEMAS DE COMUNICAÇÃO DIGITAIS

Este capítulo tem o objetivo de apresentar uma visão geral dos sistemas de comunicação digitais. Além disso, visa demonstrar qual é o papel principal dos códigos controladores de erro no projeto destes sistemas. O capítulo se inicia com uma breve introdução de como os trabalhos de Shannon e Hamming contribuíram para o desenvolvimento da teoria da informação e dos códigos controladores de erro. Além disso, apresenta conceitos básicos de um modelo simplificado de um sistema de comunicação digital.

2.1 SHANNON E A TEORIA DA INFORMAÇÃO

Em 1948 Claude E. Shanon publicou um artigo que demarcou o início da teoria da informação [3]. Em seu trabalho, ele formalizou o conceito de informação e estabeleceu os limites para a máxima quantidade de informação de maneira confiável que pode ser transmitida por meio de canais não confiáveis. Dado um canal de comunicação, Shannon provou que existe um número chamado capacidade do canal (C), tal que transmissões confiáveis não podem ter taxas (R) que superem esta capacidade, mas que, no entanto, podem atingir um limite muito próximo desta.

Shannon introduziu em seu artigo uma métrica pela qual a informação pôde ser quantificada. Esta métrica permite determinar o número mínimo de símbolos necessários para uma representação livre de erros de uma dada mensagem. Uma mensagem maior contendo a mesma informação tem a característica de possuir símbolos *redundantes.*

Com isto, é possível definir dois tipos de codificação:

- **Codificação de Fonte**: Códigos de fonte são utilizados para remover a redundância não controlada que naturalmente ocorre em uma sequência de dados. A codificação de fonte reduz a taxa de transmissão efetiva requerida na transmissão dos dados.

- **Codificação de Canal:** Para a codificação de canal utilizam-se códigos controladores de erro. Estes códigos são empregados para aumentar à imunidade ao ruído. Isto é possível por meio da inserção de bits de redundância de maneira controlada e criteriosa, permitindo assim, que o receptor possa identificar e/ou corrigir erros.

No mesmo trabalho Shannon introduziu o conceito de códigos como um conjunto de vetores criados para transmissão e representou matematicamente as operações de codificação e decodificação.

2.2 HAMMING E OS CÓDIGOS DE CANAL

Hamming teve como foco a construção de códigos capazes de detectar e corrigir erros. Em virtude de seu trabalho, ele é considerado por muitos como o criador dos códigos controladores de erro. O artigo intitulado *"Error Detecting and Error Correcting Codes"* [8] foi publicado em 1950 e definiu a base para o desenvolvimento deste assunto. O artigo introduziu uma coletânea de códigos que são hoje conhecidos como códigos de Hamming.

Uma das características básicas dos códigos de canal em um sistema de comunicação é a inserção de bits de redundância na mensagem a ser transmitida. Estes bits são inseridos de maneira sistemática, para que seja possível a detecção e/ou correção de erros no momento da recepção dos dados.

Códigos detectores de erro são mais facilmente implementados devido a sua simplicidade, os quais solicitam a retransmissão de dados ao transmissor caso um erro seja encontrado. Os códigos corretores de erro são utilizados para aumentar a

confiabilidade de sistemas de fluxo constante em que a retransmissão da informação se torna inviável, como é o caso dos sistemas de comunicação sem fio.

2.3 MODELO DE UM SISTEMA DE COMUNICAÇÃO DIGITAL

A figura 2.1 ilustra um modelo típico de um sistema de comunicação digital. Para que uma comunicação efetiva seja possível, é necessário transmitir a informação de um local (ponto A) para outro (ponto B). Neste caso, os pontos A e B são definidos tecnicamente como “fonte” e “receptor”. O sistema é considerado “digital”, pois usa uma sequência finita de símbolos que representam a informação.

A fonte pode ser considerada como qualquer dispositivo capaz de gerar informação. Em geral a fonte gera uma sequência de símbolos a uma média de $\boldsymbol{R_S}$ por segundo, que são então repassados ao codificador de fonte. O codificador de fonte tem como objetivo reduzir a quantidade de bits transmitidos levando-se em conta a probabilidade de ocorrência de símbolos gerados, isto é, eles são utilizados para remover a redundância que normalmente ocorre em uma sequência de símbolos. Os códigos de fonte mais comuns utilizam os algoritmos de Huffman [22].

Após a codificação de fonte, a informação passa pelo bloco dos códigos controladores de erro (também chamada de codificação de canal). Após isto, a informação é enviada ao modulador que então converte os símbolos de informação em sinais, que podem ser eficientemente transmitidos através do canal de comunicação.

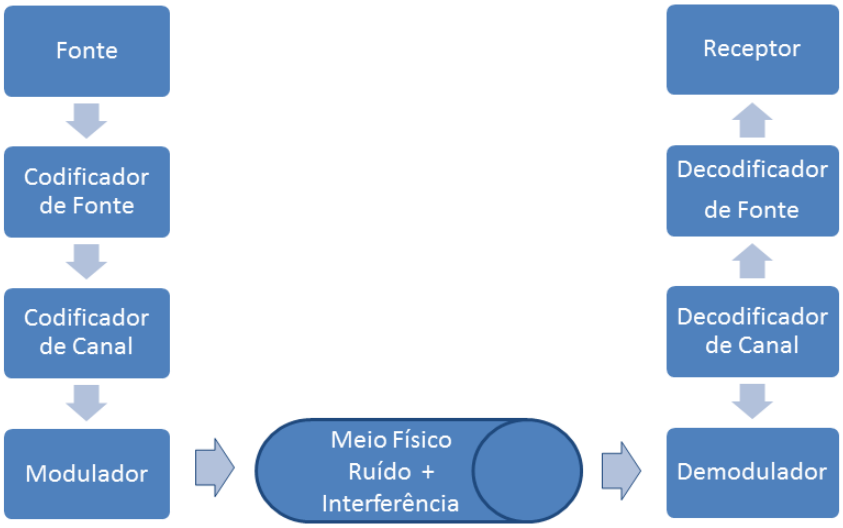

Figura 2.1 - Diagrama de blocos simplificado de um sistema de comunicação digital

A escolha criteriosa da forma de modulação reduz de maneira considerável a potência e a largura de banda requerida para a transmissão. Após o processo de modulação o sinal é então transmitido através de um meio físico, que nunca estará imune a efeitos indesejados, tais como o ruído e a interferência. Após a recepção dos dados, o sinal passará pelo processo inverso ao comentado anteriormente, para ser então entregue ao receptor.

CAPÍTULO 3

CÓDIGOS DE BLOCO LINEARES

Os códigos de bloco introduzem a adição controlada de bits de redundância na sequência de dados, permitindo ao receptor ter a habilidade de detectar e corrigir erros causados por ruídos no canal de comunicação. Códigos de bloco lineares são os mais fáceis de serem implementados e consequentemente os que são mais largamente empregados. Uma das formas de representar estes códigos lineares é por meio de matrizes geradoras e matrizes de verificação de paridade.

3.1 CÓDIGOS DE BLOCO CONTROLADORES DE ERRO

Um código de bloco controlador de erro consiste em um conjunto de *M* palavras código ($c_{0,}c_{1,}\dots,c_{m-1,}$). O processo de codificação consiste em quebrar a sequência de bits em blocos e mapear estes blocos em palavras código **c**. O mapeamento é normalmente feito bloco a bloco de forma a permitir que o processo reverso seja realizado e que a informação original possa ser recuperada.

De acordo com a figura 3.1 os blocos de k elementos são conhecidos como números de bits (ou dígitos) de informação. O valor n representa o comprimento do código, isto é, a quantidade de bits do bloco contendo a informação codificada.

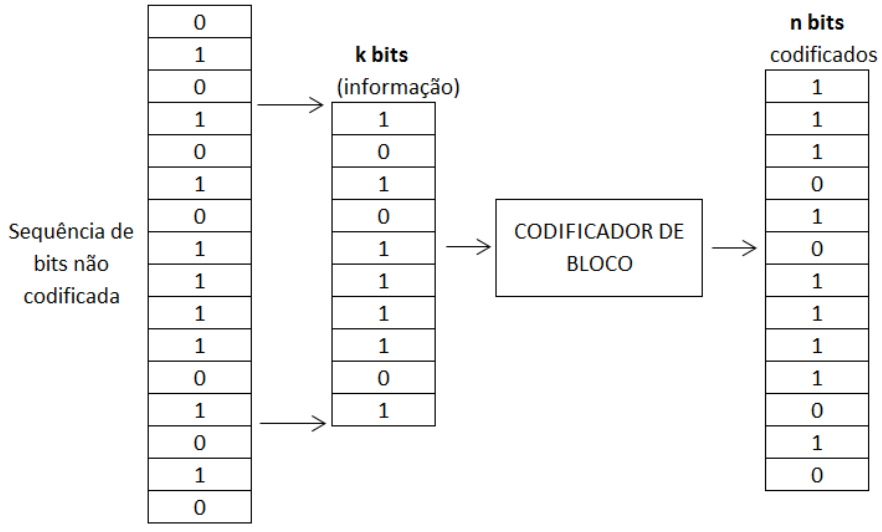

Figura 3.1 - Codificação de Bloco

Dado que $k < n$ o valor $n - k$ representa a quantidade de dígitos de redundância ou dígitos de verificação de paridade.

3.2 PESO DA PALAVRA-CÓDIGO

O peso da palavra código é o número de coordenadas não nulas do bloco. O peso da palavra-código **c** é normalmente escrito como $w(\boldsymbol{c})$.

Exemplo: w(1,0,0,1,1,0,0,1) = 4.

3.3 A DISTÂNCIA DE HAMMING

A distância de Hamming entre dois blocos **v** e **w** é o número de coordenadas em que os dois blocos diferem.

$$d_{Hamming}(\boldsymbol{v}, \boldsymbol{w}) = d(\boldsymbol{v}, \boldsymbol{w}) = |\{i | v_i \neq w_i, i = 0,1, \dots, n-1\}| \qquad (3.1)$$

A distância de Hamming permite uma boa caracterização da capacidade de detecção e correção de um código de bloco em função da mínima distância.

3.4 A MÍNIMA DISTÂNCIA DE UM CÓDIGO DE BLOCO

A mínima distância de um código de bloco C é a mínima distância de Hamming entre todos os pares de palavras-códigos no bloco C. O único padrão de erro que não pode ser detectado é quando o erro introduzido no sistema faz com que uma palavra-código transmitida se torne outra palavra código. Define-se d_{min} como a mínima distância de um código de bloco. Sendo assim, para que um padrão de erro seja indetectável, é necessário que o código da palavra-código seja alterado em no mínimo d_{min} coordenadas.

3.5 CAPACIDADE DE DETECÇÃO DE ERROS

Um código com uma distância mínima d_{min} pode então detectar todos os padrões de erro com um peso menor ou igual a $(d_{min} - 1)$, ou:

$$t_h = d_{min} - 1 \tag{3.2}$$

Em que t_h representa a capacidade de detecção de erros do código. No entanto o código pode detectar um grande número de padrões de erro com peso $w \geq d_{min}$. A equação acima estabelece um limite para o qual o código pode detectar todos os padrões de erro.

3.6 CAPACIDADE DE CORREÇÃO DE ERROS

Um código com uma distância mínima d_{min} pode então corrigir todos os padrões de erro com um peso menor ou igual a $(\left[\frac{d_{min}-1}{2}\right] + 1)$ ou mais, dependendo do tipo de código.

3.7 CÓDIGOS DE BLOCOS LINEARES

Esta seção tratará de algumas propriedades dos códigos de blocos lineares. A estrutura destes códigos os tornam particularmente fáceis de serem implementados e analisados. Uma notação é utilizada para fazer referência aos códigos lineares. Um código linear de comprimento *n* e dimensão *k* é chamado de código (n, k).

Os códigos de bloco lineares possuem algumas propriedades interessantes:

1) Uma combinação linear de duas palavras-código resulta em uma terceira palavra código. Uma consequência disto é que todos os códigos lineares possuem a palavra-código zero.

2) A distância mínima de um código linear é igual ao menor peso de uma palavra código que não seja a palavra-código zero. Consequentemente a determinação da distância mínima, da capacidade de correção e detecção de erros pode ser realizada.

3.8 MATRIZ GERADORA G

Representando $\{g_0, g_1, \dots, g_{k-1}\}$ como a base para palavras-código (n, k) de um código **C**. A matriz **G** é definida como:

$$\mathbf{G}=\begin{bmatrix} g_0 \\ \vdots \\ g_{k-1} \end{bmatrix} = \begin{bmatrix} g_{0,0} & \cdots & g_{0,n-1} \\ \vdots & \ddots & \vdots \\ g_{k-1,0} & \cdots & g_{k-1,n-1} \end{bmatrix} \tag{3.3}$$

A matriz acima é conhecida como **matriz geradora** para um código **C**. Ela é usada para blocos de dados de *k* símbolos de acordo com a equação abaixo. Sendo que $\boldsymbol{m} = (m_0, m_1, \dots, m_{k-1})$ é o conjunto de bits não codificados.

$$\boldsymbol{m}.\boldsymbol{G} = (m_0, m_1, \dots, m_{k-1}) \begin{bmatrix} g_0 \\ \vdots \\ g_{k-1} \end{bmatrix} = m_0 g_0 + m_1 g_1 + \cdots + m_{k-1} g_{k-1} = \boldsymbol{c} \tag{3.4}$$

O vetor **c** é o resultado da codificação da mensagem (palavra-código) com base na multiplicação da matriz da mensagem não codificada **m** pela matriz geradora **G**.

3.9 A VERIFICAÇÃO DE PARIDADE

Uma forma simplificada de se detectar erros consiste em adicionar um bit no final de cada palavra não codificada. Este bit extra é chamando de *bit de paridade* e é escolhido de forma a fazer com que a quantidade de bits 1s de cada palavra seja um número par. Considere então, o caso em que existem apenas 16 possíveis mensagens a serem transmitidas. Estas 16 mensagens podem ser codificadas em 4 bits (2^4). Adicionando um bit extra, a quantidade de palavras possíveis será então 32 (2^5) das quais somente 16 palavras-código contêm informação.

Sendo assim, o bit de verificação de paridade introduziu redundância. Se a quantidade de bits 1's dos primeiros quatro dígitos for ímpar, será então adicionado o bit 1. Então 1011 passará a ser 1011**1**. Se a quantidade de bits 1's dos primeiros quatro dígitos for par, será então adicionado o bit 0. Então 1010 passará a ser 1010**0.**

O bit de verificação de paridade é universalmente utilizado em computadores para detecção de erros. Este bit de verificação será efetivo se a probabilidade de erro em um bit é baixa o suficiente tal que, possa ser ignorada a possibilidade de erro de mais de um bit em uma única palavra-código. Se um erro em um único bit ocorrer um bit 0 será alterado para 1 ou um bit 1 alterado para 0 e a palavra recebida terá uma quantidade ímpar de bits 1's. Consequentemente será identificada a ocorrência de um erro. No entanto não será possível determinar qual é o bit recebido que foi alterado durante a transmissão.

3.10 A MATRIZ DE VERIFICAÇÃO PARIDADE H

A matriz de verificação de paridade **H** é utilizada para validar se a palavra recebida é uma das palavras-código. Sendo assim,

$$\mathbf{H}=\begin{bmatrix} h_0 \\ \vdots \\ h_{n-k-1} \end{bmatrix} = \begin{bmatrix} h_{0,0} & \cdots & h_{0,n-1} \\ \vdots & \ddots & \vdots \\ h_{n-k-1,0} & \cdots & h_{n-k-1,n-1} \end{bmatrix} \quad (3.5)$$

O vetor **c** é uma palavra código somente se $\boldsymbol{c}.\boldsymbol{H^T} = \mathbf{0}$. Sendo assim também é possível definir que: $\boldsymbol{G}.\boldsymbol{H^T} = \mathbf{0}$**.**

3.11 CÓDIGOS DE VERIFICAÇÃO DE PARIDADE

Os códigos de verificação de paridade usam a soma linear dos bits de informação chamados *símbolos de paridade* ou *bits de paridade* para detecção de erro ou correção. Um código de verificação de paridade simples (*single-parity-check code*) é construído por meio da adição de um código de paridade simples em um bloco de bits de dados. O bit de paridade terá o valor de um ou zero conforme necessário, para garantir que a soma de todos os bits da palavra código seja um resultado par (ou impar). A operação de soma é realizada utilizando aritmética módulo 2 (lógica do ou-exclusivo). Se a paridade é adicionada de maneira a se obter um resultado ímpar o método é chamado de paridade ímpar. Caso o bit seja adicionado de maneira a se obter um resultado par o método será chamado de paridade par.

No receptor o processo de decodificação consiste em verificar se a soma módulo-2 da palavra código irá resultar em um resultado igual a zero (paridade par). Se o resultado encontrado for um ao invés de zero, isto significa que a palavra código contêm erros. A taxa do código pode ser expressa como $k\ /\ (k+1)$. Neste caso o decodificador não pode corrigir automaticamente um bit recebido com erro. Ele pode somente detectar a presença de um número ímpar de erros de bits. Assumindo que todos os bits de erro ocorrem de maneira independente, é possível definir que a probabilidade de ocorrem j erros em um bloco de n bits é igual a:

$$P\ (j,n) = \binom{n}{j} p^{j}(1-p)^{(n-1)} \qquad (3.6)$$

em que p é a probabilidade de que a palavra código é recebida com erro e onde:

$$\binom{n}{j} = \frac{n!}{j!(n-j)!} \qquad (3.7)$$

é o número de diferentes maneiras com j bits de n podem estar em erro.

Assim, para um código de verificação de paridade simples com detecção de erro, a probabilidade de um erro indetectável P_{nd} para um bloco de n bits é definida da seguinte maneira:

$$P_{nd} = \sum_{j=1}^{\substack{\frac{n}{2}\,(para\ n\ par) \\ \frac{n-1}{2}\,(para\ n\ ímpar)}} \binom{n}{2j} p^{2j}\,(1-p)^{n-2j} \tag{3.8}$$

3.12 TESTE DA SÍNDROME

Sendo $r = r_1, r_2, \ldots, r_n$ o vetor recebido como resultado da transmissão de $U = u_1, u_2, \ldots, u_n$ é possível definir *r* como:

$$r = U + e \tag{3.9}$$

onde $e = e_1, e_2, \ldots, e_n$ é um vetor de erro ou um padrão de erro introduzido pelo canal. Existe um total de $2^n - 1$ potenciais padrões de erro (diferentes de zero) no espaço de 2^n n-tuplas. A *síndrome* de *r* é definida como:

$$S = r.H^T \tag{3.10}$$

A síndrome é o resultado da verificação de paridade realizada em *r* para determinar se **r** é um membro válido do conjunto de palavras código. Se, de fato *r* for um membro o teste da síndrome têm um valor **zero**. Se *r* contem erros detectáveis, o teste da síndrome têm um valor diferente de zero. Se *r* contem erros corrigíveis, a síndrome tem alguns valores diferentes de zero que podem ser reconhecidos por possuírem um padrão de erro particular. O decodificador tomará então ações para localizar os erros e corrigi-los ou solicitar a retransmissão.

3.13 CANAL BINÁRIO SIMÉTRICO

O canal binário simétrico (*BSC - Binary Symmetric Channel*) é um canal discreto sem memória que possui um alfabeto binário de entrada e saída e transições de probabilidade simétricas. Isto pode ser descrito pelas probabilidades condicionais:

$$P(0|1) = P(1|0) = p \tag{3.11}$$

$$P(1|1) = P(0|0) = 1 - p$$

como ilustrado na figura 3.2. A probabilidade que um símbolo de saída seja diferente do símbolo de entrada é p, e a probabilidade de que o símbolo de saída seja idêntico ao símbolo de entrada é $(1-p)$. O canal BSC é um exemplo de um canal de decisão abrupta (*hard-decision channel*) o que significa que mesmo que valores contínuos de sinal sejam recebidos pelo demodulador, o BSC somente permite decisões inflexíveis tais que cada símbolo de saída do demodulador consiste em um dos dois valores binários.

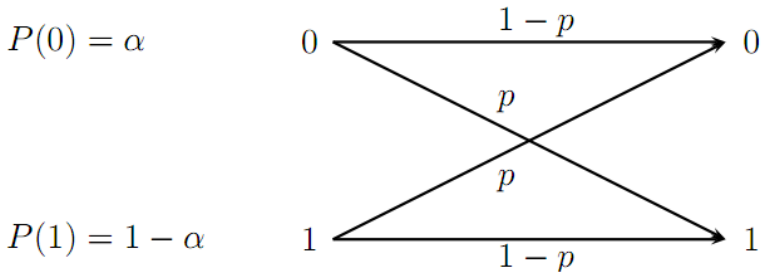

Figura 3.2 - Canal Binário Simétrico (BSC)

CAPÍTULO 4

CÓDIGOS DE VERIFICAÇÃO DE PARIDADE DE BAIXA DENSIDADE

4.1 INTRODUÇÃO

Os Códigos de Verificação de Paridade de Baixa Densidade (*Low Density Parity Check Codes - LDPC)* são considerados uma classe de códigos de bloco linear que oferecem um desempenho surpreendentemente próximo ao limite de Shannon no canal AWGN [2]. O nome veio da característica de possuírem uma *Matriz de Verificação de Paridade H* esparsa, isto é, a matriz H possui apenas alguns 1's em comparação à quantidade de 0's. Esta característica advém do fato de estes códigos possuírem uma grande distância mínima e consequentemente uma baixa probabilidade de erro.

Os códigos LDPC foram introduzidos por Gallager na década de 70, mas, devido ao esforço necessário para a implementação computacional do seu codificador/decodificador e à introdução dos códigos *Reed-Solomon* [9], eles foram ignorados por décadas. Seguidos pela descoberta dos *Turbo Codes* [10] os códigos LDPC foram redescobertos por meio do trabalho de Mackay e Neal [2], e voltaram ser pesquisados pela comunidade científica.

Neste capítulo são discutidos os principais conceitos dos códigos LDPC. Inicialmente será demonstrada a representação regular dos códigos utilizando matrizes e sua representação por meio de grafos bipartidos. A representação gráfica dos códigos LDPC leva ao conceito dos códigos LDPC irregulares introduzidos por Tanner [11] em 1981. Além disso, serão abordados vários métodos de construção dos códigos LDPC bem como os processos de codificação e decodificação.

4.2 REPRESENTAÇÃO MATRICIAL

Um código de bloco linear ***C*** com uma taxa $R = k/n$ pode ser definido em termos de uma matriz de verificação de paridade $\boldsymbol{H}_{(n-k)\,x\,n} = [h_1, h_2, \dots, h_n]$. Cada elemento h_{ij} da matriz H é um elemento de um campo finito do campo de *Galois* [12] GF(*p*). O código **C** é um conjunto de todos os valores **x** que se encontram no espaço nulo de **H**, tal que, **H**.**x** = 0. Dada uma matriz de paridade **H,** é possível encontrar a matriz $\boldsymbol{G_{k\,x\,n}}$ tal que $\boldsymbol{G.H^T = 0.}$ A matriz geradora pode ser utilizada como um codificador de acordo com $\boldsymbol{x^T = u^T.G}$.

Na sua forma mais simples, um código LDPC é um código de bloco linear com a matriz de paridade esparsa, isto é, que possui uma baixa densidade de elementos não nulos. Em [1] Gallager propôs a construção de códigos LDPC por inserção randômica de 0's e 1's na matriz $H_{m\,x\,n}$. Ele definiu que cada coluna da matriz **H** deve ter o mesmo número d_v de 1's e que cada linha de **H** deve ter o mesmo número d_c de 1's. Por exemplo, a matriz de paridade **H** *m*=15 x *n* =20 mostrada na figura 4.1 tem $d_v = 3$ e $d_c = 4$ e define um código LDPC com comprimento $n = 20$.

$$\boldsymbol{H_{1(m\,x\,n)}} = \begin{bmatrix}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

Figura 4.1 - Matriz LDPC H_1

O percentual de 1's na matriz de paridade H de um código LDPC regular é igual a $\frac{d_c}{n}$ e tende a se aproximar de zero na medida em que o bloco fica maior. Em virtude disto é que o código LDPC é definido como de **baixa densidade**.

É necessário então determinar a taxa do código definido por H_1. Levando-se em conta que a matriz é randomicamente construída, não existe nenhuma garantia de que as linhas são linearmente independentes, isto é, de que nenhum dos elementos da matriz seja uma combinação linear dos outros. O maior conjunto de vetores linearmente independentes de uma matriz é conhecido como posto (r*ank)*. Neste caso o *posto* da matriz H_1 é 13 < m = 15, e esta matriz de verificação paridade define um código com uma taxa R = 7/20. Em geral, esta matriz construída randomicamente não será uma matriz *full rank*, isto é, com todos os seus elementos linearmente independentes. Além disso, nesta matriz $m \neq n - k$. É possível com certeza eliminar a dependência linear das linhas para encontrar uma nova matriz $(n-k)x\ n$, no entanto, esta matriz não será mais regular. Para códigos LDPC com um grande valor de *n*, é conveniente manter a matriz de verificação de paridade original, mesmo que não possua elementos totalmente linearmente independentes. É oportuno se referir a $1 - \frac{m}{n} = 1 - \frac{d_v}{d_c}$ como a taxa projetada do código.

Tendo definido a código LDPC regular (d_v, d_c), o próximo passo é construir um instância particular do código. Para fazer isto, é necessária a escolha de um $d_v, d_c, n\ e\ k$, e que necessariamente $md_c = nd_v$, para que seja um código regular. Além disso d_v<d_c para que a taxa R seja menor ou igual a 1. Assumindo que o comprimento *n* do bloco e a taxa R sejam definidos pela aplicação, é ainda necessário escolher valores apropriados de d_v e d_c. Em [1] Gallager mostrou que a distância mínima de um código LDPC típico aumenta linearmente com *n*, tal que $d_v \geq 3$. Em virtude disto, a maioria dos códigos LDPC regulares são construídos com d_v e $d_c$na ordem de 3 ou 4, levando-se em conta as restrições acima. Para blocos com comprimentos de ordem maior, a inserção aleatória de 1's na matriz **H,** para que cada linha tenha exatamente a mesma quantidade d_c de 1's, necessita de um considerável esforço, mas vários métodos vêm sendo desenvolvidos para atingir este objetivo [2,5].

4.3 REPRESENTAÇÃO POR MEIO DE GRAFOS BIPARTIDOS

Um importante avanço na teoria dos códigos LDPC ocorreu quanto Tanner [11] usou grafos bipartidos para prover uma representação gráfica da matriz de verificação de paridade. (Como consequência, o grafo bipartido de um código LDPC algumas vezes é chamado de *Grafo de Tanner*). Isto facilitou a compreensão do processo de decodificação iterativa.

Um grafo bipartido é um grafo representado por um diagrama com duas regiões distintas. Estas regiões são conectadas por linhas de acordo com o número de elementos 1's da matriz de verificação de paridade **H**. Estas duas regiões são conhecidas como *nós de variáveis* e *nós de verificação*. Existe um nó de variável para cada um dos *n* bits do código e existe um nó de verificação para cada uma das *m* linhas de uma matriz **H**. Uma ligação existe entre um nó de variável e um nó de verificação somente se $h_{ij} = 1$. O grafo bipartido correspondente à matriz de verificação de paridade $\boldsymbol{H_1}$ é mostrado na figura 4.2. No grafo o número de linhas que entram em um determinado nó, é chamado de grau do nó. Sendo assim, um grafo bipartido de um código LDPC (d_v, d_c) contêm *n* nós de variáveis de grau d_v e *m* nós de verificação de grau d_c.

Está claro que uma matriz de verificação de paridade pode ser deduzida de uma grafo bipartido, e, portanto ele pode ser utilizado para definir um código **C**. Com isto, os códigos LDPC podem ser definidos como um conjunto de nós de variáveis, um conjunto de nós de verificação e uma série de ligações. É desta forma que atualmente estes códigos são tratados.

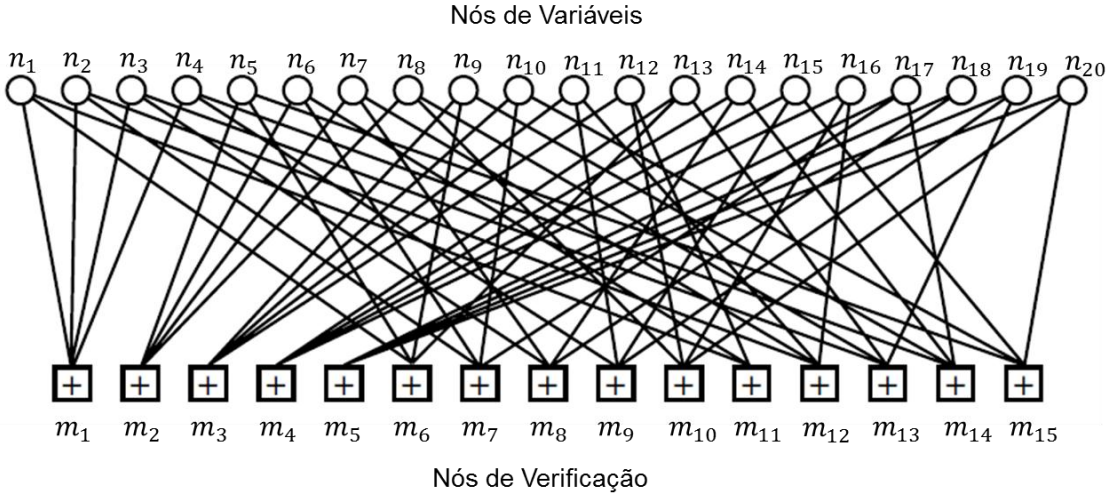

Figura 4.2 - Grafo Bipartido de um código LDPC regular (3,4) para a matriz H_1

Nota-se que o par (d_v, d_c), juntamente com o comprimento *n* do código, define muito mais um conjunto de códigos do que um código específico. Este conjunto pode ser definido por $\boldsymbol{C^n}(d_v, d_c)$. Tendo definido os graus dos nós de verificação e dos nós de variáveis, as ligações do grafo podem ser definidas aleatoriamente.

As primeiras 6 das 15 equações de *Verificação de Paridade* da matriz H_1 são:

$$n_1 + n_2 + n_3 + n_4 = 0 \quad (4.1)$$

$$n_5 + n_6 + n_7 + n_8 = 0 \quad (4.2)$$

$$n_9 + n_{10} + n_{11} + n_{12} = 0 \quad (4.3)$$

$$n_{13} + n_{14} + n_{15} + n_{16} = 0 \quad (4.4)$$

$$n_{17} + n_{18} + n_{19} + n_{20} = 0 \quad (4.4)$$

$$n_1 + n_5 + n_9 + n_{13} = 0 \quad (4.6)$$

...

4.4 O CONCEITO DE *GIRTH*

Um importante conceito dos grafos bipartidos é o do ciclo com comprimento γ e é definido como o percurso fechado formado por γ caminhos. O menor comprimento de todos os ciclos de um grafo bipartido é definido como *girth.* A linhas em negrito da figura 4.3 demostram um *girth* de comprimento 4.

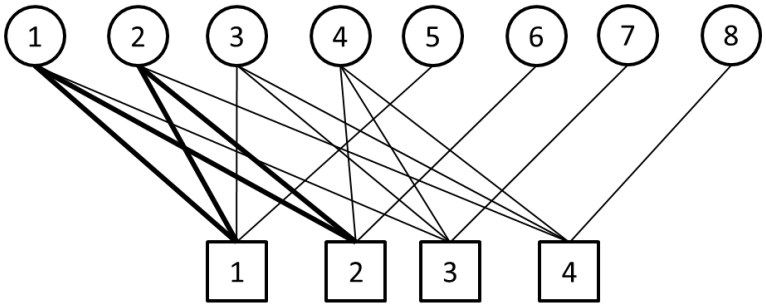

Figura 4.3 - Exemplo de um *girth* de comprimento 4

Para que um algoritmo de decodificação BP (*Belief Propagation*) de um código LDPC tenha um bom desempenho, procura-se sempre um *girth* elevado, isto é, os ciclos curtos devem ser evitados. Sendo assim, durante a construção de um código LDPC para que ele tenha boas propriedades (grande distância mínima e *girth*) algumas regras devem ser seguidas. Qualquer grafo bipartido deve ter um

girth igual a um número par e o seu valor mínimo deve ser 4. Uma das formas de evitar a existência de ciclos de dimensão 4 é garantir que quaisquer duas colunas de uma matriz H contenham no máximo um elemento com o valor 1 na mesma posição.

4.5 CÓDIGOS LDPC REGULARES E IRREGULARES

Um código LDPC é dito regular se d_v é constante para todas as colunas e se $d_c = d_v \,.\left(\frac{n}{m}\right)$ também é constante para todas as linhas. Um exemplo de um código regular foi observado na figura 4.2 em que $d_v = 3$ e $d_c = 4$. É também possível verificar a regularidade de uma matriz de verificação de paridade por meio da representação gráfica. Existe a mesma quantidade de linhas entrando em cada um dos nós de variáveis, bem como este valor também é constante para os nós de verificação. Os códigos LDPC irregulares possuem uma baixa densidade de elementos não nulos, mas, no entanto, o número de 1's nas linhas e colunas da matriz **H** não é constante. Richardsom e Urbanke provaram em [18] que os códigos LDPC irregulares têm um desempenho superior, se comparados com os códigos regulares para blocos longos. No entanto, devido a implementação de hardware dos códigos LDPC regulares ser mais simples em relação aos irregulares, eles são mais largamente aplicados.

4.6 CONSTRUÇÃO DE CÓDIGOS LDPC

Diferentes algoritmos têm sido utilizados para construção de códigos LDPC otimizados. Gallager propôs um método e Mackay propôs outro por meio da construção semi-aleatória de matrizes de verificação de paridade esparsas. Isto pode indicar que a construção de códigos LDPC com bom desempenho não chega a ser um problema. Na realidade por meio do método aleatório, sem deixar de considerar as regras previamente estabelecidas, é alta a probabilidade de se obter um bom código. O problema é quando estes códigos aumentam a complexidade do processo de codificação.

4.7 PROCESSO DE CODIFICAÇÃO

O processo de codificação de códigos LDPC é grosseiramente definido da seguinte maneira: defina alguns nós de variáveis para colocar os bits de mensagem. Em um segundo passo calcule os valores faltantes para os outros nós. Uma solução óbvia para isto será resolver as equações de verificação de paridade. Isto envolverá operações baseadas em toda a matriz de verificação de paridade e sua complexidade será quadrática de acordo com o comprimento do bloco. No entanto, na prática outros métodos mais rebuscados têm sido utilizados para garantir que o processo de codificação se dê de uma maneira mais rápida e fácil.

4.8 ALGORITMOS DE CODIFICAÇÃO

Um algoritmo de codificação para um código linear *k* e comprimento de bloco n é um algoritmo que calcula uma palavra código de *k* bits originais $(x_1, x_2, \dots x_n)$. Para que seja possível comparar estes algoritmos, é importante introduzir o conceito de custo. O custo de um algoritmo é o número de operações matemáticas sobre $\mathbb{F}_2$ que o algoritmo em questão utiliza.

Se a base $g_1, g_2, \dots, g_k$ é conhecida então a codificação pode ser realizada calculando-se $x_1 g_1 + x_2 g_2 + \dots + x_k g_k$. Se um algoritmo direto é utilizado para realizar a codificação, então o número de operações necessárias para isto depende do peso de *Hamming* do vetor base. Se os vetores forem densos (alta densidade de elementos não nulos), então o custo do algoritmo de codificação é proporcional a nk. Para códigos com taxa constante, o custo é proporcional a n^2, o que pode ser demorado para algumas aplicações.

Infelizmente, os códigos LDPC são dados pelo espaço nulo de uma matriz esparsa, mais do que pelo espaço das linhas da mesma matriz. Para um dado código LDPC é altamente improvável que exista uma matriz geradora baseada em vetores esparsos, portanto o algoritmo de codificação direto usa um número de operações que é proporcional a n^2. No entanto, o ideal é projetar algoritmos de codificação nos quais o custo seja proporcional a n.

Existem duas possibilidades para o projeto destes códigos. Uma delas é a consideração das modificações de códigos LDPC que estão equipadas com algoritmos de rápida codificação (*fast encoding).* Outra maneira é tentar encontrar algoritmos de codificação mais rápidos para códigos LDPC tradicionais. Cada uma das possibilidades possuem prós e contras, que serão discutidos mais adiante.

Uma forma simples de se obter códigos de grafos esparsos com rápida codificação é por meio da modificação da construção do código LDPC de tal maneira que os nós de verificação tenham valores, e o valor de cada nó de verificação seja a soma dos valores de seus nós de variáveis adjacentes. Sendo assim, uma codificação é eficiente se o grafo for esparso. As palavras-código deste algoritmo consistem nos valores dos nós de mensagem acrescentados pelos valores dos nós de verificação. Este método de construção leva a um codificador linear no tempo, mas seu problema principal é o processo de decodificação.

Outra classe de códigos obtidos de grafos esparsos e equipados com codificadores rápidos são os códigos RA (*Repeat-Accumulate)* de Divsalar [27]. A construção destes códigos é similar ao processo discutido acima. No entanto, ao invés de proteger os nós de verificação com outra camada de um grafo esparso, a proteção é feita por meio de um grafo denso e os nós de verificação do primeiro grafo nunca são transmitidos. Grafos densos geralmente não são responsáveis por uma rápida codificação. No entanto, um grafo denso escolhido em um código RA é uma estrutura especial que torna o processamento computacional mais fácil.

4.9 DECODIFICAÇÃO DE CÓDIGOS LDPC

O algoritmo utilizado para decodificar os códigos LDPC foi descoberto de maneira independente por diversas vezes e, em virtude disto, aparece com diferentes nomes. Os nomes mais comuns são BF (*Belief Propagation)*, algoritmo de transferência de mensagem (*message passing algorithm)* e algoritmo de soma a produto (sum-*product algorithm).* A complexidade do algoritmo de decodificação está relacionada à densidade da matriz de paridade e, em consequência disto, durante o desenvolvimento de um código LDPC procura-se a menor densidade possível.

Para que seja possível explicar o processo de decodificação é necessário apresentar dois métodos de decodificação, que variam de acordo com o modelo do canal de transmissão. Quando o conjunto de símbolos na entrada do decodificador é finito, diz-se que o processo de decodificação se dará por decisão abrupta (*"hard decision").* Outro método é a decodificação por decisão suave (*"soft decision"*), que considera a distribuição probabilística dos símbolos recebidos, sem a decisão bit a bit, pois toma a sua decisão baseada na palavra inteira.

Os algoritmos de decisão abrupta são menos complexos e exigem menos processamento computacional. Os algoritmos de decisão suave que são baseados no conceito de transferência de mensagens (que será discutido no item 4.10) apresenta um melhor desempenho em termos de uma menor taxa de erro de bit (BER) é consequentemente são os algoritmos preferidos para aplicações práticas.

4.10 ALGORITMOS DE TRANSFERÊNCIA DE MENSAGEMS

Como já foi comentado, a principal vantagem dos códigos LDPC é que eles podem ser decodificados utilizando algoritmos de decodificação iterativa no qual a complexidade cresce linearmente com o comprimento do bloco do código. Eles são chamados de *algoritmos de transferência de mensagens* (*message passing algorithms)* devido ao fato de que cada ciclo dos algoritmos de mensagens passa dos nós de variáveis para os nós de verificação e retorna novamente para os nós de variáveis. Todos os algoritmos de transferência de mensagens precisam respeitar a seguinte regra (introduzida com os *Turbo Codes): uma mensagem enviada de um nó n ao longo de uma ligação e, não pode depender de nenhuma mensagem previamente recebida na ligação e.*

Antes de determinar o algoritmo, é necessário formular o problema da decodificação. Um código LDPC é definido em termos da matriz de verificação de paridade **H** que é utilizada para realizar a decodificação. Para codificar a sequência de dados *u* é necessário obter a matriz geradora ***G*** tal que $\boldsymbol{G}.\boldsymbol{H}^{\boldsymbol{T}} = \mathbf{0}$. O processo para encontrar a matriz **G** mais adequada pode ser simplificado se a matriz ***H*** for convertida para a sua forma sistemática, tal que $\boldsymbol{H_S} = [\boldsymbol{A}|\boldsymbol{I_{n-k}}]$.

Com isto a matriz geradora **G** pode ser definida como $\boldsymbol{G_S = [I_K|A^T]}$ e a sequência de dados pode ser codificada de acordo com a equação $\boldsymbol{x^T = u^T G_s}$.

Um importante código de transferência de mensagem é o algoritmo de *Belief Propagation (BP).* O algoritmo BP foi apresentado por J. Pearl em [13] é um exemplo que pertence a uma classe ampla de algoritmos como discutido em [14,15,16,17]. Ele é utilizado para diversas aplicações tais como processamento digital de sinais e inteligência artificial. Este algoritmo permite a decodificação suave (*Soft Decoding)* e considera a distribuição probabilística dos símbolos recebidos.

4.11 EVOLUÇÃO DA DENSIDADE

Após a definição do processo de codificação e decodificação dos códigos LDPC via algoritmos de transferência de mensagens, é possível apresentar uma forma de escolher os parâmetros e códigos para atingir um maior desempenho. Richardson e Urbanke [20, 21] usam uma proposta chamada evolução da densidade (*"density evolution")* que compara os diferentes métodos de construção de códigos LDPC regulares e irregulares. A evolução da densidade teve a sua origem nas fórmulas de evolução de probabilidade de erro apresentadas por Gallager e as modernas variações dos códigos SISO (*Soft-Input Soft-Output)* são em geral, uma generalização destas fórmulas.

Para o desenvolvimento de bons códigos, são escolhidos os conjuntos com os melhores limiares para os quais e selecionado um código com o maior comprimento que pode ser acomodado para a implementação. O projeto do código pode consistir na amostragem aleatória de alguns códigos do conjunto e na seleção do melhor deles.

CAPÍTULO 5

SIMULAÇÃO COMPUTACIONAL EM MATLAB

O programa desenvolvido em Matlab, tem como função apresentar uma introdução ao processo de simulação de códigos LDPC em um ambiente computacional baseado em software. O código-fonte está disponível em sua versão original no Anexo A, e foi desenvolvido por Nugroho [23]. O principal objetivo do software é avaliar o desempenho dos códigos LDPC em um canal AWGN no qual o é utilizada uma modulação BPSK. O modelo utilizado para simulação pode ser entendido por meio da figura 5.1, que demonstra cada etapa do processo utilizando um diagrama de blocos:

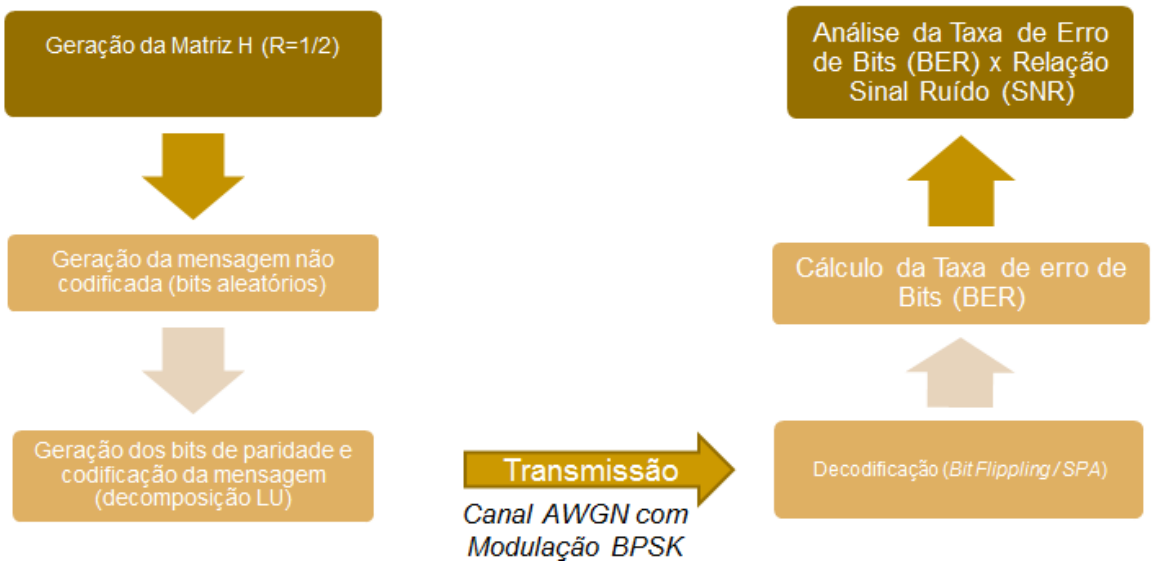

Figura 5.1 - Modelo utilizado para simulação de códigos LDPC em Matlab.

O programa permite realizar a decodificação utilizando quatro algoritmos: Algoritmo de Troca de Bits (*Bit-Flipping*), Algoritmo de Soma e Produto no domínio das probabilidades (*SPA - Sum Product Algoritm*), Algoritmo de Soma e Produto no domínio Logarítmico (*LSPA - Logarithmic Sum-Product Algorithm*), e por último Algoritmo de Soma e Produto Simplificado no domínio Logarítmico (*MS-LSPA Min-Sum-Product Algorithm*).

No entanto, neste capítulo serão apenas analisados dois primeiros algoritmos apresentados anteriormente por meio da representação gráfica da taxa de erro de bits (*BER - Bit Error Rate*) para diferentes valores de relação sinal/ruído (*SNR - Signal Noise Rate*). A seguir, será brevemente explicada cada etapa do processo de simulação realizada pelo software.

5.1 CRIAÇÃO DA MATRIZ LDPC

Os passos para criação da matriz LDPC são os seguintes:

- Selecionar o método para criação da matriz de verificação de paridade esparsa, sendo que o método 0 é o *Evencol*, que considera a mesma quantidade de elementos não nulos em cada coluna da matriz, mas que no entanto gera uma matriz irregular. O método 1 é o *Evenboth* que considera a mesma quantidade de 1s para as linhas e colunas da matriz **H** e consequentemente, gera uma matriz regular.

- Definir a quantidade de linhas (M) e colunas (N) da matriz **H**;

- Caso seja necessário, evitar ciclos de comprimento 4 (*girth* elevado), conforme explicado no item 4.4.

A função **makeLDPC.m** possui cinco parâmetros:

- M: Número de linhas;
- N: Número de colunas;
- *method*: 0:*Evencol*, 1: *Evenboth*;
- *noclycle*: 1= evita ciclos de comprimento elevado;
- *onePerCol*: Quantidade de 1s por linha (ou linhas e colunas, caso o método escolhido seja o *Evenboth*).

A saída da função será uma matriz de verificação de paridade **H** com M linhas e N colunas. A função só tem condições de gerar matrizes com uma taxa $R = 1/2$.

5.2 GERAÇÃO DOS BITS DE VERIFICAÇÃO DE PARIDADE

Os bits de verificação de paridade são gerados utilizando um processo conhecido como decomposição LU esparsa. Este método tem como objetivo reordenar a matriz de verificação de paridade. A função **makeParityChk.m** possui três parâmetros:

- *dSource*: Bits originados da fonte de dados. Neste caso os bits são gerados de forma aleatória simulando uma fonte real.
- *H*: Matriz de verificação de paridade (definida na etapa 5.2).
- *Strategy*: Estratégia utilizada para a decomposição LU. 0:First, 1: *Mincol* e 2:*Minprod*.

A saída da função será:

- *c*: Vetor de bits de verificação de paridade;
- *newH*: Matriz **H** reordenada (que será utilizada para o processo de codificação/decodificação).

5.3 TRANSMISSÃO DOS DADOS

Para a transmissão, será considerado o modelo de um canal AWGN e a modulação utilizada será a BPSK. Para que isto seja possível os bits gerados como 0 e 1 serão mapeados para -1 e +1, simulando uma variação de fase de 180° na representação dos 0s e 1s. Além disso, será adicionado ao sinal transmitido um valor de ruído gerado seguindo a função de distribuição de probabilidades de um canal AWGN.

5.4 ANÁLISE DOS ALGORITMOS DE DECODIFICAÇÃO

A simulação foi realizada utilizando os seguintes algoritmos de decodificação:

- Algoritmo de Troca de Bits (*Bit-Flipping Algorithm)*: Algoritmo proposto por Gallager e que possui baixa complexidade. Ele utiliza o método de decodificação por decisão abrupta (explicado no item 4.9). A idéia principal é negar a menor quantidade de bits da palavra código recebida, de forma que todas a equações da matriz de verificação de paridade **H** sejam satisfeitas. Este algoritmo se torna mais simples quando a matriz **H** possui uma baixa densidade de 1s e quando o grau dos nós de variáveis de verificação do grafo bipartido é baixo. A função **decodeBitFlip.m** decodifica os bits recebidos utilizando este algoritmo.

- Algoritmo de Soma e Produto (*SPA*): São também designados como algoritmos de transferência de mensagens e utilizam o processo de decodificação por decisão suave. Ele leva em conta a distribuição probabilística dos bits recebidos e apresenta o melhor desempenho para os códigos LDPC. Richardson e Urbanke [20] demonstraram ser possível aproximar (a menos de 0,0045 db) da capacidade de um canal AWGN considerando um código de comprimento infinito. A função **decodeProbDomain.m** decodifica os bits recebidos no domínio das probabilidades.

A figura 5.2 é o resultado da simulação utilizando o software tratado neste capítulo e que tem como objetivo comparar os dois códigos anteriores. Observa-se que o desempenho do algoritmo SPA é consideravelmente maior quando comparado ao Algoritmo de Troca de Bits.

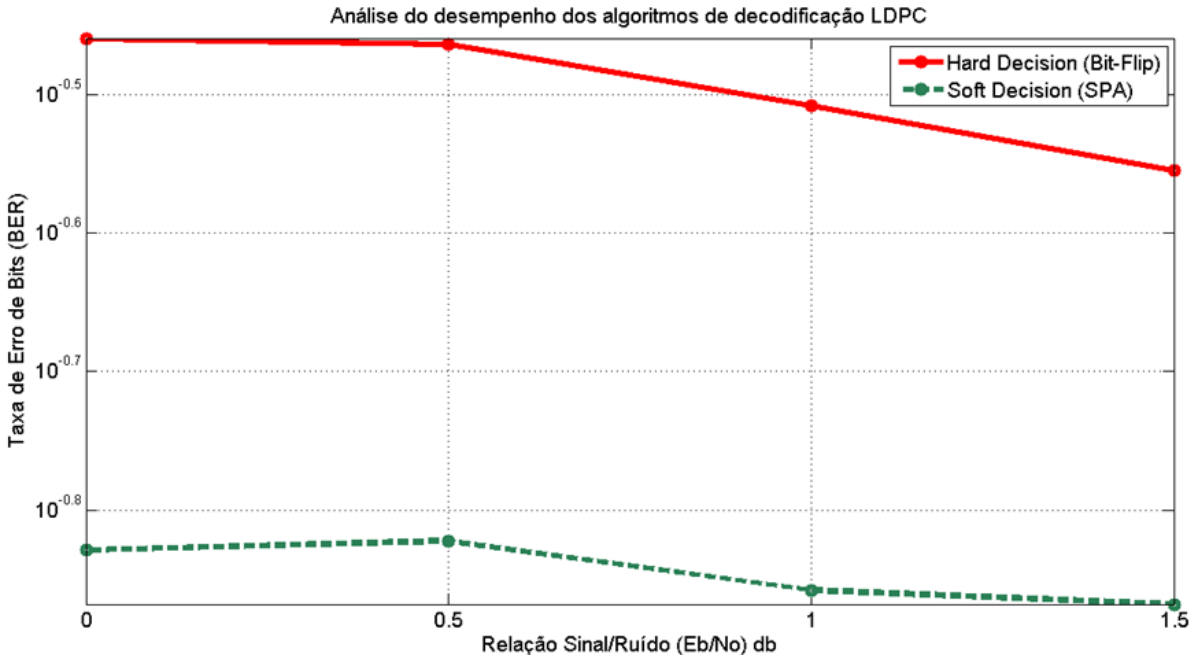

Figura 5.2 - Análise do desempenho dos algoritmos de decodificação LDPC

CAPÍTULO 6

CONCLUSÃO E TRABALHOS FUTUROS

O objetivo principal desta monografia foi obter uma visão geral dos códigos LDPC e de como eles podem ser utilizados para aplicações práticas em sistemas de comunicação digital. Vale ressaltar a importante contribuição de Gallager para a comunidade científica com a introdução destes códigos, mesmo que de maneira teórica. Após a sua criação, estes códigos se desenvolveram consideravelmente ao longo das décadas e são objetos de pesquisas nos sistemas de comunicação modernos. Com esta pesquisa foi possível obter a base necessária para compreender como funciona o processo de codificação/decodificação de canal e qual a sua importância dentro de um sistema de telecomunicações confiável.

Além disso, foi possível compreender como se constroem os códigos LDPC de forma eficiente e como eles podem ser representados, seja por meio de matrizes, ou utilizando grafos bipartidos, o que facilita o processo de compreensão. Alguns algoritmos foram tratados, os quais utilizam diferentes métodos para atingir uma eficiência maior e tornar o processo de decodificação mais rápido.

Um aspecto que poderá ser tratado em trabalhos futuros é a análise detalhada dos algoritmos de decodificação LDPC. Isto é possível por meio de um estudo comparativo entre os diferentes métodos analisando como eles se comportam em ambientes com interferência aditiva do ruído gaussiano e do desvanecimento plano e seletivo em frequência. Os resultados poderão ser avaliados por meio de simulações computacionais e Matlab, tal como foi feito no Capítulo 4, onde será possível analisar o comportamento de cada algoritmo em diferentes condições de canal.

REFERÊNCIAS BIBLIOGRÁFICAS

1. R. G. Gallager, ***"Low-Density Parity-Check Codes"***, MIT Press, Cambridge, MA, 1963.
2. D. J. C. MacKay and R. M. Neal, ***"Near Shannon limit performance of low density parity check codes"*****,** Electronics Letters , vol. 32, August. 1996.
3. C. E. Shannon, ***"A Mathematical Theory of Communication"***, Bell Systems Technical Journal, vol. 27, 1948.
4. R.M Tanner, ***"A recursive approach to low complexity codes"***, IEEE Transactions on Information Theory, September, 1981.
5. D. Mackay, ***"Good Error correcting codes based on very sparse matrices"***, IEEE Transactions on Information Theory, March, 1999.
6. D. J. C. MacKay and R. M. Neal, ***"Good Codes based on Very Sparse Matrices"*****,** Cryptography and Coding, 5th IMA Conference, Colin Boyd (ed.), Lecture Notes in Computer Science, 1995, Springer, Berlim.
7. M. Luby, M. Mitzenmacher, M. Shokrollahi e D. Spielman, ***"Analysis of Low Density Codes and Improved Designs Using Irregular Graphs"*****. Proceedings of** the 30th ACM Symposium on Theory of Computing (STOC), May 1998.
8. RW Hamming, ***"Error Detecting and Error Correcting Codes"***, Bell System Technical Journal 26(2):147-160 Handcock, M.S., 2008. Exponential random graph (ERG or p*) models. Workshop given at Sunbelt XXXVIII Conference 1950.
9. Welch, L. R. (1997), ***"The Original View of Reed–Solomon Codes".***
10. R. J. McEliece, D. J. C. MacKay and J.-F. Cheng. ***"Turbo Decoding as an Instance of Pearl's `Belief Propagation' Algorithm"***. IEEE Journal on Selected Areas in Communication.
11. R.M Tanner, ***"A recursive approach to low complexity codes"***, IEEE Transactions on Information Theory, p.533-547, September, 1981.
12. Lidl, Rudolf; Niederreiter, Harald (1997), ***"Finite Fields"*** (2nd ed.), Cambridge University Press.
13. Pearl, J. ***"Probabilistic reasoning in intelligent systems - Networks of plausible inference"***, Morgan - Kaufmann 1988.
14. S. M. Aji and R. J. McEliece, ***"The generalized distributive law"*** IEEE Trans. Inform. Theory, pp. 325–343, March 2000.
15. G. D. Forney, ***"Codes on graphs: Normal realizations"*** IEEE Trans. Inform. Theory, pp. 520–548, Feb. 2001.
16. F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, ***"Factor graphs and the sum product algorithm"*** IEEE Trans. Inform. Theory, Feb. 2001.
17. N. Wiberg, H.-A. Loeliger, and R. Kotter, ***"Codes and iterative decoding on general graphs"*** Eur. Trans. Telecommun., pp. 513–525," Sept./Oct. 1995.
18. N. Wiberg, H.-A. Loeliger, and R. Kotter, **"Codes and iterative decoding on general graphs."** Eur. Trans. Telecommun., pp. 513–525," Sept./Oct. 1995.
19. D. Divsalar, H. Jin, and R. McEliece, ***"Coding theorems for 'Turbo-like' codes"*** in Proceedings of the 1998 Allerton Conference, pp. 201-210, 1998.

20. T. J. Richardson, M. Shokrollahi, and R. Urbanke, ***"Design of capacity-approaching irregular low-density parity-check code"*** IEEE Trans. Inform. Theory, pp. 619–637, Feb. 2001.

21. T. J. Richardson and R. Urbanke***, "The capacity of low-density parity-check codes under message-passing decoding."*** IEEE Trans. Inform. Theory, pp. 599–618 Feb. 2001.

22. D.A. Huffman, ***"A Method for the Construction of Minimum-Redundancy Codes"***, Proceedings of the I.R.E., September 1952, pp 1098–1102. Huffman's original article.

23. B. S. Nugroho, *"LDPC Code. Using MATLAB and C MEX"*, Disponível em: <https://sites.google.com/site/bsnugroho/ldpc>. Acesso em 02/12/2011.

ANEXO A - CÓDIGO-FONTE EM LINGUAGEM MATLAB. SIMULAÇÃO DE CODIFICAÇÃO/DECODIFICAÇÃO LPDC EM CANAL AWGN.

A.1) ldpcBER.m

```
% Bit error rate of BPSK modulated LDPC codes under AWGN channel
%
%
% Copyright Bagawan S. Nugroho, 2007
% http://bsnugroho.googlepages.com

clc;
clear all;

% LDPC matrix size, rate must be 1/2
% Warning: encoding - decoding can be very long for large LDPC matrix!
M = 1000;
N = 2000;

% Method for creating LDPC matrix (0 = Evencol; 1 = Evenboth)
method = 1;

% Eliminate length-4 cycle
noCycle = 1;

% Number of 1s per column for LDPC matrix
onePerCol = 3;

% LDPC matrix reorder strategy (0 = First; 1 = Mincol; 2 = Minprod)
strategy = 2;

% EbN0 in dB
EbN0 = [0 0.5 1 1.5];

% Number of iteration;
iter = 5;

% Number of frame (N bits per frame)
frame = 10;

% Make the LDPC matrix
H = makeLdpc(M, N, 1, 1, onePerCol);

for i = 1:length(EbN0)

   ber1(i) = 0;
   ber2(i) = 0;

   % Make random data (0/1)
   dSource = round(rand(M, frame));

   for j = 1:frame
      fprintf('Frame : %d\n', j);
```

```
      % Encoding message
      [c, newH] = makeParityChk(dSource(:, j), H, strategy);
      u = [c; dSource(:, j)];

      % BPSK modulation
      bpskMod = 2*u - 1;
      % Additional white gaussian noise
      N0 = 1/(exp(EbN0(i)*log(10)/10));
      tx = bpskMod + sqrt(N0/2)*randn(size(bpskMod));

      % Decoding (select decoding method)
      vhat1 = decodeProbDomain(tx, H, newN0, iter);
      vhat2 = decodeBitFlip(tx, newH, ter);

      % Get bit error rate (for brevity, BER calculation includes parity
bits)
      [num1, rat1] = biterr(vhat1', u);
      ber1(i) = (ber1(i) + rat1);

      [num2, rat2] = biterr(vhat2', u);
      ber2(i) = (ber2(i) + rat2);

   end % for j

   % Get average of BER
   ber1(i) = ber1(i)/frame;
   ber2(i) = ber2(i)/frame;

end % for i

% Plot the result
semilogy(EbN0, ber1, 'o-');
hold;
semilogy(EbN0, ber2, 'o--');
grid on;
hold off;
```

A.2) makeLdpc.m

```
function H = makeLdpc(M, N, method, noCycle, onePerCol)
% Create R = 1/2 low density parity check matrix
%
%  M         : Number of row
%  N         : Number of column
%  method    : Method for distributing non-zero element
%              {0} Evencol : For each column, place 1s uniformly at random
%              {1} Evenboth: For each column and row, place 1s uniformly at
random
%  noCyle    : Length-4 cycle
%              {0} Ignore (do nothing)
%              {1} Eliminate
%  onePerCol: Number of ones per column
%
%  H         : Low density parity check matrix
%
%
% Copyright Bagawan S. Nugroho, 2007
% http://bsnugroho.googlepages.com
```

```
% Number of ones per row (N/M ratio must be 2)
if N/M ~= 2
   fprintf('Code rate must be 1/2\n');
end
onePerRow = (N/M)*onePerCol;

fprintf('Creating LDPC matrix...\n');

switch method
   % Evencol
   case {0}
      % Distribute 1s uniformly at random within column
      for i = 1:N
         onesInCol(:, i) = randperm(M)';
      end

      % Create non zero elements (1s) index
      r = reshape(onesInCol(1:onePerCol, :), N*onePerCol, 1);
      tmp = repmat([1:N], onePerCol, 1);
      c = reshape(tmp, N*onePerCol, 1);

      % Create sparse matrix H
      H = full(sparse(r, c, 1, M, N));

   % Evenboth
   case {1}
      % Distribute 1s uniformly at random within column
      for i = 1:N
         onesInCol(:, i) = randperm(M)';
      end

      % Create non zero elements (1s) index
      r = reshape(onesInCol(1:onePerCol, :), N*onePerCol, 1);
      tmp = repmat([1:N], onePerCol, 1);
      c = reshape(tmp, N*onePerCol, 1);

      % Make the number of 1s between rows as uniform as possible

      % Order row index
      [r, ix] = sort(r);

      % Order column index based on row index
      for i = 1:N*onePerCol
         cSort(i, :) = c(ix(i));
      end

      % Create new row index with uniform weight
      tmp = repmat([1:M], onePerRow, 1);
      r = reshape(tmp, N*onePerCol, 1);

      % Create sparse matrix H
      % Remove any duplicate non zero elements index using logical AND
      S = and(sparse(r, cSort, 1, M, N), ones(M, N));
      H = full(S);

end % switch

% Check rows that have no 1 or only have one 1
```

```
for i = 1:M

   n = randperm(N);
   % Add two 1s if row has no 1
   if length(find(r == i)) == 0
      H(i, n(1)) = 1;
      H(i, n(2)) = 1;
   % Add one 1 if row has only one 1
   elseif length(find(r == i)) == 1
      H(i, n(1)) = 1;
   end

end % for i

% If desired, eliminate any length-4 cycle
if noCycle == 1

   for i = 1:M
      % Look for pair of row - column
      for j = (i + 1):M
         w = and(H(i, :), H(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(H(i, :))) < length(find(H(j, :)))
               % Repeat the process until only one column left
               for cc = 1:lc - 1
                  H(j, c1(cc)) = 0;
               end
            else
               for cc = 1:lc - 1
                  H(i, c1(cc)) = 0;
               end
            end % if

         end % if
      end % for j
   end % for i

end % if

fprintf('LDPC matrix is created.\n');
```

A.3) makeParityChk.m

```
function [c, newH] = makeParityChk(dSource, H, strategy)
% Generate parity check vector bases on LDPC matrix H using sparse LU
decomposition
%
%  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

   % 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

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

% Parity check vector found by solving sparse LU
c = mod(U\(L\z), 2);
```

```
% Return the rearrange H
newH = H;

fprintf('Message encoded.\n');
```

A.4) decodeProbDomain.m

```
function vHat = decodeProbDomain(rx, H, N0, iteration)
% Probability-domain sum product algorithm LDPC decoder
%
%  rx        : Received signal vector (column vector)
%  H         : LDPC matrix
%  N0        : Noise variance
%  iteration : Number of iteration
%
%  vHat      : Decoded vector (0/1)
%
%
% Copyright Bagawan S. Nugroho, 2007
% http://bsnugroho.googlepages.com

[M N] = size(H);

% Prior probabilities
P1 = ones(size(rx))./(1 + exp(-2*rx./(N0/2)));
P0 = 1 - P1;

% Initialization
K0 = zeros(M, N);
K1 = zeros(M, N);
rji0 = zeros(M, N);
rji1 = zeros(M, N);
qij0 = H.*repmat(P0', M, 1);
qij1 = H.*repmat(P1', M, 1);

% Iteration
for n = 1:iteration

   fprintf('Iteration : %d\n', n);

   % ----- Horizontal step -----
   for i = 1:M

      % Find non-zeros in the column
      c1 = find(H(i, :));

      for k = 1:length(c1)

         % Get column products of drji\c1(l)
         drji = 1;
         for l = 1:length(c1)
            if l~= k
               drji = drji*(qij0(i, c1(l)) - qij1(i, c1(l)));
            end
         end % for l

         rji0(i, c1(k)) = (1 + drji)/2;
         rji1(i, c1(k)) = (1 - drji)/2;
```

```
        end % for k

    end % for i

    % ------ Vertical step ------
    for j = 1:N

        % Find non-zeros in the row
        r1 = find(H(:, j));

        for k = 1:length(r1)

            % Get row products of prodOfrij\ri(l)
            prodOfrij0 = 1;
            prodOfrij1 = 1;
            for l = 1:length(r1)
                if l~= k
                    prodOfrij0 = prodOfrij0*rji0(r1(l), j);
                    prodOfrij1 = prodOfrij1*rji1(r1(l), j);
                end
            end % for l

            % Update constants
            K0(r1(k), j) = P0(j)*prodOfrij0;
            K1(r1(k), j) = P1(j)*prodOfrij1;

            % Update qij0 and qij1
            qij0(r1(k), j) = K0(r1(k), j)./(K0(r1(k), j) + K1(r1(k), j));
            qij1(r1(k), j) = K1(r1(k), j)./(K0(r1(k), j) + K1(r1(k), j));

        end % for k

        % Update constants
        Ki0 = P0(j)*prod(rji0(r1, j));
        Ki1 = P1(j)*prod(rji1(r1, j));

        % Get Qj
        Qi0 = Ki0/(Ki0 + Ki1);
        Qi1 = Ki1/(Ki0 + Ki1);

        % Decode Qj
        if Qi1 > Qi0
            vHat(j) = 1;
        else
            vHat(j) = 0;
        end

    end % for j

end % for n
```

A.4) decodeBitFlip.m

```
function vHat = decodeBitFlipping(rx, H, iteration)
% Hard-decision/bit flipping sum product algorithm LDPC decoder
%
%  rx        : Received signal vector (column vector)
%  H         : LDPC matrix
%  iteration : Number of iteration
%
%  vHat      : Decoded vector (0/1)
%
%
% Copyright Bagawan S. Nugroho, 2007
% http://bsnugroho.googlepages.com

[M N] = size(H);

% Prior hard-decision
ci = 0.5*(sign(rx') + 1);

% Initialization
rji = zeros(M, N);

% Asscociate the ci matrix with non-zero elements of H
qij = H.*repmat(ci, M, 1);

% Iteration
for n = 1:iteration

   fprintf('Iteration : %d\n', n);

   % ----- Horizontal step -----
   for i = 1:M

      % Find non-zeros in the column
      c1 = find(H(i, :));

      % Get the summation of qij\c1(k)
      for k = 1:length(c1)

         rji(i, c1(k)) = mod(sum(qij(i, c1)) + qij(i, c1(k)), 2);

      end % for k

   end % for i

   % ------ Vertical step ------
   for j = 1:N

      % Find non-zero in the row
      r1 = find(H(:, j));

      % Number of 1s in a row
      numOfOnes = length(find(rji(r1, j)));

      for k = 1:length(r1)

         % Update qij, set '1' for majority of 1s else '0', excluding r1(k)
```

```
            if numOfOnes + ci(j) >= length(r1) - numOfOnes + rji(r1(k), j)
               qij(r1(k), j) = 1;
            else
               qij(r1(k), j) = 0;
            end

         end % for k

         % Bit decoding
         if numOfOnes + ci(j) >= length(r1) - numOfOnes
            vHat(j) = 1;
         else
            vHat(j) = 0;
         end

      end % for j

   end % for n
```