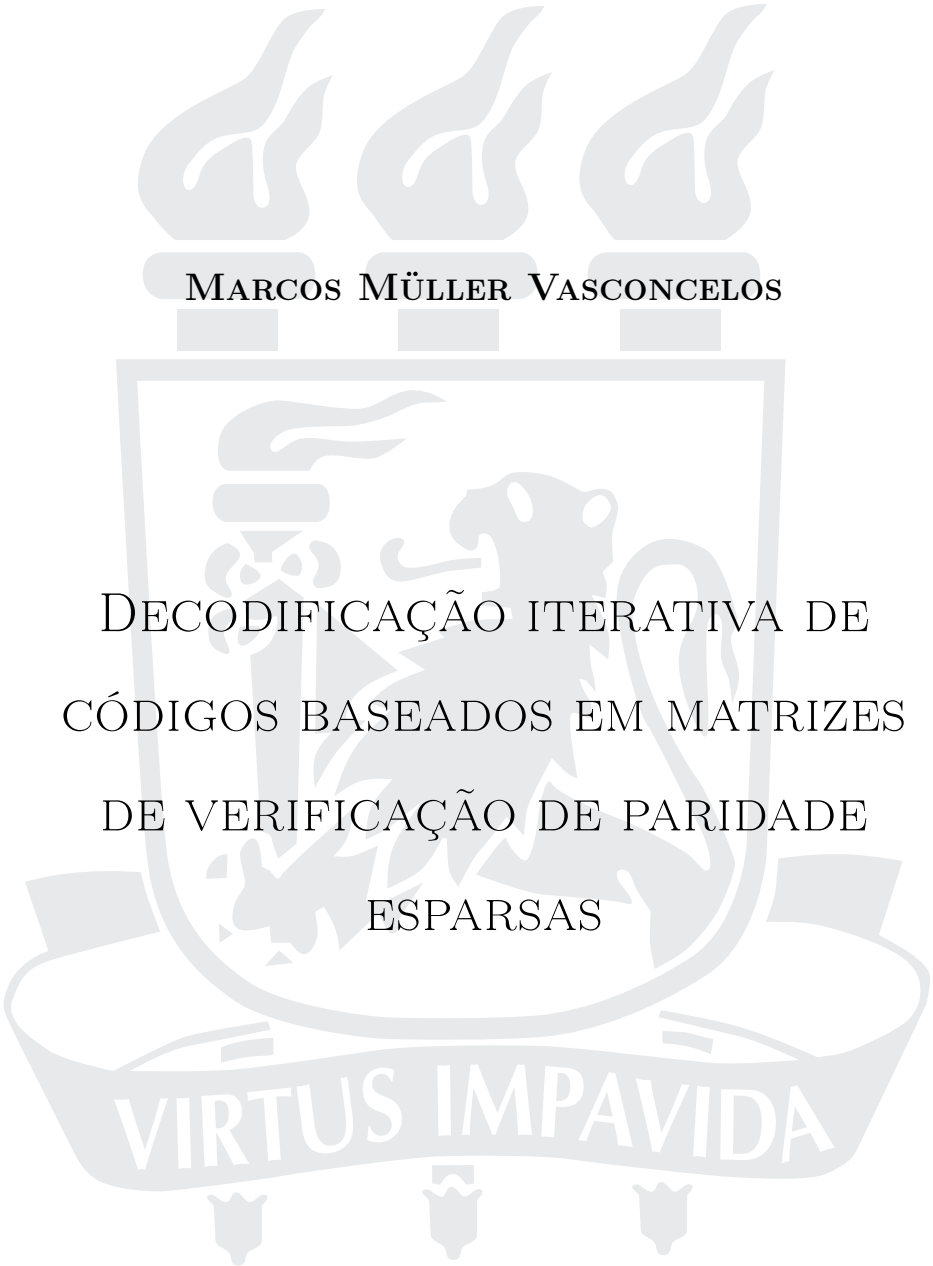
Universidade Federal de Pernambuco
Centro de Tecnologia e Geociências
Programa de Pós-graduação em Engenharia Elétrica



Marcos Müller Vasconcelos

Decodificação iterativa de códigos baseados em matrizes de verificação de paridade esparsas

Recife, Abril de 2006.

Marcos Müller Vasconcelos

Decodificação iterativa de códigos baseados em matrizes de verificação de paridade esparsas

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Pernambuco como parte dos requisitos para obtenção do grau de **Mestre em Engenharia Elétrica**

Orientador: Prof. Valdemar Cardoso da Rocha Jr, Ph.D.

Recife, Abril de 2006.

V331d **Vasconcelos, Marcos Müller**

Decodificação iterativa de códigos baseados em matrizes de verificação de paridade esparsas / Marcos Müller Vasconcelos. - Recife: O Autor, 2006.

96 f. : il., gráfs., tabs.

Dissertação (Mestrado) – Universidade Federal de Pernambuco. CTG. Dpto. de Engenharia Elétrica. Programa de Pós-Graduação em Engenharia Elétrica, 2006.

Inclui referências.

1. Engenharia Elétrica. 2. Códigos Low-Density Parity-Check. 3. Canal Gilbert-Elliot. 4. Algoritmo Soma-Produto. 5. Decodificação Iterativa. 6. Grafos Fator. I. Título.

621.3 CDD (22. ed.) **BCTG/2007-91**

Universidade Federal de Pernambuco

Pós-Graduação em Engenharia Elétrica

PARECER DA COMISSÃO EXAMINADORA DE DEFESA DE
TESE DE MESTRADO ACADÊMICO DE

MARCOS MÜLLER VASCONCELOS

TÍTULO

"DECODIFICAÇÃO ITERATIVA DE CÓDIGOS BASEADOS EM MATRIZES DE VERIFICAÇÃO DE PARIDADE ESPARSAS"

A comissão examinadora composta pelos professores: VALDEMAR CARDOSO DA ROCHA JÚNIOR, DES/UFPE, CECÍLIO JOSÉ LINS PIMENTEL, DES/UFPE e JAIME PORTUGHEIS, DC/UNICAMP, sob a presidência do primeiro, consideram o candidato **MARCOS MÜLLER VASCONCELOS APROVADO**.

Recife, 17 de abril de 2006.

JOAQUIM FERREIRA MARTINS FILHO
Coordenador do PPGEE

VALDEMAR CARDOSO DA ROCHA JÚNIOR
Orientador e Membro Titular Interno

JAIME PORTUGHEIS
Membro Titular Externo

CECÍLIO JOSÉ LINS PIMENTEL
Membro Titular Interno

Aos meus pais,

Armando e **Ana**,

pela força que sempre me deram

para enfrentar os desafios da vida.

Agradecimentos

Gostaria de agradecer a todos que colaboraram direta ou indiretamente na elaboração desta dissertação, especialmente ao professor Valdemar Cardoso da Rocha Jr, que desde o início da minha trajetória acadêmica me orienta com muita dedicação, carinho e amizade.

Aos professores Ricardo Campello e Hélio Magalhães, por terem me ensinado o verdadeiro significado das palavras *professor* e *pesquisador*. Seu caráter, dedicação e amor à engenharia sempre serão exemplos a serem seguidos por mim. Ao professor Cecilio Pimentel, pelas longas conversas sobre códigos, grafos, algoritmos de decodificação e tantos outros assuntos interessantes. À professora Marcia Mahon, que ministrou o meu primeiro curso de códigos corretores de erros, iniciando o meu interesse pelo assunto.

Aos colegas do grupo de Engenharia de Sistemas do professor Fernando Campello pelas discussões sempre interessantes sobre os mais diversos assuntos. Aos bons amigos que fiz no programa de pós-graduação: Sérgio Campello, Carmelo Bastos, Eric Arantes, André Leite, Danielle Barros, Maria de Lourdes Alcoforado e Renato Cintra. É um honra ter amigos tão talentosos e inteligentes. Aos meu grandes amigos e futuros parceiros de pesquisa Danilo Silva, Eric Bouton e Leonardo Limongi.

Finalmente, agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro ao longo destes dois anos.

Marcos Müller Vasconcelos

Universidade Federal de Pernambuco

17 de Abril de 2006

A teoria é o general;
os experimentos são os soldados.

— **Leonardo da Vinci**

Resumo da Dissertação apresentada à UFPE como parte dos requisitos necessários para a obtenção do grau de Mestre em Engenharia Elétrica

Decodificação iterativa de códigos baseados em matrizes de verificação de paridade esparsas

Marcos Müller Vasconcelos

Abril/2006

Orientador: Prof. Valdemar Cardoso da Rocha Jr, Ph.D.
Área de Concentração: Comunicações
Palavras-chaves: códigos low-density parity-check, canal Gilbert-Elliott, algoritmo Soma-Produto, decodificação iterativa, grafos fator.
Número de páginas: 96

Códigos baseados em matrizes esparsas têm desempenhado um importante papel em teoria da codificação. Os códigos *low-density parity-check* (LDPC) constituem uma famosa família de códigos definidos a partir de matrizes de verificação de paridade esparsas que apresentam desempenhos excelentes no canal com ruído aditivo Gaussiano branco (RAGB). O sucesso desses códigos se deve a sua representação através de grafos, que permite a operação de um algoritmo de decodificação iterativo cuja complexidade cresce linearmente com o comprimento dos blocos. Esta dissertação apresenta um estudo sobre códigos LDPC e sua principal ferramenta de análise, a *density evolution*. Para isso, a representação gráfica de códigos de bloco lineares e o funcionamento do algoritmo de decodificação Soma-Produto são apresentados. Algumas técnicas de projeto de códigos LDPC são discutidas e seu desempenho no canal RAGB é avaliado por meio de simulações. Baseando-se nestas ferramentas, a *density evolution* para os canal RAGB é derivada em forma integral e em forma aproximada. Por fim, uma modificação no algoritmo Soma-Produto é proposta para decodificação de códigos LDPC no canal Gilbert-Elliott.

Abstract of Dissertation presented to UFPE as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering

Iterative decoding of codes based on sparse parity-check matrices

Marcos Müller Vasconcelos

April/2006

Supervisor: Prof. Valdemar Cardoso da Rocha Jr, Ph.D.
Area of Concentration: Communications
Keywords: low-densiy parity-check codes, Gilbert-Elliott channel, Sum-Product algorithm, iterative decoding, factor graphs
Number of pages: 96

Codes based on sparse matrices are playing an important role in coding theory lately. The low-density parity-check (LDPC) codes constitute a famous family of codes defined by sparse parity-check matrices which have excellent performance under iterative decoding over the additive white Gaussian noise (AWGN) channel. The success of these codes is in a great part due to their graph based representation, which allows their efficient decoding through an iterative algorithm whose complexity grows linearly with the block length. This dissertation addresses the study of LDPC codes and their analysis using density evolution. In order to accomplish this objective, the graphical representation of linear block codes and the Sum-Product decoding algorithm are presented. Some construction techniques for LDPC codes are discussed and their performance over the AWGN channel is determined by simulations. Based on these tools, density evolution for the AWGN channel is derived in both integral and approximated forms. Finally, the Sum-Product algorithm is modified in a novel scheme for decoding LDPC codes over the Gilbert-Elliott channel.

Lista de Figuras

Lista de Tabelas

Conteúdo

capítulo 1

Introdução

Códigos baseados em matrizes esparsas têm desempenhado um importante papel em teoria da codificação [1]. Os códigos *low-density parity-check* (LDPC) [2] constituem uma famosa família de códigos definidos a partir de matrizes de verificação de paridade esparsas que apresentam desempenhos excelentes em uma grande variedade de canais. O sucesso desses códigos se deve a três fatores principais: 1) os códigos LDPC podem ser representados eficientemente através de grafos; 2) a representação gráfica permite a decodificação eficiente através de um algoritmo iterativo baseado em *belief propagation* cuja complexidade cresce linearmente com o comprimento da palavra-código; e 3) a existência de uma técnica analítica conhecida como *density evolution* para analisar e projetar códigos com desempenho próximo da capacidade de diversos canais.

Esta dissertação trata da decodificação iterativa de códigos definidos por matrizes de verificação de paridade esparsas. O principal interesse é em apresentar a teoria por trás do algoritmo de decodificação Soma-Produto (SP) e analisar seu desempenho. Como contribuição, é proposta uma modificação no algoritmo SP para decodificação de códigos LDPC em um canal com memória, o canal Gilbert-Elliott.

1.1 Comunicação em canais ruidosos

Em sistemas de comunicação digital é desejável transmitir informação com baixa probabilidade de erro de bit. Duas possíveis soluções para se atingir este objetivo são: 1) construir circuitos com componentes mais confiáveis e aumentar a potência dos sinais transmitidos; e 2)

codificar a informação a ser transmitida, inserindo redundância de maneira controlada, para reduzir os efeitos da distorção introduzida pelo canal ruidoso. A solução 1 implica em melhorar o canal fisicamente; enquanto a solução 2, ilustrada na Figura 1.1, possibilita uma diminuição da probabilidade de erro em troca de um aumento de complexidade na implementação do sistema de comunicação.

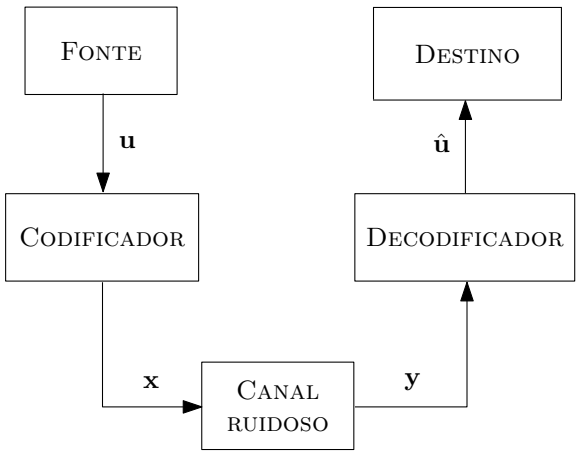


Figura 1.1: *Sistema de comunicações codificado.*

No sistema mostrado na Figura 1.1, a mensagem $\mathbf{u}$ é um vetor de comprimento K. Este vetor é codificado, *i.e.*, mapeado em um vetor $\mathbf{x}$ de comprimento N e transmitido através de um canal ruidoso. O vetor recebido $\mathbf{y}$ é então processado por um decodificador, que faz uso da redundância introduzida na codificação para obter uma estimativa da informação transmitida $\hat{\mathbf{u}}$. A codificação é uma forma de aumentar a confiabilidade do sistema. Um dos objetivos da teoria da codificação é projetar códigos que maximizem a taxa de transmissão e minimizem a probabilidade de erro.

Em 1948, C. E. Shannon [3] estabeleceu os limites teóricos da transmissão de informação através de um canal ruidoso. Baseado no teorema da codificação de canal, Shannon mostrou que com um par codificador/decodificador apropriado, é possível obter taxas de erro arbitrariamente pequenas desde que a taxa de transmissão seja menor do que uma grandeza definida como a capacidade do canal. Apesar de garantir a existência, o teorema não mostra como construir tal código e durante muitos anos buscou-se por códigos capazes de atingir a capacidade de diversos modelos de canal.

Entretanto, a busca por codificadores/decodificadores possui um limitador crucial: a complexidade de decodificação. A decodificação ótima de códigos é um problema NP-completo,

i.e., o melhor algoritmo conhecido para resolvê-lo possui uma complexidade que aumenta exponencialmente com o comprimento do código [4]. Por isso, os pesquisadores de teoria da codificação procuraram projetar códigos que apresentassem bom desempenho e que ao mesmo tempo apresentassem propriedades algébricas que permitissem uma decodificação eficiente com baixa complexidade. Porém, apenas em 1993, com a introdução dos códigos Turbo por C. Berrou, A. Glavieux e P. Thitimajshima [5] mostrou-se que é possível se aproximar do limite de Shannon, com uma complexidade aceitável.

Ao atingir um desempenho excelente no canal com ruído aditivo gaussiano branco (RAGB), os códigos Turbo marcaram o início de uma nova era na teoria da codificação. Ao invés de utilizar um algoritmo ótimo, o decodificador Turbo implementa um algoritmo de decodificação iterativo sub-ótimo cuja complexidade cresce linearmente com o comprimento do código [6]. Além disso, estes códigos possuem outras características desejáveis, tais como: 1) um codificador simples; 2) podem ser representados através de grafos; 3) o algoritmo de decodificação é um caso especial de *belief propagation* [7]; e 4) o desempenho do código melhora quando aumenta o comprimento da maioria dos ciclos no grafo que representa o código. O rompimento no paradigma da decodificação ótima levou à redescoberta de uma classe de códigos há muito esquecida e que compartilham algumas das características dos códigos Turbo: os códigos *low-density parity-check*.

1.2 Códigos com matrizes de verificação de paridade esparsas

Os códigos LDPC são definidos a partir de matrizes de verificação de paridade esparsas, isto é, o número de elementos não-nulos é muito menor que o número de elementos nulos. Estes códigos foram introduzidos por R. Gallager em 1963 [2] mas foram esquecidos pois sua complexidade era considerada inviável para a capacidade computacional da época. Entretanto, a intensa atividade de pesquisa em códigos Turbo e decodificação iterativa despertou o interesse em diversos esquemas semelhantes, resultando na redescoberta dos códigos LDPC por diversos grupos de pesquisa. O artigo de D. Mackay e R. Neal [8] publicado em 1996 é creditado por esta redescoberta, juntamente com o trabalho de M. Sipser e D. Spielman [9] e de N. Wiberg [10].

Assim como os códigos Turbo, os códigos LDPC podem ser representados através de grafos. Em 1981, R. Tanner introduziu uma representação gráfica para códigos de bloco que ficou conhecida como grafos de Tanner [11]. Mais tarde, a representação de Tanner foi generalizada

através da definição dos grafos fator. A decodificação baseada em *belief propagation* é obtida através do algoritmo Soma-Produto (SP) [12]. Este algoritmo opera através da passagem de mensagens pelos ramos do grafo fator que representa o código. A complexidade do algoritmo SP aumenta linearmente com o comprimento do código e, mesmo sendo um algoritmo de decodificação sub-ótimo para grafos com ciclos, permite que códigos LDPC apresentem um desempenho próximo da capacidade do canal RAGB [8].

Os códigos LDPC propostos por Gallager são regulares; isto acontece quando o número de elementos não nulos (peso de Hamming) das linhas (ou colunas) da matriz de verificação de paridade é o mesmo para toda linha (ou coluna). Este tipo de código apresenta um desempenho empírico comparável aos códigos Turbo para grandes comprimentos de bloco ($N \approx 10^5$). Entretanto, os códigos LDPC possuem a vantagem de não apresentar *patamares de erros*[1] a baixas taxas de erro de bit (*bit error rate* ou BER) como os códigos Turbo, fenômeno atribuído a sua baixa distância livre [13]. Além disso os códigos LDPC não necessitam de longos entrelaçadores para alcançar bom desempenho.

Um grande avanço na teoria dos códigos LDPC foi o desenvolvimento de uma técnica que estabelece analiticamente que, quando decodificados com o algoritmo SP, é possível atingir probabilidades de erro arbitrariamente pequenas quando o comprimento do código aumenta. Este resultado foi provado por T. Richardson e R. Urbanke [14] com o auxílio de uma técnica chamada *Density Evolution* (DE). A DE é usada para calcular as funções densidade de probabilidade das mensagens trocadas pelos nós no grafo a cada iteração do algoritmo de decodificação. A análise via DE é computacionalmente intensa e diversos métodos para simplificá-la têm sido propostos, como a análise por aproximação Gaussiana [15].

Um outro passo importante foi a descoberta que códigos LDPC irregulares que apresentam um desempenho ainda mais próximo da capacidade do canal RAGB [16]. Em um código irregular, os pesos das linhas e colunas da matriz de verificação de paridade são governados por distribuições de probabilidade chamadas distribuições de grau e não são constantes. A DE pode ser utilizada para encontrar boas distribuições de grau que originam códigos com um desempenho, que em muitos casos superam os códigos Turbo [17]. Recentemente, um código LDPC irregular capaz de trabalhar a 0.0045 dB do limite de Shannon para o canal RAGB foi construído [18]. Após o sucesso dos códigos LDPC no canal RAGB, a comunidade científica passou a estudar o seu comportamento em canais mais complexos, por exemplo, canais com

[1] Região da curva de desempenho em que o aumento da relação sinal-ruído não implica em uma diminuição significativa da taxa de erro de bit.

memória [19].

1.3 Codificação para canais com memória

O ruído introduzido por um canal sem memória em um determinado instante de tempo é modelado por uma variável aleatória independente dos valores passados ou futuros por ela assumidos [20]. Alguns exemplos de canais sem memória são os canais RAGB e o binário simétrico (*binary symmetric channel* ou BSC). Em um canal com memória, o ruído introduzido depende do estado em que se encontra o canal; e os estados do canal são variáveis aleatórias dependentes. O canal com desvanecimento Rayleigh correlacionado no tempo e o canal Gilbert-Elliott são exemplos de canais com memória. Um canal com memória pode ser dependente ou independente da forma de onda por ele transmitida. Nesta dissertação serão considerados apenas os canais cujo estado atual independe da forma de onda transmitida, ou seja, são independentes dos dados transmitidos/armazenados.

Várias estratégias podem ser utilizadas para realizar comunicação confiável em canais com memória. Uma possível solução implementa a estimação do estado do canal a cada intervalo de transmissão e usa esta informação para decodificar os símbolos transmitidos. Esta técnica é chamada de *decodificação por estimação*. Outra técnica consiste em entrelaçar as palavras-código transmitidas pelo canal de forma a diminuir a correlação entre os erros contidos em um mesmo bloco. O entrelaçamento não elimina a memória do canal e sim a transforma em uma memória *latente*. Após o desentrelaçamento, utiliza-se um decodificador para canais sem memória para estimar a seqüência de informação transmitida. Apesar de eficaz, esta estratégia não utiliza a memória *latente* do canal. Uma forma de melhorar o sistema com entrelaçamento é incorporar ao decodificador a informação adicional sobre a memória do canal usando probabilidades de erro condicionadas aos erros anteriores no canal.

O canal Gilbert-Elliott (GE) [21, 22] foi inicialmente proposto para modelar redes telefônicas em que o chaveamento de linhas de transmissão de alta tensão, induzia surtos de erros que não eram capazes de ser modelados por canais sem memória. Este canal também pode ser utilizado para modelar um grande número de situações práticas como canais telefônicos, enlaces de microondas, canais de comunicação via-satélite, sistemas de gravação óptica e magnética, canais de acesso múltiplo por espalhamento espectral, desvanecimento em canais de rádio móvel e sistemas de comunicação com perda de sincronismo [23]. A capacidade do canal GE foi determinada por M. Mushkin e I. Bar-David [24]. Mushkin e Bar-David pro-

puseram um esquema que permite a decodificação em um sistema com entrelaçamento, que preserva a capacidade do canal GE, chamado *decodificação por decisão com realimentação*. Recentemente, uma análise de desempenho de códigos LDPC em canais GE foi proposta por A. Eckford usando um esquema de decodificação por estimação [19]. O esquema utilizado por Eckford utiliza ferramentas e algoritmos desenvolvidos especificamente para canais com memória e, portanto são ferramentas mais elaboradas e complexas que as tradicionais.

Nesta dissertação, é proposto um sistema de decodificação por decisão com realimentação para códigos LDPC no canal GE. Este novo sistema utiliza ferramentas para comunicação em canais sem memória e, com um pequeno aumento na complexidade do decodificador, pode apresentar melhoria significativa de desempenho em relação a estratégia que utiliza apenas o entrelaçamento.

1.4 Objetivos

Esta dissertação apresenta um estudo sobre códigos LDPC e sua principal ferramenta de análise, a *density evolution*. Para isso, é necessário introduzir a representação gráfica de códigos de bloco lineares e o funcionamento do algoritmo Soma-Produto. Outro objetivo é discutir algumas técnicas de projeto de códigos LDPC e avaliar seu desempenho no canal RAGB por meio de simulações computacionais. Baseando-se nestas ferramentas, a *density evolution* para os canal RAGB é derivada em forma integral e em forma aproximada. Por fim, uma modificação no algoritmo Soma-Produto é proposta para decodificação de códigos LDPC no canal Gilbert-Elliott.

1.5 Organização da dissertação

O conteúdo da dissertação está distribuído ao longo de sete capítulos. As referências bibliográficas do trabalho são apresentadas nas últimas páginas do texto e encontram-se organizadas na ordem em que foram citadas. Um breve resumo de cada capítulo é mostrado abaixo.

Capítulo 2. Os fundamentos da representação de códigos através de grafos são apresentados.

Capítulo 3. O algoritmo Soma-Produto é introduzido e utilizado na decodificação de códigos de bloco em canais sem memória.

Capítulo 4. A estrutura e algumas técnicas de construção dos códigos LDPC são descritas.

Capítulo 5. A técnica *density evolution* para análise de desempenho e convergência do algoritmo Soma-Produto é apresentada.

Capítulo 6. Um sistema de decodificação do tipo decisão com realimentação para decodificação de códigos LDPC em canais Gilbert-Elliott, é proposto.

Capítulo 7. Por fim, algumas conclusões e sugestões para trabalhos futuros são discutidas.

CAPÍTULO 2

MODELOS GRÁFICOS PARA CÓDIGOS DE BLOCO LINEARES

Na teoria de códigos de códigos corretores de erros, diversas classes de códigos podem ser representadas graficamente de forma conveniente. As *treliças* são um exemplo de representação gráfica de códigos convolucionais e de bloco. A descrição de códigos de bloco lineares por meio de grafos foi definida diversas vezes. No contexto de códigos LDPC, o modelo utilizado se baseia em *grafos fator* [12] que, por sua vez possuem suas origens nos trabalhos de R. M. Tanner [11]. Tanner utilizou grafos bipartite para representar as restrições que os símbolos de uma palavra-código devem satisfazer. Os grafos fator estão relacionados aos grafos de Tanner e às *redes Bayesianas*. Isto permite que um algoritmo de decodificação baseado em *belief propagation*[1] chamado algoritmo Soma-Produto seja utilizado. Os conceitos e exemplos descritos neste capítulo seguem o tratamento dado em [12].

De agora em diante, um código de bloco linear binário será denotado por $\mathcal{C}(N, K)$, no qual as palavras-código são vetores binários de comprimento N e possuem K dígitos de informação. Este conjunto de palavras-código forma um subespaço K-dimensional do espaço vetorial $\mathbb{F}_2^N$ das N-uplas binárias. A taxa de $\mathcal{C}(N, K)$ é $R = K/N$ e a soma entre duas variáveis binárias x e y será denotada por $x \oplus y$, na qual $\oplus$ significa a operação "OU-EXCLUSIVO".

[1] Método probabilístico utilizado para atualizar e obter probabilidades marginais a partir de um conjunto de variáveis observadas.

2.1 Grafos de Tanner

Esta seção inicia com algumas definições necessárias para se introduzir o modelo gráfico proposto por Tanner para códigos de bloco lineares.

Definição 2.1 (Grafo) *Um grafo $\mathcal{G}$ é uma estrutura que consiste de um conjunto de vértices (ou nós), denotado por $V = \{v_1, v_2, \dots\}$, e um conjunto de ramos, denotado por $E = \{e_1, e_2, \dots\}$. Cada ramo e_k pertencente ao conjunto E está associado a um par (ordenado ou não-ordenado) (v_i, v_j) de vértices (não necessariamente distintos) do conjunto V. Os vértices v_i e v_j são chamados de* ***vértices terminais*** *de e_k. O grafo $\mathcal{G}$ é denotado por $\mathcal{G} = (V, E)$.*

Normalmente, um grafo é representado por um diagrama através de pontos correspondendo aos vértices e linhas correspondendo aos ramos. Um ramo em um grafo $\mathcal{G}$ pode ser *direcionado*, quando o par de vértices que o define é ordenado, ou *não-direcionado* caso contrário. O grafo $\mathcal{G}$ é dito *simples* se: 1) não existe nenhum ramo conectando um nó a si mesmo; 2) existe no máximo um ramo entre quaisquer par de vértices; e 3) todos os ramos são não-direcionados. Em um grafo simples, dois vértices x e y pertencentes ao conjunto V são *adjacentes* se o par (x, y) corresponde a um ramo em E. O conjunto de todos os vértices que são adjacentes a x é chamado de *vizinhança* de x.

Um subgrafo do grafo $\mathcal{G} = (V, E)$ é um grafo cujos conjuntos de vértices e de ramos são ambos subconjuntos dos de $\mathcal{G}$. Um subgrafo de $\mathcal{G}$ é denotado por $\mathcal{G}' = (V', E')$. Para cada ramo $e_k \in E'$, ambos vértices de e_k devem pertencer a V'.

Uma seqüência de vértices distintos, iniciada no vértice x e terminada no vértice y, é chamada de *caminho* entre x e y se todos os vértices consecutivos nesta seqüência forem adjacentes. Se existir pelo menos um caminho entre x e y, então x e y são denominados *conectados*. Quando dois vértices x e y são conectados, a *distância* entre x e y é definida como o número de ramos do caminho mais curto entre x e y, e é denotada por $d(x, y)$. Um caminho fechado que inicia e termina em x é chamado de um *ciclo* de x. O *ciclo mínimo* é o comprimento do menor ciclo em um grafo. Para cada vértice, define-se um ciclo mínimo local denotado por g_x como o comprimento do menor ciclo que passa por x. Portanto, o ciclo mínimo g do grafo $\mathcal{G} = (V, E)$ é dado por

$$g = \min_{x \in V}(g_x) \tag{2.1}$$

Os grafos de Tanner se baseiam em uma importante classe de grafos denominada grafos bipartite.

Definição 2.2 (Grafo bipartite) *Um grafo bipartite é um grafo cujo conjunto de vértices V é dividido em dois subconjuntos disjuntos, denotados por V_c e V_s. Nenhum par de vértices pertencentes a um mesmo subconjunto é adjacente.*

A Figura 2.1 é um exemplo de grafo bipartite: os vértices pretos e brancos constituem dois subconjuntos de vértices. Os vértices pretos estão conectados apenas a vértices brancos e vice-versa. No caso da Figura 2.2, os vértices pretos possuem conexões entre si e portanto o grafo não é bipartite.

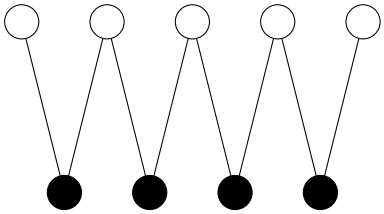


Figura 2.1: *Grafo bipartite*

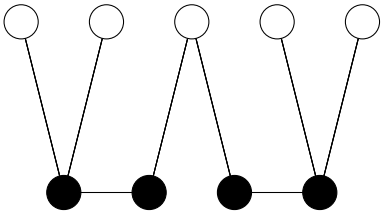


Figura 2.2: *Grafo não-bipartite*

Um código de verificação de paridade de comprimento N é um código de bloco linear no qual as palavras-código satisfazem um conjunto de equações de paridade que restringem as possíveis combinações dos símbolos que as compõem. Um grafo de Tanner para o código linear $\mathcal{C}(N, K)$ é obtido a partir de qualquer sistema de equações de paridade que definem o código. Portanto, podemos construir um grafo de Tanner que representa um código através de uma matriz de verificação de paridade $\mathbf{H}$. Como códigos lineares não possuem matrizes de verificação de paridade únicas, um único código pode ser representado por mais de um grafo de Tanner. A seguir será considerada a matriz de verificação de paridade $\mathbf{H}$ associada ao código de bloco linear sistemático $\mathcal{C}(N, K)$.

Definição 2.3 (Grafo de Tanner) *Um grafo de Tanner de um código de bloco $\mathcal{C}(N, K)$ é um grafo bipartite não-direcionado obtido a partir de uma matriz de verificação de paridade $\mathbf{H}$ com N colunas e M linhas. O grafo de Tanner possui N* ***nós de símbolo*** *correspondentes às colunas de $\mathbf{H}$, e M* ***nós de verificação*** *correspondentes às linhas de $\mathbf{H}$. Um ramo conecta o nó de símbolo x_n ao m-ésimo nó de verificação c_m se e somente se $h_{m,n} = 1$, onde $h_{m,n}$ denota o elemento na m-ésima linha e n-ésima coluna de $\mathbf{H}$.*

Para um grafo de Tanner $\mathcal{G} = (V, E)$, o conjunto dos nós de símbolo é denotado por $V_s = \{x_1, x_2, \cdots, x_N\}$ e o conjunto dos nós de verificação é denotado por $V_c = \{c_1, c_2, \cdots, c_M\}$, de modo que $V = V_s \cup V_c$. Um elemento e_k do conjunto de ramos E é denotado por (c_m, x_n).

O grau de um nó de símbolo (ou de verificação) x_n (ou c_m) em um grafo de Tanner, corresponde ao número de ramos que incidem neste nó e é denotado por $d(x_n)$ (ou $d(c_m)$). Um grafo de Tanner é chamado de *regular* quando $d(x_n) = d_s$, $1 \leq n \leq N$ e $d(c_m) = d_c$, $1 \leq m \leq M$, caso contrário, o grafo é chamado *irregular*. Em um grafo de Tanner, todos os ciclos possuem comprimento par.

Exemplo 2.1 *Considere o código de Hamming* $\mathcal{C}(7, 4)$ *descrito pela seguinte matriz de verificação de paridade* $\mathbf{H}$*:*

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

Uma palavra pertencente a este código $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ *deve satisfazer a seguinte condição:*

$$\mathbf{x}\mathbf{H}^T = \mathbf{0}.$$

Então, as equações de paridade são:

$$\begin{aligned} x_1 \oplus x_2 \oplus x_4 \oplus x_5 &= 0 \\ x_1 \oplus x_3 \oplus x_4 \oplus x_6 &= 0 \\ x_2 \oplus x_3 \oplus x_4 \oplus x_7 &= 0. \end{aligned}$$

O grafo de Tanner para este código possui 3 *nós de verificação (representando as equações de paridade) e* 7 *nós de símbolo (representando os símbolos da palavra-código). Denotando os nós de símbolo por* $\bigcirc$ *e os nós de verificação por* $\boxplus$*, o grafo de Tanner para* $\mathcal{C}(7, 4)$ *é mostrado na Figura 2.3.*

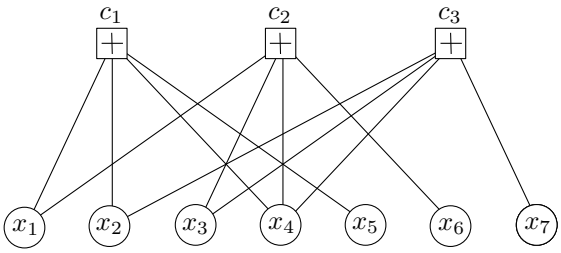


Figura 2.3: *Grafo de Tanner para o código de Hamming* $\mathcal{C}(7,4)$.

O grafo de Tanner mostrado na Figura 2.3 apresenta ciclos. Por exemplo, na Figura 2.3, a seqüência $\{x_1, c_1, x_4, c_2, x_1\}$ constitui um ciclo de comprimento 4.

2.2 Grafos fator

Os grafos fator são uma generalização dos grafos de Tanner. Com eles é possível modelar não apenas códigos lineares como também funções densidade de probabilidade e equações de estado de sistemas dinâmicos. Um grafo fator representa graficamente a estrutura de fatoração de uma função de várias variáveis. Sejam $x_1, x_2, \cdots, x_N$ um conjunto de variáveis. Cada uma dessas variáveis assume valores em um *domínio* (ou alfabeto) $\mathcal{A}_n$, geralmente finito. Seja $g(x_1, x_2, \cdots, x_N)$ uma função com domínio $\mathcal{S} = \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_N$ e imagem $\mathbb{R}$. O domínio $\mathcal{S}$ também é chamado de *espaço de configurações* das variáveis $x_1, x_2, \cdots, x_N$. Quando a função $g(x_1, x_2, \cdots, x_N)$, denominada de função *global*, é fatorável no produto de diversas funções mais simples chamadas *funções locais*, é possível representá-la da seguinte forma:

$$g(x_1, x_2, \cdots, x_N) = \prod_{m \in \mathcal{M}} f_m(\mathbf{x}_m),$$

na qual $\mathcal{M}$ é um conjunto discreto de índices, $\mathbf{x}_m$ é um subconjunto de $\{x_1, x_2, \cdots, x_N\}$ e $f_m(\mathbf{x}_m)$ é uma função que possui os elementos de $\mathbf{x}_m$ como argumentos.

Definição 2.4 (Grafo fator) *Seja* $g(x_1, x_2, \cdots, x_N) = \prod_{m \in \mathcal{M}} f_m(\mathbf{x}_m)$ *uma função global. Um grafo fator é um grafo bipartite não-direcionado que expressa a estrutura de fatoração de uma função global. Um grafo fator possui um* ***nó de variável*** *para cada variável* x_n*, um* ***nó de função*** *para cada função local* f_m*, e um ramo conectando o nó de variável* x_n *ao nó de função* f_m *se e somente se* x_n *é um argumento de* f_m.

Exemplo 2.2 *Seja* $g(x_1, x_2, x_3, x_4, x_5)$ *uma função de cinco variáveis expressa como o produto*

$$g(x_1, x_2, x_3, x_4, x_5) = f_A(x_1) f_B(x_2) f_C(x_1, x_2, x_3) f_D(x_3, x_4) f_E(x_3, x_5).$$

De modo que $\mathcal{M} = \{A, B, C, D, E\}$, $\mathbf{x}_A = \{x_1\}$, $\mathbf{x}_B = \{x_2\}$, $\mathbf{x}_C = \{x_1, x_2, x_3\}$, $\mathbf{x}_D = \{x_3, x_4\}$ *e* $\mathbf{x}_E = \{x_3, x_5\}$. *O grafo fator que representa* $g(x_1, x_2, x_3, x_4, x_5)$ *é mostrado na Figura 2.4.*

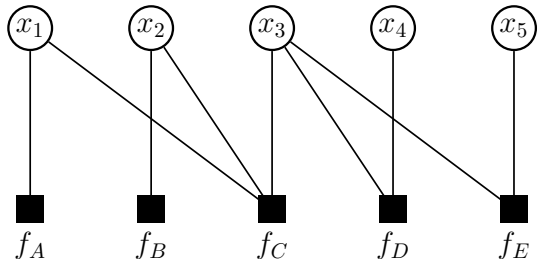


Figura 2.4: *Grafo fator para a função* $g(x_1, x_2, x_3, x_4, x_5)$.

2.3 Modelagem de sistemas usando grafos fator

A modelagem de sistemas utilizando grafos fator divide-se em duas classes: a modelagem *probabilística* e a modelagem *comportamental*. A modelagem probabilística, faz uso de um grafo fator para representar uma função densidade de probabilidade conjunta das variáveis presentes no sistema. A estrutura de fatoração desta função fornece informações importantes sobre as dependências estatísticas entre estas variáveis.

A modelagem comportamental, descreve o sistema a partir de configurações válidas de suas variáveis. Tais configurações são denominadas *comportamentos*. Nesta abordagem, um grafo fator é usado para representar a função indicadora (ou característica) para um dado comportamento. Fatorações desta função indicadora fornecem informações sobre a estrutura do modelo.

No contexto de codificação de canal, o sistema de comunicações é modelado pelos comportamentos válidos (o conjunto de palavras-código) e pela função densidade de probabilidade conjunta *a posteriori* das variáveis que definem as palavras-código dada a saída do canal. Portanto, trata-se de uma modelagem híbrida.

Se P é uma proposição booleana envolvendo um conjunto de variáveis, a função indicadora $\delta[P]$, que assume valores no conjunto $\{0, 1\}$, indicando a veracidade da proposição P, é definida da seguinte forma

$$\delta[P] \triangleq \begin{cases} 1, & \text{se } P \text{ é verdadeira;} \\ 0, & \text{caso contrário.} \end{cases}$$

Se $\wedge$ denota o operador lógico "E", uma importante propriedade da função indicadora é:

$$\delta[P_1 \wedge P_2 \wedge \cdots \wedge P_M] = \delta[P_1] \cdot \delta[P_2] \cdot \ldots \cdot \delta[P_M].$$

2.3.1 Modelagem comportamental

Sejam $x_1, x_2, \cdots, x_N$ um conjunto de variáveis com espaço de configurações $\mathcal{S} = \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_N$. Um *comportamento* em $\mathcal{S}$ é qualquer subconjunto $\mathcal{B} \subseteq \mathcal{S}$. Os elementos de $\mathcal{B}$ são as *configurações válidas*. O sistema é especificado através do seu comportamento $\mathcal{B}$ e por isso este processo é chamado de modelagem comportamental.

Na modelagem comportamental de um código, o domínio de cada variável é um alfabeto finito $\mathcal{A}$, o espaço de configurações é o produto Cartesiano $\mathcal{S} = \mathcal{A}^N$. Então o comportamento $\mathcal{C} \subseteq \mathcal{S}$ é chamado de *código de bloco* de comprimento N sobre $\mathcal{A}$, e as configurações válidas são *palavras-código*. Se $\mathcal{A} = \mathbb{F}_2$, o código $\mathcal{C}$ é binário.

A função característica para o código $\mathcal{C}$ é definida como:

$$\chi_{\mathcal{C}}(x_1, x_2, \cdots, x_N) \triangleq \delta[(x_1, x_2, \cdots, x_N) \in \mathcal{C}].$$

É possível dar a $\chi_{\mathcal{C}}$ uma interpretação probabilística se considerarmos que $\chi_{\mathcal{C}}$ é proporcional a uma distribuição de probabilidade uniforme sobre as palavras-código (configurações válidas).

Em muitos casos, a presença de uma determinada configuração em um comportamento $\mathcal{B}$ pode ser determinada aplicando uma série de testes (verificações), cada uma delas envolvendo algum subconjunto das variáveis do sistema. Uma configuração é considerada válida apenas se ela passar em todos os testes. Ou seja, a proposição $(x_1, x_2, \cdots, x_n) \in \mathcal{B}$ pode ser escrita como a conjunção lógica de proposições mais simples, cada uma delas definindo um *comportamento local*. No caso em que o comportamento é um código, os vetores devem satisfazer diversas equações de verificação de paridade para que sejam considerados válidos. Então, $\chi_{\mathcal{C}}$ pode ser fatorada e representada por um grafo fator.

Exemplo 2.3 (Código de bloco) *A função característica para qualquer código linear definido por uma matriz de verificação de paridade* $\mathbf{H}$*, de dimensões* $M \times N$*, pode ser representada por um grafo fator que possui* N *nós de variável e* M *nós de função. Por exemplo, seja* $\mathcal{C}(6,3)$ *é um código linear binário definido pela matriz de verificação de paridade:*

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

então $\mathcal{C}(6,3)$ *é o conjunto de todos os vetores binários de comprimento* 6*, que satisfazem simultaneamente as três equações expressas na forma matricial* $\mathbf{x}\mathbf{H}^T = \mathbf{0}$*. A função característica para* $\mathcal{C}(6,3)$ *é*

$$\chi_{\mathcal{C}}(x_1, x_2, \cdots, x_6) = \delta[(x_1, x_2, \cdots, x_6) \in \mathcal{C}(6,3)].$$

Usando as equações de verificação de paridade obtidas a partir de $\mathbf{H}$*,* $\chi_{\mathcal{C}}$ *pode ser fatorada da seguinte maneira*

$$\begin{aligned} \chi_{\mathcal{C}}(x_1, x_2, \cdots, x_6) &= \delta[x_1 \oplus x_3 \oplus x_4 = 0] \cdot \delta[x_1 \oplus x_2 \oplus x_5 = 0] \cdot \delta[x_2 \oplus x_3 \oplus x_6 = 0]; \\ &= f_1(x_1, x_3, x_4) \cdot f_2(x_1, x_2, x_5) \cdot f_3(x_2, x_3, x_6). \end{aligned}$$

O grafo fator que representa a fatoração desta função característica é mostrado na Figura 2.5, na qual

$$\begin{aligned} f_1(x_1, x_3, x_4) &= \delta[x_1 \oplus x_3 \oplus x_4 = 0] \\ f_2(x_1, x_2, x_5) &= \delta[x_1 \oplus x_2 \oplus x_5 = 0] \\ f_3(x_2, x_3, x_6) &= \delta[x_2 \oplus x_3 \oplus x_6 = 0]. \end{aligned}$$

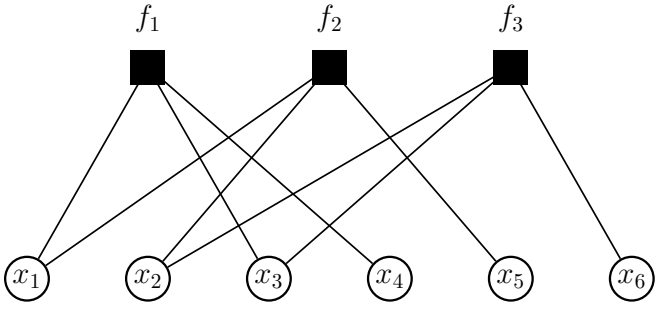


Figura 2.5: *Um grafo de Tanner para o código* $\mathcal{C}(6,3)$*.*

2.3.2 Modelagem probabilística

Uma outra classe de funções que podem ser representadas por grafos fatores são as funções densidade de probabilidade. Para ilustrar este caso, será considerado o modelo do sistema de comunicações no qual a palavra-código $\mathbf{x} = (x_1, x_2, \cdots, x_N)$ com comprimento N de um código $\mathcal{C}(N, K)$ é escolhida e transmitida através de um canal ruidoso produzindo a seqüência de saída $\mathbf{y} = (y_1, y_2, \cdots, y_N)$, em que $\mathbf{y} \in \mathbb{R}^N$, como ilustrado na Figura 2.6.

Figura 2.6: *Canal de comunicações ruidoso.*

Para cada $\mathbf{y}$ fixo, a distribuição de probabilidade *a posteriori* conjunta para as componentes do vetor aleatório $\mathbf{x}$ é

$$\begin{aligned} P(\mathbf{x}|\mathbf{y}) &= \frac{P(\mathbf{x},\mathbf{y})}{p(\mathbf{y})}; \\ &= \frac{p(\mathbf{y}|\mathbf{x})P(\mathbf{x})}{p(\mathbf{y})}; \\ &= \frac{p(\mathbf{y}|\mathbf{x})P(\mathbf{x})}{\sum_{\mathbf{x}} p(\mathbf{y}|\mathbf{x})P(\mathbf{x})}, \end{aligned}$$

em que $\sum_{\mathbf{x}} p(\mathbf{y}|\mathbf{x})P(\mathbf{x}) = p(\mathbf{y})$ é uma constante de normalização (pois $\mathbf{y}$ é fixo), $P(\mathbf{x})$ é a distribuição de probabilidade *a priori* sobre as palavras-código transmitidas e $p(\mathbf{y}|\mathbf{x})$ é a densidade de probabilidade condicional do vetor aleatório $\mathbf{y}$ quando $\mathbf{x}$ é transmitido.

Assumindo que a distribuição *a priori* é uniforme sobre as palavras do código $\mathcal{C}(N, K)$, *i.e.*,

$$P(\mathbf{x}) = \frac{\chi_{\mathcal{C}}(\mathbf{x})}{|\mathcal{C}|},$$

em que $\chi_{\mathcal{C}}$ é a função característica para $\mathcal{C}(N, K)$ e $|\mathcal{C}|$ é a cardinalidade do código $\mathcal{C}(N, K)$. Então

$$P(\mathbf{x}|\mathbf{y}) = \frac{\chi_{\mathcal{C}}(\mathbf{x})p(\mathbf{y}|\mathbf{x})}{|\mathcal{C}| \sum_{\mathbf{x}} p(\mathbf{y}|\mathbf{x})P(\mathbf{x})}.$$

Ou seja,

$$P(\mathbf{x}|\mathbf{y}) \propto \chi_{\mathcal{C}}(\mathbf{x})p(\mathbf{y}|\mathbf{x}).$$

Como o vetor $\mathbf{y}$ é fixo durante o processo de decodificação, a função $g(\mathbf{x}) = \chi_{\mathcal{C}}(\mathbf{x})p(\mathbf{y}|\mathbf{x})$ é uma função dos símbolos de $\mathbf{x}$ com as componentes do vetor $\mathbf{y}$ como parâmetros. Se o canal

é sem memória e sem realimentação, então $p(\mathbf{y}|\mathbf{x})$ é fatorável da seguinte forma:

$$p(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{N} p(y_i|x_i).$$

Logo, escreve-se

$$g(x_1, x_2, \cdots, x_N) = \chi_{\mathcal{C}}(x_1, x_2, \cdots, x_N) \prod_{i=1}^{N} p(y_i|x_i). \tag{2.2}$$

Como foi visto anteriormente, a função característica $\chi_{\mathcal{C}}(\mathbf{x})$ pode ser fatorada em funções características locais e a estrutura de fatoração da função global $g(\mathbf{x})$ pode ser representada por um grafo fator. O grafo fator da função $g(\mathbf{x})$ representa uma versão *escalonada* da distribuição *a posteriori* sobre $\mathbf{x}$.

Exemplo 2.4 (Distribuições *a posteriori*) *Seja $\chi_{\mathcal{C}}$ a função característica do código $\mathcal{C}(6,3)$ do Exemplo 2.3. Então,*

$$\begin{aligned} g(x_1, x_2, \cdots, x_6) &= \chi_{\mathcal{C}}(x_1, x_2, \cdots, x_6) \prod_{i=1}^{6} p(y_i|x_i); \\ &= \delta[(x_1, x_2, \cdots, x_6) \in \mathcal{C}] \prod_{i=1}^{6} p(y_i|x_i); \\ &= \delta[x_1 \oplus x_3 \oplus x_4 = 0] \cdot \delta[x_1 \oplus x_2 \oplus x_5 = 0] \cdot \delta[x_2 \oplus x_3 \oplus x_6 = 0] \cdot \prod_{i=1}^{6} p(y_i|x_i). \end{aligned}$$

Representa-se $g(x_1, x_2, \cdots, x_6)$ através do grafo fator da Figura 2.7.

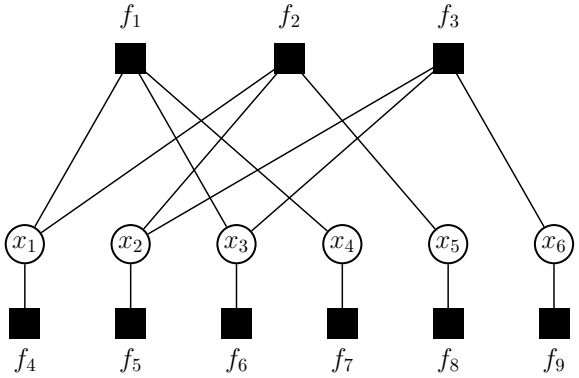


Figura 2.7: *Grafo fator para a distribuição a posteriori sobre* $\mathbf{x}$.

Na Figura 2.7, os nós de função são os seguintes:

$$\begin{aligned} f_1(x_1, x_3, x_4) &= \delta[x_1 \oplus x_3 \oplus x_4 = 0] \\ f_2(x_1, x_2, x_5) &= \delta[x_1 \oplus x_2 \oplus x_5 = 0] \\ f_3(x_2, x_3, x_6) &= \delta[x_2 \oplus x_3 \oplus x_6 = 0] \\ f_4(x_1) &= p(y_1|x_1) \\ f_5(x_2) &= p(y_2|x_2) \\ f_6(x_3) &= p(y_3|x_3) \\ f_7(x_4) &= p(y_4|x_4) \\ f_8(x_5) &= p(y_5|x_5) \\ f_9(x_6) &= p(y_6|x_6). \end{aligned}$$

Dado um grafo fator $\mathcal{G}'$ para $\chi_{\mathcal{C}}(\mathbf{x})$, é possível obter um grafo fator $\mathcal{G}$ que representa a distribuição *a posteriori* sobre $\mathbf{x}$ simplesmente acrescentando a $\mathcal{G}$ nós de função correspondendo aos termos $p(y_i|x_i)$, como mostrado na Figura 2.7 para o código $\mathcal{C}(6, 3)$ do Exemplo 2.3.

capítulo 3

O algoritmo Soma-Produto

A grande vantagem da modelagem de códigos por meio de grafos fator é permitir a utilização de algoritmos de decodificação iterativos cuja complexidade aumenta linearmente com o comprimento do código [25]. A decodificação é um problema de inferência estatística: dado um conjunto de observações, deseja-se inferir qual a palavra-código mais provável de ter sido transmitida, ou qual o valor mais provável de cada símbolo transmitido [26].

No processo de decodificação símbolo-a-símbolo de máxima probabilidade *a posteriori*, existe o interesse em obter distribuições de probabilidade marginais a partir de distribuições de probabilidade conjuntas (este procedimento será chamado de *marginalização*). No entanto, a marginalização de funções é um processo que geralmente possui um custo computacional elevado e cuja complexidade cresce exponencialmente com o número de variáveis N [26]. Portanto, é importante desenvolver algoritmos de marginalização eficientes que explorem a estrutura de fatoração da função global.

O algoritmo Soma-Produto (SP) se beneficia da estrutura da fatoração de uma determinada função global para calcular funções marginais. Isso é realizado através da *passagem de mensagens* através dos ramos do grafo fator que representa a função global. Quando o grafo fator não possui ciclos, o algoritmo calcula funções marginais exatas. Quando o grafo apresenta ciclos, o algoritmo computa apenas uma aproximação das funções marginais e funciona iterativamente [12].

Este capítulo, apresenta conceitos introduzidos em [12]. É assumido que $\mathbf{x}$ é um vetor

aleatório discreto de comprimento N, cujas componentes x_n, $1 \leq n \leq N$, são variáveis aleatórias binárias; e $\mathbf{y}$ é um vetor aleatório de comprimento N, cujas componentes y_n, $1 \leq n \leq N$, são variáveis aleatórias contínuas.

3.1 Descrição do algoritmo Soma-Produto

A decodificação símbolo-a-símbolo de máxima probabilidade *a posteriori*, requer que o valor mais provável para o símbolo x_n seja escolhido, dado um conjunto de observações $\mathbf{y}$. Portanto, é necessário calcular a função distribuição de probabilidade *a posteriori* $P(x_n|\mathbf{y})$ e então selecionar o valor de x_n que a maximiza, para $1 \leq n \leq N$. Supondo que $g(x_1, \cdots, x_N)$ é uma função global proporcional à função distribuição de probabilidade *a posteriori* conjunta $P(\mathbf{x}|\mathbf{y})$, a função distribuição de probabilidade *a posteriori* marginal $P(x_n|\mathbf{y})$ também será proporcional à marginalização de $g(x_1, \cdots, x_N)$ em relação à variável x_n. A função marginal $g_n(x_n)$ é definida a partir da função global da seguinte forma

$$g_n(x_n) \triangleq \sum_{x_1} \cdots \sum_{x_{n-1}} \sum_{x_{n+1}} \cdots \sum_{x_N} g(x_1, \cdots, x_N). \tag{3.1}$$

Estes somatórios múltiplos serão muito utilizados e para simplificar a notação, representa-se apenas as variáveis sobre as quais a função global *não é somada*. Define-se a *não-soma* em relação a x_n, que é representada por

$$g_n(x_n) \triangleq \sum_{\sim\{x_n\}} g(x_1, \cdots, x_N). \tag{3.2}$$

Exemplo 3.1 *Seja $g(x_1, x_2, x_3)$ uma função de três variáveis x_1, x_2 e x_3. Então, a não-soma em relação a x_2 é denotada por:*

$$g_2(x_2) = \sum_{\sim\{x_2\}} g(x_1, x_2, x_3) = \sum_{x_1} \sum_{x_3} g(x_1, x_2, x_3).$$

Durante o cálculo de $g_n(x_n)$ para todo $n \in \{1, \cdots, N\}$, existem vários termos em comum que podem ser re-utilizados para diminuir o número total de operações. O algoritmo SP é um procedimento computacionalmente eficiente utilizado para calcular as funções marginais de forma exata quando o grafo fator que representa $g(x_1, \cdots, x_N)$ não possui ciclos. Na presença de ciclos o algoritmo SP é limitado a calcular aproximações das funções marginais, sendo portanto um algoritmo *sub-ótimo*.

O algoritmo SP opera pela troca de mensagens através dos ramos de um grafo fator. Antes de descrever o algoritmo, é importante observar que as mensagens trocadas entre os nós em

cada passo do algoritmo são *funções*. Quando um ramo conecta um nó de variável x_n a um nó de função f_m, as mensagens podem ser enviadas em ambas direções ($x_n \rightarrow f_m$ ou $f_m \rightarrow x_n$). É possível imaginar que existe um processador associado a cada nó do grafo fator, os ramos do grafo fator representam canais de comunicação entre estes processadores e as mensagens trocadas contêm informação sobre as funções marginais. A cada passo, esta informação é recebida, processada e atualizada pelos nós. Depois de atualizada por um processador, a informação é transmitida para seus processadores adjacentes.

Independentemente da direção, uma mensagem passada através do ramo conectado à variável x_n é sempre uma função com domínio $\mathcal{A}_n$ (uma função de x_n). Por exemplo, seja x_n uma variável binária definida no alfabeto $\{0, 1\}$. Então, as mensagens passadas pelos ramos que estão conectados a x_n serão da forma $\mu(x_n)$. Tais funções podem ser representadas por vetores ou listas como $\big[\mu(0) \;\; \mu(1)\big]$, ou, se a escala não for importante, pela razão $\mu(0)/\mu(1)$ ou $\log\big[\mu(0)/\mu(1)\big]$.

Notação

Seja a função global $g(x_1, \cdots, x_N) = \prod_{m \in \mathcal{M}} f_m(\mathbf{x}_m)$ cuja estrutura de fatoração é representada por um grafo fator $\mathcal{G}$. Cada um dos fatores $f_m(\mathbf{x}_m)$ é uma função de um subconjunto $\mathbf{x}_m$ das variáveis que compõem $\mathbf{x} = \{x_1, \cdots, x_N\}$. O conjunto $\mathcal{N}(m)$ é composto pelos índices das variáveis em $\mathbf{x}_m$. De forma semelhante, $\mathcal{M}(n)$ representa o conjunto dos índices dos fatores dos quais a n-ésima variável é argumento. O conjunto $\mathcal{N}(m)\backslash n$ denota o conjunto $\mathcal{N}(m)$ excluindo o elemento n e $\mathcal{M}(n)\backslash m$ denota o conjunto $\mathcal{M}(n)$ excluindo o elemento m.

O algoritmo SP envolve mensagens de dois tipos: as *mensagens de nós de variável para nós de função*, denotadas por $q_{n\rightarrow m}$, e as *mensagens de nós de função para nós de variável*, denotadas por $r_{m\rightarrow n}$. Uma mensagem enviada através de um ramo que conecta o fator f_m à variável x_n é sempre uma função de x_n. O algoritmo SP é definido em termos de uma regra de iniciação, duas regras de atualização (uma para cada tipo de nó) e uma regra de terminação. Estas regras de atualização são definidas de modo a satisfazer o *princípio da informação extrínseca*, introduzido junto com os códigos Turbo.

Regra 3.1 (Princípio da informação extrínseca) *Uma mensagem enviada por um nó v_i através de um ramo e_k não pode depender de nenhuma mensagem previamente recebida pelo ramo e_k.*

As regras do algoritmo SP em sua forma genérica são descritas a seguir.

1. **Regra de iniciação** – Existem duas formas de iniciar o algoritmo. Se o grafo fator $\mathcal{G}$ não possuir ciclos, então este possui nós terminais denominados *folhas* e o algoritmo inicia por elas. Todas as folhas transmitem mensagens para seus respectivos vizinhos. Cada folha de variável x_n envia ao seu vizinho a função identidade $\mu(x_n) = 1$ e cada folha de função f_m envia a própria função $f_m(x_n)$ ao seu vizinho. Em suma,

   para uma folha de variável:

   $$q_{n\to n}(x_n) = 1;$$

   para uma folha de função:

   $$r_{m\to n}(x_n) = f_m(x_n).$$

   Caso $\mathcal{G}$ possua ciclos, o algoritmo pode ser iniciado fazendo com que todas as mensagens de nó de variável para nó de função iniciais sejam a função identidade. A troca de mensagens inicia por todos os nós de variável de forma simultânea, ou seja,

   para todo n, m**:**

   $$q_{n\to m}(x_n) = 1.$$

2. **Regra de atualização para nós de variável** – A mensagem $q_{n\to m}(x_n)$ enviada por um nó de variável x_n a um nó função f_m, onde $m \in \mathcal{M}(n)$ é dada por

   $$q_{n\to m}(x_n) = \prod_{m' \in \mathcal{M}(n)\setminus m} r_{m'\to n}(x_n). \tag{3.3}$$

   Ou seja, a mensagem enviada de um nó de variável x_n para um nó de função adjacente f_m é obtida pela multiplicação das mensagens recebidas por x_n dos seus vizinhos excluindo f_m.

3. **Regra de atualização para nós de função** – A mensagem $r_{m\to n}(x_n)$ enviada por um nó função f_m a um nó de variável x_n, onde $n \in \mathcal{N}(m)$ é dada por

   $$r_{m\to n}(x_n) = \sum_{\sim\{x_n\}} \left(f_m(\mathbf{x}_m) \cdot \prod_{n' \in \mathcal{N}(m)\setminus n} q_{n'\to m}(x_{n'}) \right). \tag{3.4}$$

   Ou seja, a mensagem enviada de um nó função f_m para um nó variável adjacente x_n é obtida pela multiplicação das mensagens recebidas dos seus vizinhos excluindo x_n com o próprio $f_m(\mathbf{x}_n)$ e pela não-soma em relação a x_n.

4. **Regra de terminação** – A função marginal (ou sua aproximação) em relação a x_n é obtida por

$$g_n(x_n) = \prod_{m \in \mathcal{M}(n)} r_{m \to n}(x_n).$$

Ou seja, a função marginal para x_n é obtida pela multiplicação de todas as mensagens recebidas pelo nó variável x_n.

No algoritmo SP, uma mensagem de saída de qualquer nó v_i só pode ser calculada e transmitida através do ramo e_k assim que todas as mensagens, exceto a que chega pelo ramo e_k, chegarem ao nó v_i. Isto ocorre para que não seja violado o princípio da informação extrínseca. É por esse motivo que em grafos sem ciclos a troca de mensagens é iniciada pelas folhas, já que estas não recebem mensagens de nenhum outro nó. No caso da presença de ciclos, o algoritmo força com que todos os nós de variável transmitam a função identidade, garantindo que todos os nós de função sejam capazes de calcular as mensagens de saída.

Em um grafo sem ciclos, o algoritmo SP termina uma vez que todos os nós de variável receberam uma mensagem de cada um de seus vizinhos. Neste caso, se o grafo $\mathcal{G}$ possui $|E|$ ramos, a condição de finalização é atingida em no máximo $2|E|$ passos [25]. Nos grafos com ciclos, o algoritmo não termina em um número finito de passos e procede de forma iterativa até que um critério de parada pré-estabelecido seja satisfeito.

Teorema 3.1 *Seja $g(x_1, x_2, \cdots, x_N)$ uma função global representada por um grafo fator finito sem ciclos $\mathcal{G}$. A função marginal para x_n, $g_n(x_n)$, calculada segundo o algoritmo Soma-Produto é dada por $g_n(x_n) = \prod_{m \in \mathcal{M}(m)} r_{m \to n}(x_n)$ e é exatamente a função marginalizada*

$$g_n(x_n) = \sum_{\sim\{x_n\}} g(x_1, x_2, \cdots, x_N).$$

O Teorema 3.1 garante a convergência do algoritmo SP quando o grafo fator não possui ciclos e sua prova é encontrada em [27]. No entanto, para grafos fator que possuem ciclos locais longos, o algoritmo SP pode ser aplicado para obter aproximações das funções marginais muito próximas dos valores exatos. A suposição de que os ciclos mínimos locais são longos, permite assumir independência entre as mensagens nas vizinhanças dos vértices do grafo. Isto faz com que o cálculo realizado pelo algoritmo SP seja exato pelo menos por um certo número de iterações, determinadas pelo comprimento do ciclo mínimo g. Para grafos de ciclo mínimo igual a g, é possível assumir independência entre as mensagens em até $g/2$ iterações [1].

Mesmo com esta limitação, o algoritmo SP pode ser usado para decodificar códigos corretores de erro. Neste caso, o objetivo é estimar os símbolos mais prováveis de terem sido transmitidos, ao invés de encontrar distribuições de probabilidade marginais exatas.

3.2 Passagem de mensagens em grafos com ciclos

As regras de atualização não dependem da presença de ciclos no grafo fator. Embora não seja possível garantir a convergência do algoritmo SP em grafos com ciclos, a aplicação das regras de atualização geralmente produz bons resultados [26].

A presença de ciclos no grafo resulta em uma propagação de mensagens indefinida, implicando em um algoritmo iterativo. A medida que as iterações são realizadas, as mensagens passam a perder sua interpretação como informação sobre as funções marginais desejadas. Mesmo nos casos em que o algoritmo converge, as funções resultantes geralmente não são as verdadeiras funções marginais.

Apesar deste problema, o algoritmo SP é capaz de produzir excelentes resultados na decodificação de códigos definidos em grafos com ciclos. Os códigos LDPC, são exemplos de códigos cujos grafos fator possuem ciclos e que são capazes de atingir um desempenho muito próximo do limite de Shannon para o canal RAGB quando decodificados através do algoritmo SP.

A passagem de mensagens ocorrem em tempo discreto, de acordo com um *cronograma* que determina quais são os ramos que estão ativos e em que direções as mensagens são passadas durante cada passo do algoritmo. Um exemplo é cronograma *invasivo*, que ativa cada ramo nas duas direções a cada iteração do algoritmo. Esta é o cronograma padrão para decodificação de códigos LDPC e é definida da seguinte forma:

Definição 3.1 (Cronograma invasivo) *No cronograma invasivo, cada iteração do algoritmo SP possui apenas dois passos. No primeiro passo,* ***todos*** *os nós de variável passam as mensagens para os nós de verificação. Em seguida,* ***todos*** *os nós de verificação passam as mensagens para os nós de variável.*

A terminação do algoritmo SP em um grafo fator com ciclos ocorre quando um número fixo de iterações é realizado ou quando o algoritmo resulta em decisões de símbolos que formam uma palavra-código.

3.3 Decodificação com o algoritmo Soma-Produto

Nesta seção, descreve-se como utilizar o algoritmo SP na decodificação de um código corretor de erros $\mathcal{C}(N, K)$ no domínio probabilístico, *i.e.*, as mensagens passadas são distribuições de probabilidade. Como a maioria dos códigos práticos possuem grafos de Tanner com ciclos, será assumido que o algoritmo SP funciona de forma iterativa, *i.e.*, sub-ótima.

Um decodificador baseado no algoritmo SP realiza a decodificação símbolo-a-símbolo com entradas e saídas suaves. Ele processa os símbolos recebidos iterativamente para aumentar a confiabilidade de cada símbolo decodificado. As medidas de confiabilidade de cada símbolo decodificado, calculadas ao final de cada iteração, são utilizadas como entrada para a próxima iteração [13].

A primeira etapa é representar a distribuição de probabilidade conjunta *a posteriori* dos símbolos das palavras-código transmitidas $\mathbf{x} = (x_1, x_2, \cdots, x_N)$ condicionadas à saída do canal $\mathbf{y} = (y_1, y_2, \cdots, y_N)$, *i.e.*, $P(\mathbf{x}|\mathbf{y})$. É importante lembrar que o vetor $\mathbf{y}$ assume valores em $\mathbb{R}^N$ e está fixo durante todo o processo de decodificação.

As mensagens enviadas pelo nó de variável x_n ao nó de verificação f_m serão denotadas por $q_{n\to m}(0)$ e $q_{n\to m}(1)$ e corresponde à probabilidade da variável x_n assumir os valores 0 e 1, respectivamente, baseada em todas as equações de verificação de paridade envolvendo x_n, exceto a m-ésima equação. O cálculo desta probabilidade é realizado a partir das mensagens recebidas provenientes de todos os nós de verificação envolvendo x_n, exceto f_m.

De maneira semelhante, $r_{m\to n}(0)$ e $r_{m\to n}(1)$ denotam as mensagens do nó de verificação f_m ao nó de variável x_n e também correspondem às probabilidades da variável x_n assumir os valores 0 e 1, respectivamente, baseada em todos os símbolos envolvidos na equação f_m exceto o n-ésimo símbolo. Esta probabilidade é calculada a partir das mensagens recebidas dos demais nós de variável conectados ao nó de verificação f_m, exceto x_n. Os passos do algoritmo SP para decodificação são apresentados a seguir.

1. **Iniciação** – O algoritmo SP inicia pelas folhas de função inferiores, passando as mensagens correspondentes às funções $p(y_n|x_n)$ para seus respectivos nós de variável x_n adjacentes. Então, todos nós de variável passam para seus vizinhos a mensagem

$$q_{n\to m}(0) = p(y_n|x_n = 0)$$

e

$$q_{n\to m}(1) = p(y_n|x_n = 1).$$

Estas probabilidades são calculadas a partir do modelo matemático do canal.

2. **Atualização para nós de verificação** – Cada nó de verificação f_m representa a função $f_m(\mathbf{x}_m) = \delta\big[\boxplus_{n\in\mathcal{N}(m)} x_n = 0\big]$. Se $x_n = i$, em que $i \in \{0, 1\}$, reescreve-se $f_m(\mathbf{x}_m|x_n = i) = \delta\big[\boxplus_{n'\in\mathcal{N}(m)\setminus n} x_{n'} = i\big]$.

Para $i = 0, 1$:

$$r_{m\to n}(i) = \sum_{\sim\{x_n\}} \delta\big[\boxplus_{n'\in\mathcal{N}(m)\setminus n} x_{n'} = i\big] \prod_{n'\in\mathcal{N}(m)\setminus n} q_{n'\to m}(x_{n'}) \tag{3.5}$$

A Figura 3.1 ilustra a passagem de mensagem entre o nó de verificação f_3 e o nó de variável x_7 na decodificação de um código de comprimento 8 com 4 equações de verificação de paridade.

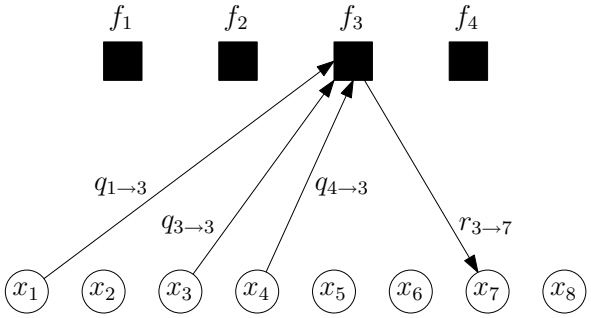


Figura 3.1: *Passagem de mensagem de um nó de verificação para um nó de variável.*

3. **Atualização para nós de variável** – Após a passagem das mensagens dos nós de verificação para os nós de variável, as mensagens $q_{n\to m}$ são atualizadas.

Para $i = 0, 1$:

$$q_{n\to m}(i) = \alpha_{n\to m} \cdot p(y_n|x_n = i) \cdot \prod_{m'\in\mathcal{M}(n)\setminus m} r_{m'\to n}(i), \tag{3.6}$$

em que a constante $\alpha_{n\to m}$ é escolhida de modo que $q_{n\to m}(0) + q_{n\to m}(1) = 1$. A Figura 3.2 ilustra a passagem de mensagem entre o nó de variável x_2 e o nó de verificação f_4 na decodificação de um código de comprimento 8 com 4 equações de verificação de paridade.

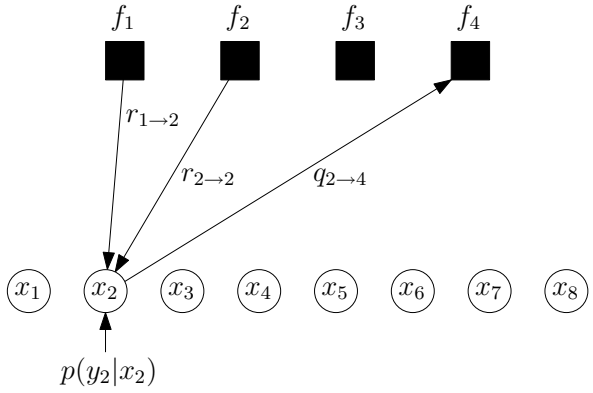


Figura 3.2: *Passagem de mensagem de um nó de variável para um nó de verificação.*

4. **Terminação** – A cada iteração, são calculadas as probabilidades *pseudo-posteriori*.

   Para $i = 0, 1$:

$$q_n(i) = \alpha_n \cdot p(y_n|x_n = i) \cdot \prod_{m\in\mathcal{M}(n)} r_{m\to n}(i). \tag{3.7}$$

   A constante α_n é escolhida de modo que $q_n(0)+q_n(1) = 1$. A partir de $q_n(1)$, $1 \leq n \leq N$, o decodificador estima o valor mais provável para cada símbolo transmitido. O valor estimado $\hat{x}_n$ para o símbolo x_n é dado por

$$\hat{x}_n \triangleq \begin{cases} 1, & \text{se } q_n(1) \geq 0.5; \\ 0, & \text{caso contrário.} \end{cases}$$

   Se o vetor estimado $\hat{\mathbf{x}}$ satisfizer a condição $\hat{\mathbf{x}}\mathbf{H}^T = \mathbf{0}$, o algoritmo encerra e retorna o vetor $\hat{\mathbf{x}}$ como palavra decodificada. Caso contrário, o algoritmo repete os passos 2 e 3 até que um número máximo de iterações (pré-definido) seja atingido. Neste caso, o algoritmo retorna a última estimativa $\hat{\mathbf{x}}$ como sendo o vetor mais provável de ter sido transmitido, mesmo que ele não pertença ao código $\mathcal{C}(N, K)$.

3.3.1 A regra da tangente hiperbólica

Como já foi comentado, para variáveis binárias, cada mensagem no algoritmo SP pode ser representada como o logaritmo da razão entre as duas componentes do vetor que a representa. Neste caso, é preciso derivar novas regras de atualização para o algoritmo. Esta

seção apresenta uma importante regra baseada na *álgebra de verossimilhança* [28] para obter a expressão da regra de atualização implementada pelos nós de verificação.

Seja U uma variável aleatória binária. O logaritmo da razão de verossimilhança (log-*likelihood ratio* ou LLR) de U é definido como

$$L(U) \triangleq \log\left(\frac{P_U(0)}{P_U(1)}\right),$$

no qual $P_U(u)$ denota a probabilidade de U assumir o valor u e o logaritmo é o logaritmo natural. O sinal do logaritmo da razão de verossimilhança $L(U)$ determina qual o valor mais provável que U pode assumir e é a partir dele que é realizada a decisão abrupta. A magnitude de $|L(U)|$ é a confiabilidade desta decisão. Se a variável aleatória binária U for condicionada a uma variável aleatória discreta V, então

$$\begin{aligned} L(U|V) &= \log\left(\frac{P_{U|V}(0|v)}{P_{U|V}(1|v)}\right); \\ &= \log\left(\frac{P_U(0)}{P_U(1)}\right) + \log\left(\frac{P_{V|U}(v|0)}{P_{V|U}(v|1)}\right); \\ &= L(U) + L(V|U). \end{aligned}$$

Se os valores assumidos pela variável aleatória U forem equiprováveis, $L(U) = 0$, implicando em $L(U|V) = L(V|U)$. É possível expressar a distribuição de probabilidade de U em função de $L(U)$:

$$P_U(0) = \frac{e^{L(U)}}{e^{L(U)} + 1},$$

e

$$P_U(1) = \frac{1}{e^{L(U)} + 1}.$$

O que implica em

$$P_U(0) - P_U(1) = \frac{e^{L(U)} - 1}{e^{L(U)} + 1} = \tanh\left(\frac{L(U)}{2}\right). \tag{3.8}$$

Supondo que U e V são variáveis aleatórias binárias estatisticamente independentes, o LLR da soma módulo 2 destas variáveis é calculado através de

$$\begin{aligned} P(U \oplus V = 0) &= P_{U,V}(0,0) + P_{U,V}(1,1) \\ &= P_U(0) \cdot P_V(0) + P_U(1) \cdot P_V(1), \end{aligned}$$

e

$$\begin{aligned} P(U \oplus V = 1) &= P_{U,V}(0,1) + P_{U,V}(1,0) \\ &= P_U(0) \cdot P_V(1) + P_U(1) \cdot P_V(0). \end{aligned}$$

Usando o mesmo argumento da Equação (3.8), obtém-se

$$\begin{aligned}\tanh\left(\frac{L(U\oplus V)}{2}\right) &= P(U\oplus V=0)-P(U\oplus V=1);\\ &= (P_U(0)-P_U(1))\cdot(P_V(0)-P_V(1));\\ &= \tanh\left(\frac{L(U)}{2}\right)\cdot\tanh\left(\frac{L(V)}{2}\right).\end{aligned}$$

Portanto,

$$L(U\oplus V)=2\cdot\tanh^{-1}\left(\tanh\left(\frac{L(U)}{2}\right)\cdot\tanh\left(\frac{L(V)}{2}\right)\right) \tag{3.9}$$

A Equação (3.9) é chamada de **regra da tangente hiperbólica**.

Uma simplificação prática segue do fato de que as funções $\tanh(x)$ e $\tanh^{-1}(x)$ são monotonicamente crescentes e possuem simetria ímpar, implicando em

$$\tanh(x)=\operatorname{sign}(x)\tanh\left(|x|\right),$$

e

$$\tanh^{-1}(x)=\operatorname{sign}(x)\tanh^{-1}\left(|x|\right).$$

Portanto, o sinal e a magnitude de $L(U\oplus V)$ são separáveis no sentido de que o sinal de $L(U\oplus V)$ depende apenas dos sinais de $L(U)$ e $L(V)$, e a magnitude $|L(U\oplus V)|$ depende apenas das magnitudes $|L(U)|$ e $|L(V)|$. Então, a Equação (3.9) pode ser reescrita como

$$L(U\oplus V)=[\operatorname{sign}(L(U))\cdot\operatorname{sign}(L(V))]\times 2\cdot\tanh^{-1}\left(\tanh\left(\frac{|L(U)|}{2}\right)\cdot\tanh\left(\frac{|L(V)|}{2}\right)\right). \tag{3.10}$$

A regra da tangente hiperbólica é usada pelo algoritmo SP no domínio LLR para calcular as atualizações das mensagens nos nós de verificação, onde a LLR da soma módulo 2 de variáveis aleatórias binárias são calculadas a partir das LLRs das variáveis sendo somadas.

3.3.2 Descrição do algoritmo

A mensagem enviada do nó x_n para o nó f_m é denotada por $L(q_{n\to m})$ e corresponde ao logaritmo da razão de verossimilhança da variável x_n. A mensagem enviada do nó f_m para o nó x_n é denotada por $L(r_{m\to n})$. Supondo que o grafo fator representa a fatoração de um distribuição de probabilidade *a posteriori*. O algoritmo SP no domínio LLR é descrito a seguir.

1. **Iniciação** – Cada folha de função $p(y_n|x_n)$ passa ao seu único vizinho um LLR inicial $L(y_n|x_n)=\log\big(p(y_n|x_n=0)/p(y_n|x_n=1)\big)$ para os nós de variável adjacentes. Então, todos nós de variável passam para seus vizinhos a mensagem

$$L(q_{n\to m})=L(y_n|x_n). \tag{3.11}$$

2. **Atualização para nós de verificação de paridade** – O m-ésimo nó de verificação de paridade recebe as mensagens $L(q_{n\to m})$, onde $n \in \mathcal{N}(m)$, e atualiza as mensagens $L(r_{m\to n})$ segundo a seguinte equação:

$$L(r_{m\to n}) = 2 \cdot \tanh^{-1}\left(\prod_{n'\in\mathcal{N}(m)\setminus n} \tanh\left(\frac{L(q_{n'\to m})}{2}\right)\right), \tag{3.12}$$

ou, alternativamente, por

$$L(r_{m\to n}) = \left(\prod_{n'\in\mathcal{N}(m)\setminus n} \text{sign}\left(L(q_{n'\to m})\right)\right) \times 2 \cdot \tanh^{-1}\left(\prod_{n'\in\mathcal{N}(m)\setminus n} \tanh\left(\frac{|L(q_{n'\to m}|}{2}\right)\right). \tag{3.13}$$

3. **Atualização para nós de variável** – O n-ésimo nó de variável recebe as mensagens $L(r_{m\to n})$, onde $m \in \mathcal{M}(n)$, e atualiza as mensagens $L(q_{n\to m})$ segundo a seguinte equação:

$$L(q_{n\to m}) = L(y_n|x_n) + \sum_{m'\in\mathcal{M}(n)\setminus m} L(r_{m'\to n}). \tag{3.14}$$

4. **Terminação** – O decodificador determina a informação *a posteriori* total sobre o símbolo n através da soma das mensagens transmitidas por todos os nós de verificação a ele conectados.

$$\Lambda_n = L(y_n|x_n) + \sum_{m\in\mathcal{M}(n)} L(r_{m\to n}). \tag{3.15}$$

O algoritmo itera até que uma palavra-código válida seja encontrada, *i.e.*, a decisão abrupta $\hat{\mathbf{x}}$ do vetor $\Lambda = (\Lambda_1, \Lambda_2, \cdots, \Lambda_N)$, em que

$$\hat{x}_n \triangleq \begin{cases} 0, & \text{se } \Lambda_n \geq 0; \\ 1, & \text{caso contrário,} \end{cases}$$

satisfaça a condição $\hat{\mathbf{x}}\mathbf{H}^T = \mathbf{0}$, ou um número máximo de iterações pré-definido é alcançado.

3.3.3 O algoritmo MIN-SUM

Apesar do algoritmo SP no domínio LLR eliminar algumas operações de multiplicação e a necessidade de normalizar as mensagens, o cálculo (computacionalmente dispendioso) de tangentes hiperbólicas e tangentes hiperbólicas inversas é necessário. No entanto, é possível aproximar a Equação (3.12) do algoritmo SP no domínio LLR, reduzindo a zero sua complexidade multiplicativa e eliminando o cálculo de tangentes hiperbólicas [29]. Para obter a

aproximação é preciso manipular a regra da tangente hiperbólica escrevendo a Equação (3.9) da seguinte forma:

$$\tanh\left(\frac{L(U\oplus V)}{2}\right)=\tanh\left(\frac{L(U)}{2}\right)\cdot\tanh\left(\frac{L(V)}{2}\right)$$

e usando a relação obtida em (3.8), obtém-se

$$\begin{aligned}\frac{e^{L(U\oplus V)}-1}{e^{L(U\oplus V)}+1} &= \left(\frac{e^{L(U)}-1}{e^{L(U)}+1}\right)\cdot\left(\frac{e^{L(V)}-1}{e^{L(V)}+1}\right)\\ &= \frac{e^{L(U)+L(V)}-e^{L(U)}-e^{L(V)}+1}{e^{L(U)+L(V)}+e^{L(U)}+e^{L(V)}+1}\end{aligned}\tag{3.16}$$

Resolvendo (3.16) para $L(U\oplus V)$, obtém-se a seguinte Equação:

$$\begin{aligned}L(U\oplus V) &= \log\left(\frac{e^{L(U)+L(V)}+1}{e^{L(U)}+e^{L(V)}}\right)\\ &= \log\left(e^{L(U)+L(V)}+1\right)-\log\left(e^{L(U)}+e^{L(V)}\right).\end{aligned}\tag{3.17}$$

Usando o logaritmo Jacobiano[1] nos dois termos do lado direito da igualdade da Equação (3.17)

$$\begin{aligned}L(U\oplus V) &= \max\left(0,L(U)+L(V)\right)+\log\left(1+e^{-|L(U)+L(V)|}\right)-\\ &\quad-\max\left(L(U),L(V)\right)-\log\left(1+e^{-|L(U)-L(V)|}\right).\end{aligned}\tag{3.18}$$

É possível mostrar que a Equação (3.18) é igual a

$$\begin{aligned}L(U\oplus V) &= \operatorname{sign}\left(L(U)\right)\operatorname{sign}\left(L(V)\right)\min\left(|L(U)|,|L(V)|\right)+\\ &\quad+\log\left(1+e^{-|L(U)+L(V)|}\right)-\log\left(1+e^{-|L(U)-L(V)|}\right).\end{aligned}\tag{3.19}$$

Desprezando os logaritmos, a Equação (3.19) é aproximada por

$$L(U\oplus V)\approx\operatorname{sign}\left(L(U)\right)\operatorname{sign}\left(L(V)\right)\min\left(|L(U)|,|L(V)|\right).\tag{3.20}$$

Esta aproximação é conhecida como a aproximação MAX-LOG. Se esta aproximação for utilizada, o algoritmo SP passa a se chamar algoritmo MIN-SUM. Este algoritmo realiza apenas multiplicações triviais (multiplicações por 1 e -1) e portanto, sua complexidade computacional multiplicativa é nula. A regra de atualização para os nós de verificação neste caso será dada por

$$\tilde{L}(r_{m\to n})\approx\left(\prod_{n'\in\mathcal{N}(m)\backslash n}\operatorname{sign}\left(L(q_{n'\to m})\right)\right)\times\min_{n'\in\mathcal{N}(m)\backslash n}\left(|L(q_{n'\to m})|\right)\tag{3.21}$$

[1] O logaritmo Jacobiano é definido como $\operatorname{jaclog}(x,y)\triangleq\log\left(e^{x}+e^{y}\right)=\max(x,y)+\log\left(1+e^{-|x-y|}\right)$.

A aproximação de uma das equações de atualização resulta em uma degradação de aproximadamente 0.5 dB no desempenho do algoritmo de decodificação [27]. Apesar disso, o algoritmo MIN-SUM é muito utilizado por apresentar um custo computacional muito abaixo do algoritmo SP baseado na regra da tangente hiperbólica, resultando em uma menor latência do decodificador.

capítulo 4

Códigos com matrizes de verificação de paridade esparsas

Os códigos LDPC [2] foram introduzidos por R. G. Gallager no início dos anos 60 e, assim como os códigos Turbo, são capazes de atingir um desempenho próximo da capacidade em diversos modelos de canais. Na época em que foram descobertos, os computadores não eram capazes de simular o desempenho de códigos com comprimentos significativos e a baixas taxas de erro. Isso fez com que os códigos LDPC passassem despercebidos pelos pesquisadores da área de códigos por muitos anos. Em 1981, R. M. Tanner generalizou o trabalho de Gallager [11] e introduziu a representação gráfica de códigos LDPC através de grafos bipartite. Novamente, os códigos LDPC caíram no esquecimento até que, em meados da década de 90, após o advento dos códigos Turbo e da decodificação iterativa, D. J. C. Mackay os redescobriu. Mackay mostrou que códigos LDPC longos, quando decodificados com o algoritmo Soma-Produto, são capazes de atingir um desempenho muito próximo ao limite de Shannon do canal RAGB [8]. A partir desta descoberta, os códigos LDPC têm sido intensamente pesquisados e utilizados para controle de erros em um grande número de sistemas de comunicação e armazenamento de dados.

Este capítulo é uma introdução aos códigos LDPC. Primeiro, a estrutura e a representação gráfica destes códigos é discutida. Em seguida, os métodos de Gallager, Mackay e *Progressive Edge Growth* para construção de códigos são apresentados. Por fim, são comparados os desempenhos de códigos construídos através destas três técnicas no canal RAGB.

4.1 Estrutura e representação gráfica

Um código de bloco linear $\mathcal{C}(N, K)$ é determinado por uma matriz geradora $\mathbf{G}$ ou uma matriz de verificação de paridade $\mathbf{H}$. Se o código for especificado por sua matriz $\mathbf{H}$, o código $\mathcal{C}(N, K)$ é o espaço nulo de $\mathbf{H}$, *i.e.*, uma N-upla binária $\mathbf{x} = (x_1, x_2, \cdots, x_N)$ é uma palavra-código se e somente se $\mathbf{x}\mathbf{H}^T = \mathbf{0}$. Um código LDPC é um código de bloco linear cuja matriz de verificação de paridade $\mathbf{H}$ é esparsa, ou seja, o número de elementos não-nulos é muito menor do que o número total de elementos da matriz.

Definição 4.1 (Códigos LDPC) *Um código LDPC é definido como o espaço nulo de uma matriz de verificação de paridade* $\mathbf{H}$ *de dimensões* $M \times N$ *que possui as seguintes propriedades: 1) cada linha possui peso constante* d_c*; 2) cada coluna possui peso constante* d_s*; e 3)* d_s *e* d_c *são pequenos quando comparados com o comprimento* N *do código e com o número de linhas* M *em* $\mathbf{H}$*, respectivamente.*

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* de comprimento N. A densidade da matriz $\mathbf{H}$ é definida como a razão entre o número de 1's e o número total de elementos na matriz $\mathbf{H}$. Para um código LDPC (d_s, d_c)-regular, o número de elementos não-nulos em $\mathbf{H}$ é

$$M \cdot d_c = N \cdot d_s.$$

Como o número total de elementos na matriz é $M \cdot N$, a densidade de $\mathbf{H}$, denotada por r, é dada por

$$r = \frac{M \cdot d_c}{M \cdot N} = \frac{d_c}{N} = \frac{N \cdot d_s}{M \cdot N} = \frac{d_s}{M}.$$

Geralmente, as técnicas de projeto de códigos LDPC não garantem que a matriz $\mathbf{H}$ irá possuir apenas linhas linearmente independentes. Ou seja, normalmente para um código LDPC

$$\text{POSTO}(\mathbf{H}) \leq M.$$

Portanto, o número de símbolos de verificação de paridade satisfaz $N - K \leq M$, com igualdade apenas quando todas a linhas de $\mathbf{H}$ forem linearmente independentes. É possível eliminar as linhas linearmente dependentes para encontrar uma matriz de verificação de paridade $N - K \times N$ de posto completo, entretanto, a matriz resultante provavelmente não possuiria

colunas e linhas de peso constante. Normalmente, as possíveis dependências entre as linhas de $\mathbf{H}$ são ignoradas e assume-se que a taxa do código é igual a

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

O grafo fator de um código LDPC (d_s, d_c)-regular é constituído de N nós de variável de grau d_s e M nós de verificação de grau d_c. Uma vez que o grau dos nós são determinados, ainda é possível escolher quais conexões em particular serão realizadas. O par (d_s, d_c), juntamente com o comprimento N do código, especifica um *conjunto* de códigos LDPC.

Exemplo 4.1 (Código LDPC regular) *Para ilustrar a estrutura dos códigos LDPC, a matriz de verificação e o grafo fator de um código LDPC* $(3, 4)$*-regular de comprimento* 12 *são mostrados abaixo.*

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

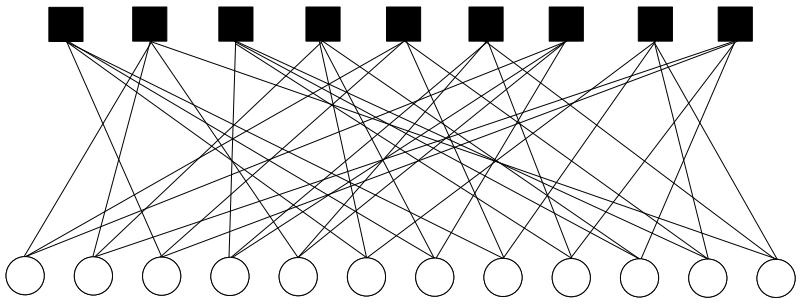


Figura 4.1: *Grafo fator para um código LDPC* $(3, 4)$*-regular.*

Este código possui taxa $R = 1 - 3/4 = 1/4$ *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 de *irregulares*. Os

códigos LDPC irregulares também possuem matrizes de verificação de paridade esparsas e são representados por grafos nos quais os nós de verificação e de variável 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

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

em que os coeficientes são números reais não-negativos, de modo que $\gamma(1) = 1$. Dado um par $\big(\lambda(x), \rho(x)\big)$ de distribuições de grau e um número natural N, define-se o conjunto de códigos LDPC $\big(\lambda(x), \rho(x)\big)$-irregulares de comprimento N, no qual

$$\lambda(x) = \sum_i \lambda_i x^{i-1}$$

é a distribuição de grau dos nós de variável, e

$$\rho(x) = \sum_i \rho_i x^{i-1}.$$

é a distribuição de grau dos nós de verificação. Os coeficientes de λ_i (ou ρ_i) denotam a fração de ramos no grafo que estão conectados a nós de variável (ou verificação) de grau i. A taxa de um código LDPC $\big(\lambda(x), \rho(x)\big)$-irregular é dada pela seguinte expressão

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

Exemplo 4.2 (Código LDPC irregular) *Seja o conjunto de códigos LDPC* $\big(\lambda(x), \rho(x)\big)$*-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 *nós de variável de grau* 2*,* 6 *nós de variável de grau* 3*,* 6 *nós de verificação de grau* 3 *e* 3 *nós de verificação de grau* 4*. Uma matriz de verificação de paridade e seu respectivo grafo fator para este código são mostrados a seguir.*

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

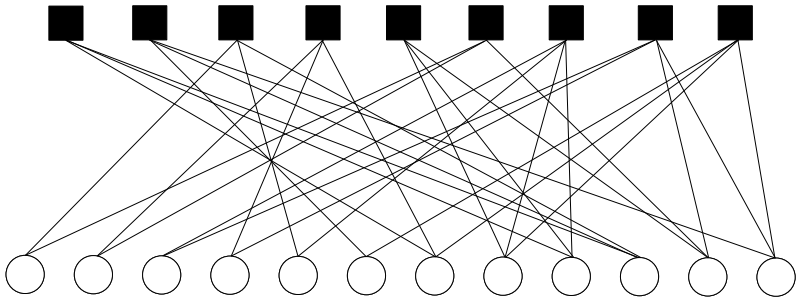


Figura 4.2: *Grafo fator para um código LDPC* $\big(\lambda(x), \rho(x)\big)$*-irregular.*

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

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

4.2 Projeto de códigos LDPC

O projeto de um código LDPC pode ser feito tanto pela de matriz de verificação de paridade como pelo grafo fator que esta matriz irá representar. No projeto de códigos LDPC é desejável que ciclos de comprimento curto sejam evitados, especialmente os de comprimento 4 [13]. Os ciclos de comprimento 4 em uma matriz $\mathbf{H}$ se manifestam como quatro 1's nos cantos de qualquer submatriz de $\mathbf{H}$. Alguns critérios de projeto são: 1) obter um desempenho próximo da capacidade; 2) obter um desempenho com baixos patamares de erro; e 3) obter uma estrutura que permite uma codificação eficiente. Nesta seção são descritas algumas técnicas de construção de códigos LDPC.

4.2.1 Códigos de Gallager

Esta técnica foi originalmente proposta por Gallager e foi o primeiro método utilizado para obter matrizes de códigos LDPC (d_s, d_c)-regulares. Um código construído da forma apresentada a seguir é denominado código LDPC de Gallager. Para construir a matriz de verificação de paridade $\mathbf{H}_{GA}$ de um código de Gallager é preciso primeiro construir uma submatriz $\mathbf{H}_1$ de dimensões $k \times k \cdot d_c$, 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 $N = k \cdot d_c$.

$$\mathbf{H}_1 = \begin{bmatrix} \underbrace{1\ 1\ 1 \cdots 1}_{d_c} & & & & \\ & \underbrace{1\ 1\ 1 \cdots 1}_{d_c} & & & \\ & & \ddots & & \\ & & & \overbrace{1\ 1\ 1 \cdots 1}^{d_c} & \\ & & & & \overbrace{1\ 1\ 1 \cdots 1}^{d_c} \end{bmatrix}.$$

Em seguida, são realizadas $d_s - 1$ permutações das colunas de $\mathbf{H}_1$ para formar outras $d_s - 1$ submatrizes $\mathbf{H}_2, \mathbf{H}_3, \cdots, \mathbf{H}_{d_s}$ de dimensões $k \times k \cdot d_c$. 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 $k \gg 1$.

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 grafo fator não possua ciclos curtos, especialmente ciclos de comprimento 4. Gallager 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 ou ciclo mínimo, porém, para comprimentos longos é muito provável que um código construído desta forma possua um bom desempenho.

Exemplo 4.3 (Código de Gallager) *A matriz* $\mathbf{H}_{GA}$ *mostrada em (4.1) foi construída a partir da técnica de Gallager, com os parâmetros* $N = 20$, $d_s = 3$ *e* $d_c = 4$.

$$\mathbf{H}_{GA} = \left[\begin{array}{cccccccccccccccccccc}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\hline
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
\hline
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{array}\right] \tag{4.1}$$

4.2.2 Códigos de Mackay

Mackay foi o primeiro pesquisador a demonstrar que os códigos baseados em matrizes esparsas eram capazes de atingir um desempenho próximo da capacidade do canal RAGB. Mackay desenvolveu algumas técnicas de construção *semi-aleatórias* que visam obter códigos cujos grafos fator não apresentem ciclos curtos [1]. Ele também fez uso de regras de otimização *ad-hoc* que empiricamente resultaram em códigos com desempenho excelente no canal RAGB. Os códigos de Mackay são considerados alguns dos melhores códigos LDPC existentes para comprimentos curtos. Existe um arquivo de códigos LDPC disponibilizados por Mackay [30].

Construção 1A

Uma matriz $\mathbf{H}$ de dimensões $M \times N$ é criada aleatoriamente com colunas de peso $d_s = 3$ e com linhas de peso *quase* constante $d_c = 6$, de modo que quaisquer duas colunas não apresentem sobreposição maior do que 1. Esta condição elimina os ciclos de comprimento 4 e é verificada quando o produto interno entre duas colunas de $\mathbf{H}$ é sempre menor ou igual a 1. O código resultante possui taxa $R = 1/2$. 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.

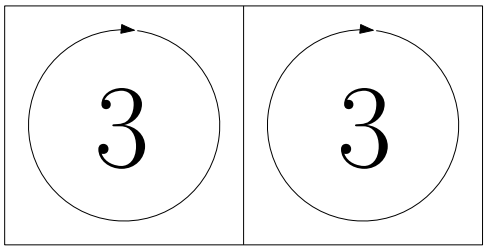


Figura 4.3: *Construção 1A para um código de Mackay* $(3, 6)$*-regular.*

Construção 2A

Uma matriz $\mathbf{H}$ de dimensões $M \times N$ é criada fazendo com que $M/2$ colunas sejam de peso 2. Entre estas $M/2$ colunas não deve existir nenhuma sobreposição. Uma forma simples de satisfazer esta condição é "empilhar" duas matrizes identidade de dimensões $M/2 \times M/2$. As demais colunas são construídas aleatoriamente com peso $d_s = 3$ mantendo o peso das linhas uniforme sempre que possível e evitando colunas com sobreposição maior do que 1. O código resultante possui taxa $R = 1/3$. A construção irregular usando colunas de peso 2 foi introduzida por apresentar bons resultados empíricos [1]. Em geral, a irregularidade do grafo, quando bem projetada, implica em uma melhora no desempenho em relação a códigos regulares de mesma taxa [26]. A Figura 4.4 ilustra a construção 2A. O número 3 denota a sobreposição de 3 *matrizes de permutação* e a seta circular representa permutações aleatórias de todas as colunas naquele bloco. As linhas diagonais representam matrizes identidade.

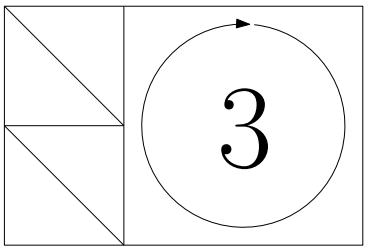


Figura 4.4: *Construção 2A para um código de Mackay irregular.*

Construção 1B e 2B

Nestas construções, algumas colunas das matrizes geradas pelas técnicas 1A e 2A são removidas para que o grafo fator do código não possua ciclos de comprimento menor do que um valor mínimo g.

Exemplo 4.4 (Códigos de Mackay) *Considere o código de Mackay* $(3, 6)$*-regular binário de comprimento* 96 *e taxa* $R = 1/2$ *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.*

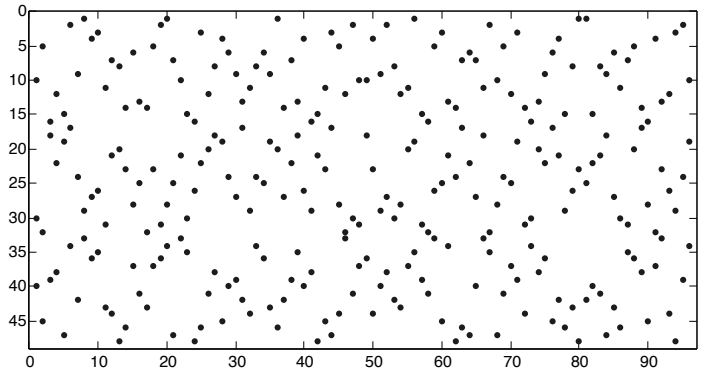


Figura 4.5: *Matriz* **H** *de um código de Mackay* $(3, 6)$*-regular.*

Um exemplo de matriz de verificação de paridade construída com o uso da técnica de construção 2A é mostrada na Figura 4.6. Este código possui comprimento 96 *e taxa* $R = 1/3$.

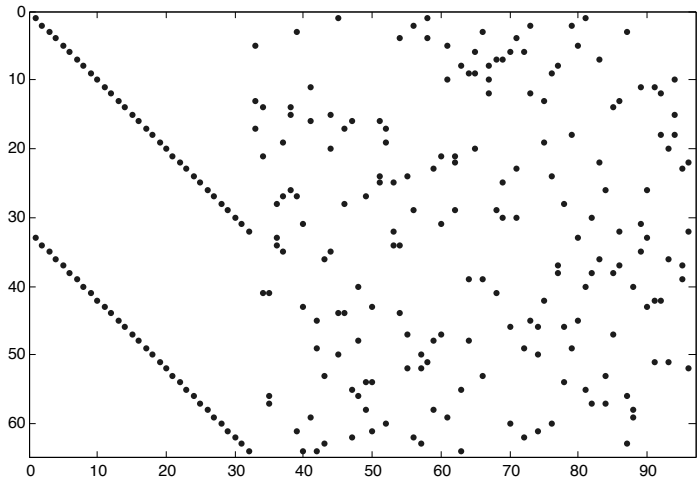


Figura 4.6: *Matriz* $\mathbf{H}$ *de um código de Mackay irregular.*

Uma desvantagem dos códigos de Mackay é a falta de uma estrutura que permita um codificador eficiente. Normalmente, sua codificação é realizada colocando a matriz $\mathbf{H}$ em forma sistemática $\left[\mathbf{P}^T|\mathbf{I}_{N-K}\right]$ através de eliminação Gaussiana. Este procedimento possui uma complexidade da ordem de $\mathcal{O}(N^3)$ e permite obter uma matriz geradora em forma sistemática $\mathbf{G} = \left[\mathbf{I}_K|\mathbf{P}\right]$. Geralmente a matriz geradora obtida desta forma não é esparsa e a codificação por multiplicação direta possui complexidade da ordem de $\mathcal{O}(N^2)$ [31].

4.2.3 O algoritmo Progressive Edge Growth

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 grafo de Tanner que o descreve. Para obter um bom código, é sempre desejável construir um grafo de Tanner que possua um ciclo mínimo grande. A construção de grafos de Tanner com o maior ciclo-mínimo possível é um difícil problema de difícil análise combinatorial. Entretanto, é possível projetar grafos de Tanner 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 nós de variável e de verificação em um grafo de Tanner de modo a maximizar os ciclos mínimos locais de cada um dos nós de variável [32].

Dado os parâmetros do grafo, *i.e.*, o número de nós de variável N, o número de nós de verificação M e a distribuição de grau dos nós de variável $\lambda(x)$, 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 nó de verificação, encontrar os nós de verificação mais distantes e então conectá-los através de ramos. Para encontrar os nós de verificação mais distantes de um determinado nó de variável x_n, é construído um subgrafo partindo de x_n e dos ramos nele incidentes $(x_n, c_{m_1}), (x_n, c_{m_2}), \cdots, (x_n, c_{m_{d(x_n)}})$. Então, os vértices $c_{m_1}, c_{m_2}, \cdots, c_{m_{d(x_n)}}$ são incluídos no subgrafo, juntamente com todos os ramos incidentes neles, excluindo $(x_n, c_{m_1}), (x_n, c_{m_2}), \cdots, (x_n, c_{m_{d(x_n)}})$. O processo continua até que uma determinada *profundidade* seja alcançada.

Neste procedimento, o subgrafo pode conter vários vértices e ramos iguais. Para um nó de variável x_n, a sua *vizinhança de profundidade* l é definida como o conjunto de nós de verificação alcançados pelo subgrafo de profundidade l expandido como mostrado na Figura 4.7 e é denotada por $\mathcal{N}^l(x_n)$. O conjunto complementar $\bar{\mathcal{N}}^l(x_n)$ é definido como $V_c \backslash \mathcal{N}^l(x_n)$. Na Figura 4.7, qualquer nó de variável que aparece pela primeira vez na profundidade l está a uma distância $2l$ em relação ao nó x_n e qualquer nó de verificação que aparece pela primeira vez na profundidade l está a uma distância de $2l + 1$ em relação ao nó x_n.

Ao expandir progressivamente o subgrafo a partir do nó de variável x_n duas situações podem ocorrer: 1) a cardinalidade do conjunto $\mathcal{N}^l(x_n)$ para de aumentar e é menor do que M; 2) $\mathcal{N}^l(x_n) \neq \varnothing$ e $\bar{\mathcal{N}}^{l+1}(x_n) = \varnothing$. No primeiro caso, nem todos os nós de verificação podem ser alcançados a partir de x_n, então conectar x_n a um dos elementos de $\bar{\mathcal{N}}^l(x_n)$ não cria nenhum ciclo adicional. No segundo caso, todos os nós de verificação são alcançados em uma profundidade $l + 1$, então a conexão de x_n a um elemento de $\bar{\mathcal{N}}^l(x_n)$ cria um ciclo de comprimento $2(l + 2)$.

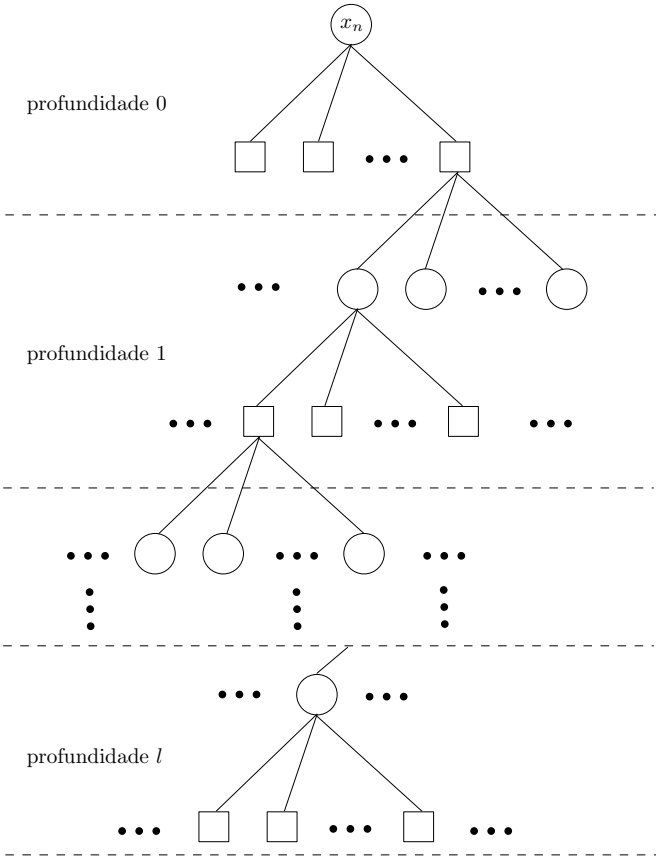


Figura 4.7: *Um subgrafo expandido a partir do nó x_n.*

Quando o algoritmo encontra mais de um nó de verificação para conectar ao nó de variável x_n, *i.e.*, $\bar{\mathcal{N}}^l(x_n)$ possui mais de um elemento, o nó com menor grau é selecionado. Tal procedimento faz com que a distribuição de graus dos nós de verificação 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 nó de verificação com menor o índice do conjunto.

Exemplo 4.5 (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 seguinte matriz de verificação de paridade.*

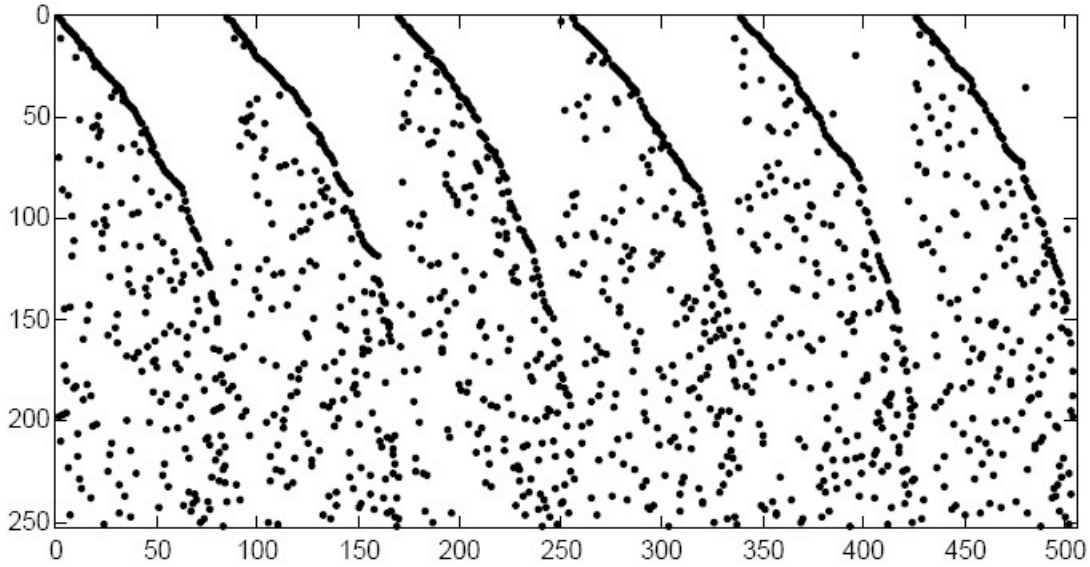


Figura 4.8: *Matriz* **H** *de um código LDPC construído com o algoritmo PEG.*

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 dos nós de verificação. No entanto, isto não influencia significativamente o seu desempenho [32].

4.3 Análise de desempenho

Nesta seção são analisados os desempenhos de diversos códigos LDPC decodificados com o algoritmo SP no domínio LLR, usando a cronograma invasivo. As palavras-código foram transmitidas através de um canal RAGB com densidade espectral de potência $N_0/2$ usando uma modulação BPSK com sinais de energia $E_s = 1$. Para calcular as taxas de erro de bit em função da razão E_b/N_0 ($E_b = E_s/R$ é a energia usada para transmitir um bit de informação), os erros foram contados ao longo de toda palavra-código e para obter cada ponto das curvas foram coletados pelo menos 100 erros de bloco. Nestas simulações foram utilizados computadores Pentium IV de 3Ghz com 1Gb de memória RAM.

A Figura 4.9 ilustra o desempenho do algoritmo SP na decodificação de um código de Mackay $(3, 6)$-regular de comprimento 96 construído através da técnica 1A[1] com um número máximo de 5, 10, 20, 40, 80 e 100 iterações.

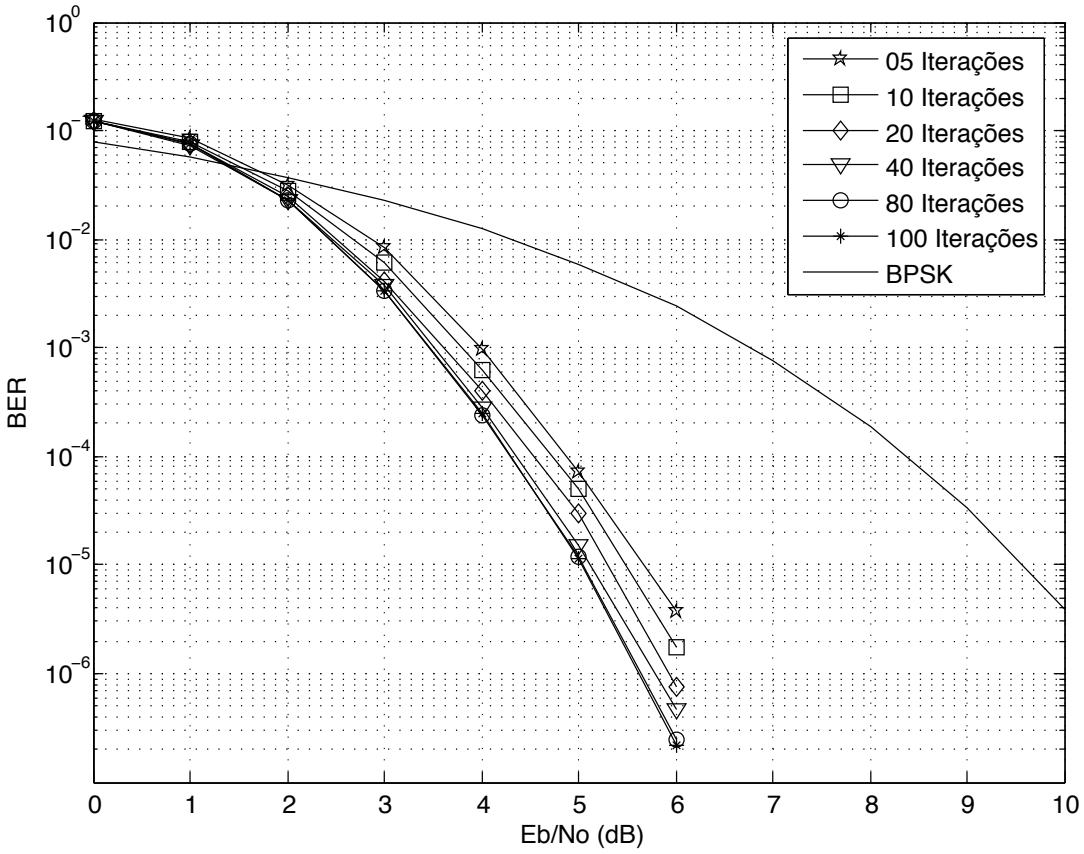


Figura 4.9: *Desempenho do código $\mathcal{C}(96, 48)$ para diversos números de iterações.*

[1]Este código é referido como `96.3.963` em [30].

É possível observar o ganho de desempenho à medida que o número de iterações aumenta. Segundo a Figura 4.9, existe um ganho de pouco mais de 0.5 dB ao aumentar o número de iterações de 5 para 80 para uma BER de 10^{-5}. Conclui-se que o aumento no número de iterações resulta em um aumento nas confiabilidades dos bits decodificados. Em contrapartida, a latência do decodificador também aumenta, como pode ser comprovado pela Tabela 4.1.

Tabela 4.1: *Tempo de simulação para obter cada uma das curvas da Figura 4.9.*

Iterações	Tempo de Simulação
05	560.953s
10	2449.406s
20	9507.406s
40	10023.328s
80	17815.532s
100	17917.281s

Uma observação interessante é a de que o ganho de desempenho ocorre até um certo número de iterações. A partir de 80 iterações o algoritmo não apresenta melhoria significativa de desempenho. O número ideal de iterações é escolhido de acordo com o comprimento do código e da distribuição de graus dos nós de variável. Um código cujo grafo de Tanner possui nós de variável de graus elevados apresentam uma convergência mais rápida do algoritmo SP. Para as demais simulações, o número máximo de iterações escolhido foi 80. Este valor foi utilizado em [32] para simulações de códigos PEG e Mackay de comprimento $N = 504$.

Utilizando a técnica de Gallager, foram contruídos códigos LDPC $(3, 6)$-regulares de comprimento 96, 204, 504 e 816. O desempenho destes códigos no canal RAGB é mostrado na Figura 4.10.

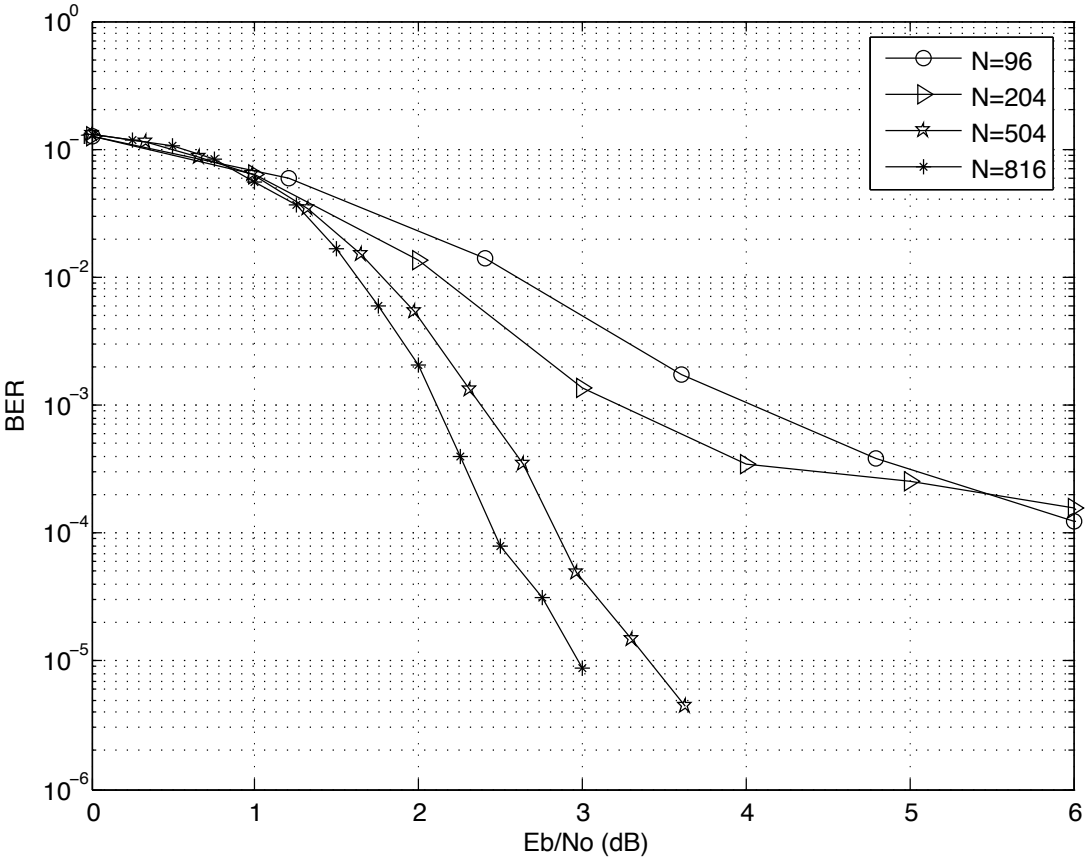


Figura 4.10: *Desempenho de códigos LDPC de Gallager para diversos comprimentos.*

Estes códigos foram gerados aleatoriamente sem evitar ciclos de comprimento 4 e, observa-se que os códigos de Gallager de comprimento 96 e 204 obtidos desta maneira apresentam patamar de erro em uma BER de 10^{-4}. Isto indica uma possível distância mínima baixa e um grande número de ciclos de comprimento 4. O mesmo não é observado para os códigos de comprimento 504 e 816, donde conclui-se que, para grandes comprimentos, os códigos construídos por este método possuem em média um bom desempenho mesmo sem as técnicas de otimização que evitam ciclos curtos.

A Figura 4.11 ilustra o desempenho de códigos de Mackay[2] $(3, 6)$-regulares de diversos comprimentos. É possível observar um princípio de patamar de erro na curva de desempenho do código de comprimento 204 em uma BER de aproximadamente 10^{-6}.

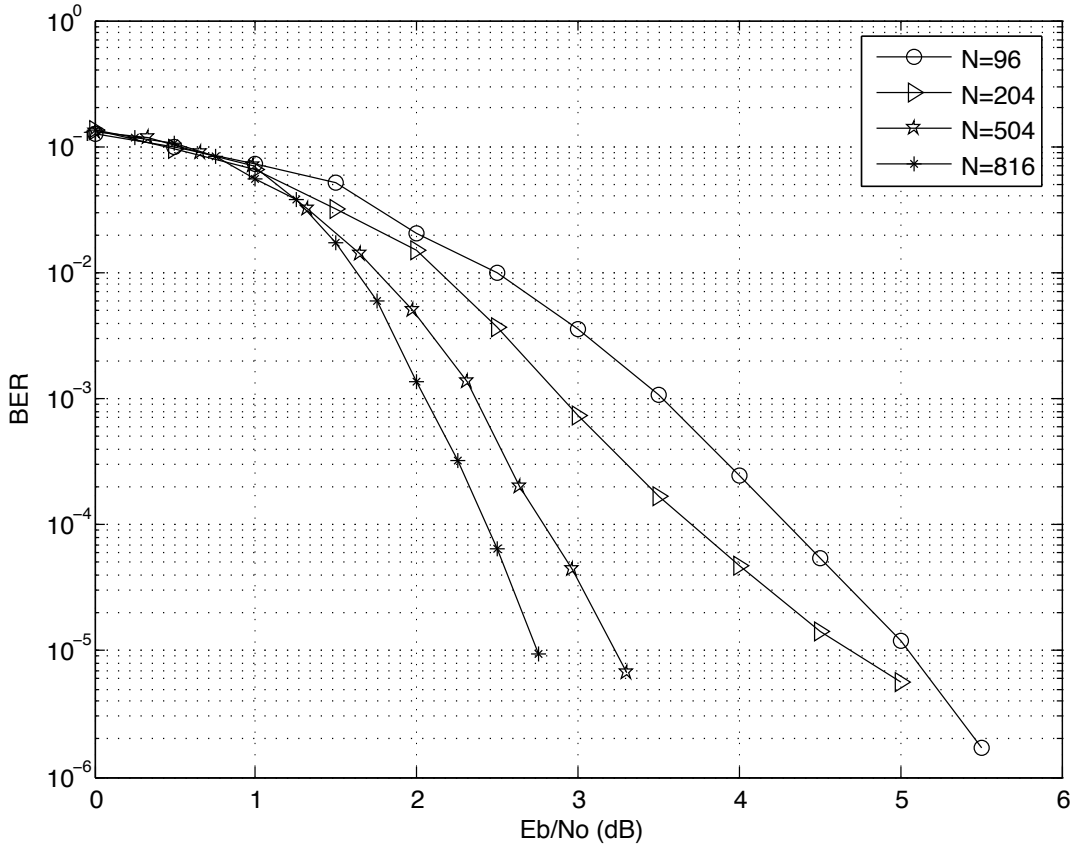


Figura 4.11: *Desempenho de códigos LDPC de Mackay para diversos comprimentos.*

[2]Códigos `96.3.963`, `204.33.486`, `252.252.3.252` e `816.3.174` disponíveis em [30].

Para analisar o desempenho dos códigos PEG, foram construídos códigos de comprimento 96, 204, 504 e 816 seguindo a regra de otimização para evitar ciclos curtos. Os códigos PEG, têm seu desempenho ilustrado na Figura 4.12 e nenhum deles apresenta patamar de erro na faixa de valores obtidas para a BER. Os códigos LDPC construídos com o algoritmo PEG apresentam um bom ciclo mínimo (tipicamente 6 ou 8). Um ciclo mínimo alto induz uma boa distância mínima do código, melhorando o desempenho e implicando em baixos patamares de erros [32].

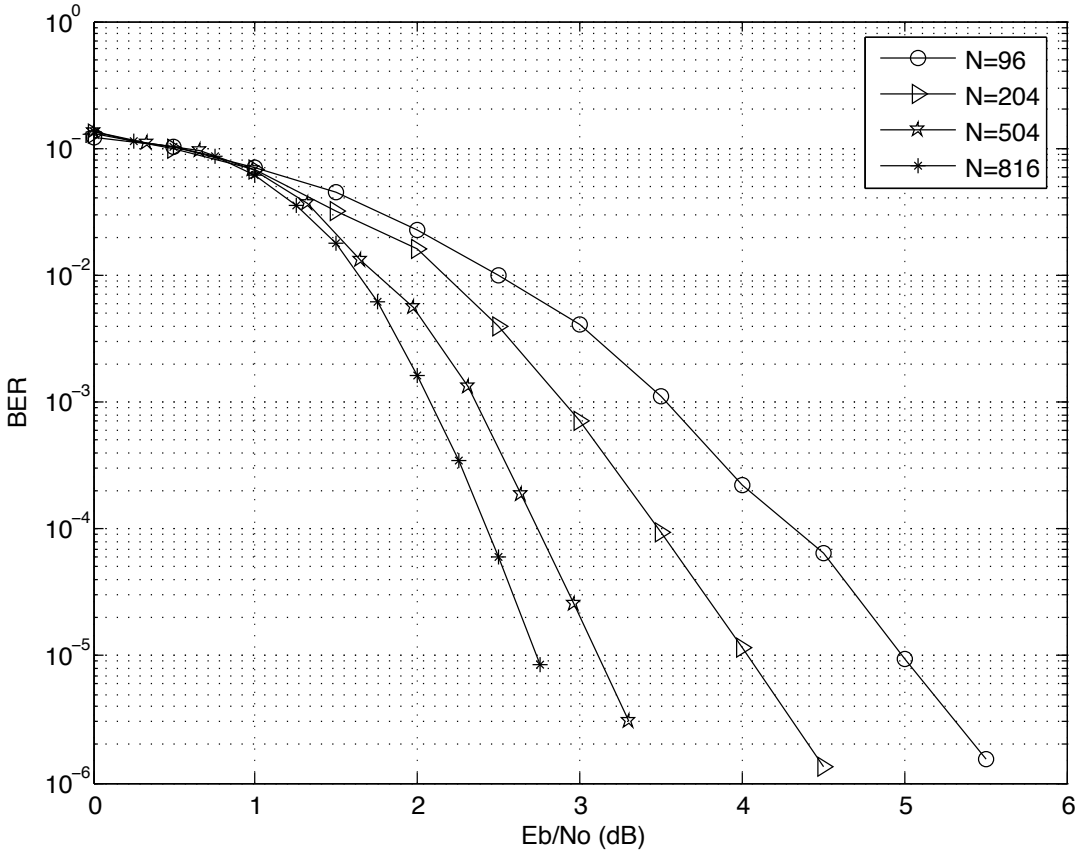


Figura 4.12: *Desempenho de códigos LDPC PEG para diversos comprimentos.*

A dependência do desempenho em relação aos graus dos nós dos grafos de Tanner é mostrada na Figura 4.13 para códigos $(3, 6)$, $(4, 8)$, $(5, 10)$ e $(6, 12)$-regulares de comprimento 504 construídos com o algoritmo PEG. Observa-se que há uma queda no desempenho com o aumento dos graus dos nós de variável. Isto pode ser explicado pela diminuição da distância mínima destes códigos, uma vez que aumentando o peso das colunas da matriz de verificação de paridade $\mathbf{H}$ é natural que sejam necessárias menos colunas cuja soma módulo dois, componente a componente, seja igual a $\mathbf{0}$. As distribuições de distância assintótica para diversos conjuntos de códigos LDPC determinadas por S. Litsyn e V. Shevelev [33] reforçam a validade desta hipótese para os códigos PEG.

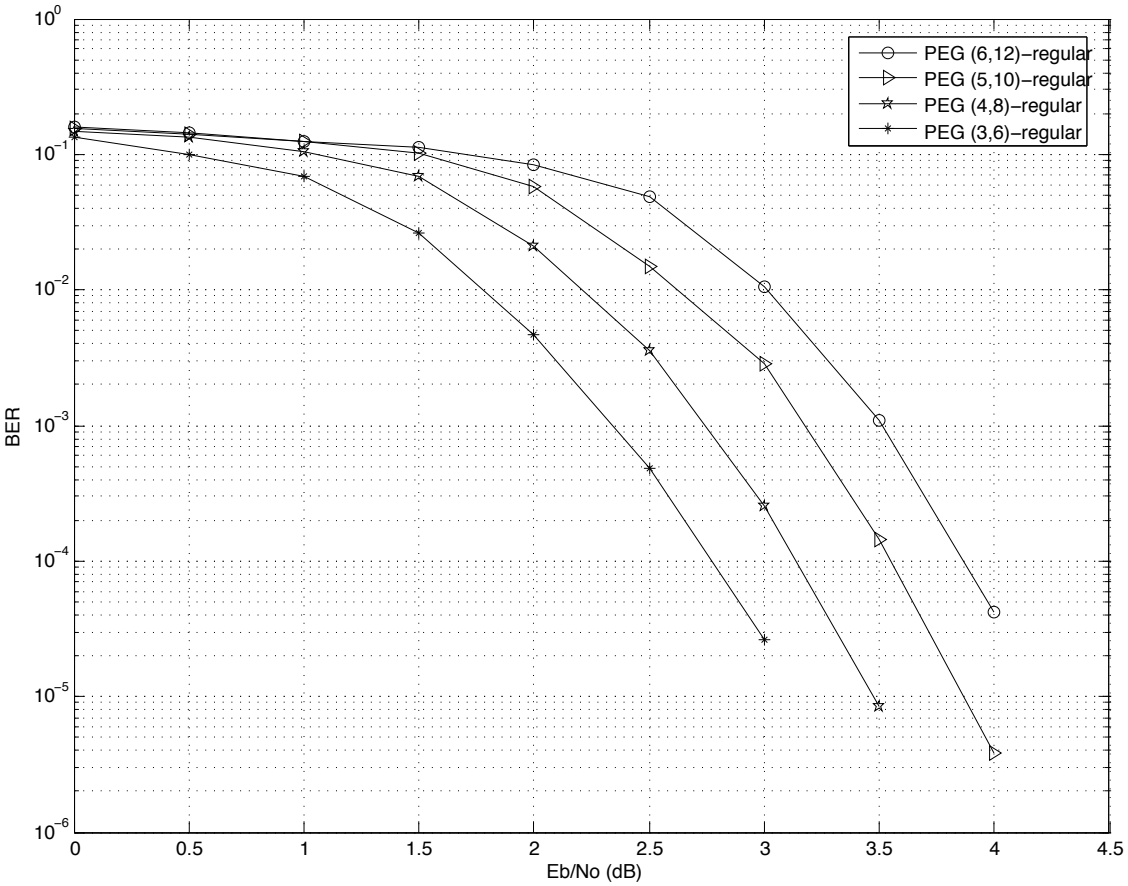


Figura 4.13: *Desempenho de códigos LDPC PEG regulares para diversos graus.*

O gráfico da Figura 4.14 mostra o desempenho dos códigos de Gallager, Mackay e PEG $(3, 6)$-regulares de comprimento 816. É possível observar que o desempenho dos 3 códigos é essencialmente o mesmo para razões E_b/N_0 menores que 2.5 dB. A partir deste ponto, o código de Gallager passa a apresentar uma queda no desempenho. Isto reforça o argumento de que os códigos em conjuntos de códigos LDPC (d_s, d_c)-regulares de comprimento N apresentam em média o mesmo desempenho, à medida que N aumenta.

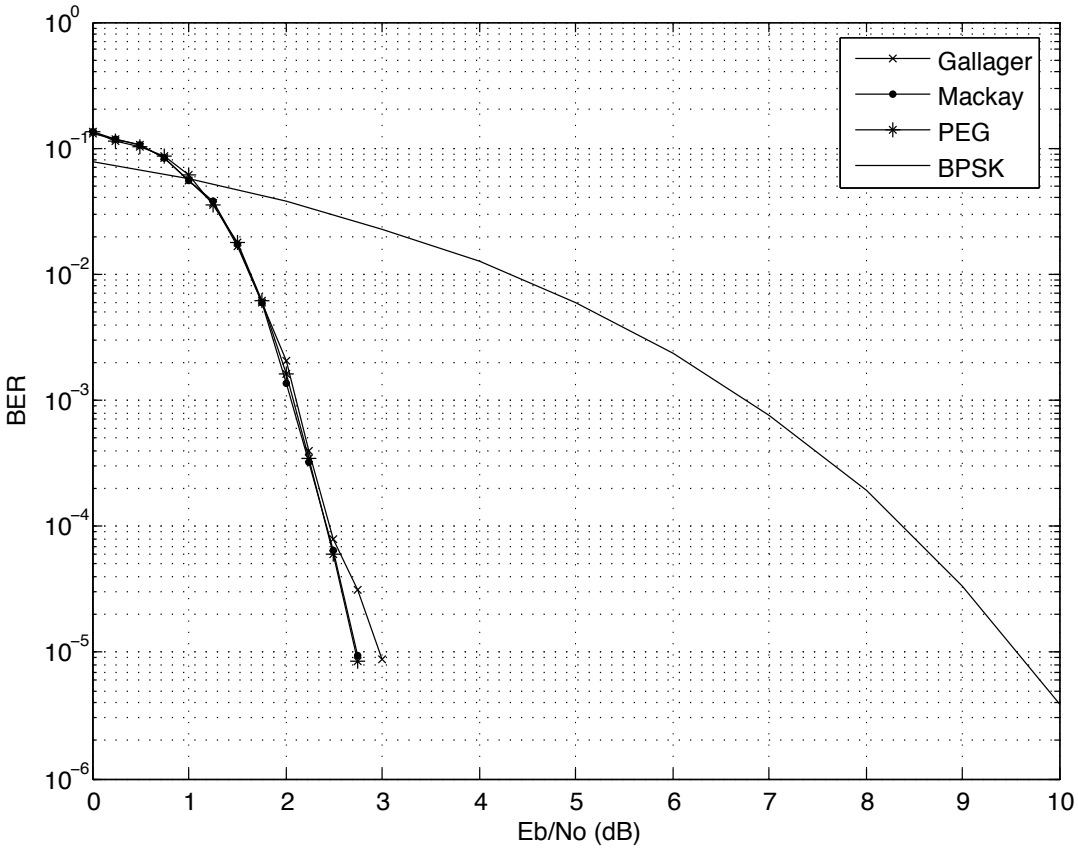


Figura 4.14: *Desempenho de códigos de comprimento 816 no canal RAGB.*

capítulo 5

Análise assintótica

A decodificação iterativa utiliza dois tipos de informação para estimar a palavra-código transmitida: a informação proveniente do canal (informação intrínseca) e a informação fornecida pelo código (informação extrínseca) [28]. A partir destas duas "fontes" de informação, o algoritmo de decodificação procura obter conhecimento sobre a palavra-código transmitida, usando-o como informação extrínseca para a próxima iteração. A decodificação é bem sucedida quando o conhecimento sobre a palavra-transmitida melhora a cada iteração.

Os métodos de análise de decodificadores iterativos procuram estudar a estatística das mensagens passadas através dos ramos do grafo fator a cada iteração. A *density evolution* [14] é o método mais famoso e completo de análise e determina a evolução das funções de densidade de probabilidade das mensagens iteração por iteração. Outros métodos de análise menos precisos acompanham a evolução de parâmetros das funções densidade como a média ou variância. Um exemplo deste método é a análise por aproximação Gaussiana [15].

Na análise assintótica do algoritmo SP a idéia principal é caracterizar a taxa de erro das mensagens em cada iteração baseando-se nas condições do canal e na taxa de erro das mensagens na iteração anterior. Este tipo de análise assume que códigos LDPC infinitamente longos são utilizados e os grafos fator que os representam não possuem ciclos. Neste caso, as mensagens que chegam a um nó de variável ou de verificação são independentes e o algoritmo funciona de forma ótima [34].

Neste capítulo, a *density evolution* é discutida e as suas equações são derivadas. Em seguida, a análise por aproximação Gaussiana é apresentada. Por fim, discute-se o cálculo de limiares de decodificação para alguns conjuntos de códigos LDPC (d_s, d_c)-regulares.

5.1 *Density Evolution*

Como visto no Capítulo 3, a cada iteração do algoritmo SP são atualizadas dois tipos de mensagens. As mensagens dos nós de símbolo para nós de verificação, denotadas por q; e as mensagens de nós de verificação para nós de símbolo, denotadas por r. A partir de agora, os sub-índices $n \rightarrow m$ e $m \rightarrow n$ serão omitidos e o algoritmo SP funcionará no domínio LLR. A abordagem seguida nesta seção incorpora elementos de [35], [19] e [14].

É importante relembrar que durante a atualização das mensagens, a mensagem que chega pelo ramo e_k é excluída do cálculo da mensagem que por ele será enviada. Isto é, apenas informação extrínseca circula pelo grafo fator. Esta é uma propriedade importante compartilhada por bons decodificadores por passagem de mensagem e que permite que o algoritmo de decodificação seja estudado analiticamente.

A análise assintótica de códigos LDPC assume que o grafo fator associado ao código não possui ciclos, ou seja, as mensagens trocadas entre os nós são independentes. Baseada nesta hipótese, a análise do algoritmo de decodificação calcula a média de mensagens incorretas que é passada a cada iteração.

Como os códigos LDPC são lineares, a N-upla $(0, 0, \cdots, 0)$ é uma palavra-código. Assumindo que a palavra-código *toda-zero* é transmitida, uma mensagem incorreta é toda mensagem cujo sinal no LLR é negativo. Utilizando uma modulação BPSK, o símbolo x, $x \in \{0, 1\}$, é mapeado no ponto $w = 1 - 2x$ da constelação de sinais e transmitido através de um canal RAGB. A saída do filtro casado y possui a seguinte função densidade de probabilidade (fdp) condicional:

$$p(y|w) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-w)^2}{2\sigma^2}\right), \tag{5.1}$$

em que $\sigma^2 = N_0/2$ é a variância do ruído.

A mensagem inicial processada pelo algoritmo SP, $L(y|x)$, é calculada da seguinte forma

$$\begin{aligned} L(y|x) &= \log\left(\frac{p(y|x=0)}{p(y|x=1)}\right); \\ &= \log\left(\exp\left(-\frac{(y-1)^2}{2\sigma^2} + \frac{(y+1)^2}{2\sigma^2}\right)\right); \\ &= \frac{-(y-1)^2 + (y+1)^2}{2\sigma^2}; \\ &= \frac{-(y^2 - 2y + 1) + (y^2 + 2y + 1)}{2\sigma^2}; \\ &= \frac{2}{\sigma^2} y. \end{aligned} \tag{5.2}$$

A partir de agora esta mensagem será denotada por q_0. Como assume-se que a palavra-código toda-zero é transmitida, então $w = 1$. Calcula-se a fdp de q_0 a partir de uma mudança de variável na fdp de y, *i.e.*, fazendo-se $y = q_0(\sigma^2/2)$, resultando em

$$\begin{aligned} p(q_0) &= \left|\frac{dy}{dq_0}\right| \cdot p(y|w=1)\Big|_{y=q_0\frac{\sigma^2}{2}}; \\ &= \frac{\sigma^2}{2} \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(q_0\frac{\sigma^2}{2}-1\right)^2}{2\sigma^2}\right); \\ &= \frac{\sigma}{2\sqrt{2\pi}} \exp\left(-\frac{\left(\frac{\sigma^2}{2}\right)^2\left(q_0-\frac{2}{\sigma^2}\right)^2}{2\sigma^2}\right); \\ &= \frac{\sigma}{2\sqrt{2\pi}} \exp\left(-\frac{\left(q_0-\frac{2}{\sigma^2}\right)^2}{2(4/\sigma^2)}\right). \end{aligned} \tag{5.3}$$

Considerando um código LDPC (d_s, d_c)-regular, a mensagem passada pelos nós de símbolo para um nó de verificação é calculada através de

$$q = q_0 + \sum_{i=1}^{d_s-1} r_i, \tag{5.4}$$

em que q_0 é a mensagem inicial e r_i, $i = 1, \cdots, d_s-1$ são as mensagens de entrada provenientes de todos os ramos incidentes, exceto o ramo pelo qual q será enviada. A Figura 5.1 ilustra o cálculo da mensagem de saída em um nó de símbolo.

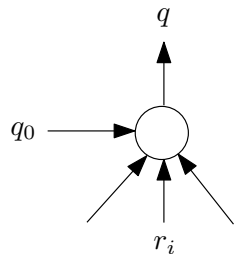


Figura 5.1: *Cálculo da mensagem de saída em um nó de símbolo.*

Como q é a soma de diversas variáveis aleatórias independentes, sua fdp é calculada pela convolução das fdp de cada uma das mensagens recebidas. Esta convolução pode ser calculada pela multiplicação das transformadas de Fourier de cada uma das fdp e em seguida calculando a transformada inversa. Sejam $p_l(q)$ e $p_l(r)$, as funções densidade de probabilidade da mensagem de saída dos nós de símbolo e de verificação na l-ésima iteração, respectivamente.

Se $\mathcal{F}$ representa a transformada de Fourier calcula-se $p_{l+1}(q)$ através de

$$p_{l+1}(q) = \mathcal{F}^{-1}\left\{\mathcal{F}\big[p(q_0)\big]\Big[\mathcal{F}[p_l(r)]\Big]^{d_s-1}\right\}, \tag{5.5}$$

em que, $p(q_0)$ é a fdp da mensagem inicial q_0 e $p_0(r) \triangleq \delta(r)$ é a função impulso unitário. Portanto, a fração de mensagens incorretas após l iterações é dada por

$$P_e(l) = \int_{-\infty}^{0} p_l(q)dq. \tag{5.6}$$

Para efetuar o cálculo da probabilidade de erro é preciso conhecer a densidade $p_l(r)$. A mensagem r passada pelo nó de verificação para um nó de símbolo é dada por

$$r = 2\tanh^{\text{-}1}\left(\prod_{i=1}^{d_c-1}\tanh\left(\frac{q_i}{2}\right)\right),$$

em que as mensagens q_i são variáveis aleatórias independentes e identicamente distribuídas. A Figura 5.2 ilustra o cálculo da mensagem de saída em um nó de verificação.

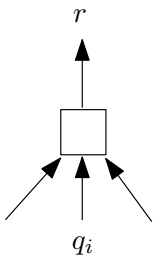


Figura 5.2: *Cálculo da mensagem de saída em um nó de verificação.*

O objetivo agora é calcular a fdp da mensagem de saída r do nó de verificação a partir das densidades das mensagens q_i, na l-ésima iteração. Inicia-se o cálculo com a seguinte relação entre as mensagens r e q_i:

$$\tanh\left(\frac{r}{2}\right) = \prod_{i=1}^{d_c-1}\tanh\left(\frac{q_i}{2}\right). \tag{5.7}$$

A expressão no lado direito da Equação (5.7) pode ser convertida em uma soma de variáveis aleatórias independentes com o uso do logaritmo. No entanto, é necessário atentar para o fato de que q_i pode assumir valores negativos e a tangente hiperbólica é uma função ímpar. Utilizando a função sinal definida por

$$\text{sign}(x) \triangleq \begin{cases} 1, & x > 0; \\ -1, & x \leq 0, \end{cases}$$

a Equação (5.7) pode ser reescrita como

$$\tanh\left(\frac{r}{2}\right) = \left(\prod_{i=1}^{d_c-1} \operatorname{sign}(q_i)\right) \exp\left(\sum_{i=1}^{d_c-1} \log\left|\tanh\left(\frac{q_i}{2}\right)\right|\right). \tag{5.8}$$

Para calcular $p_l(r)$, é preciso obter a densidade de probabilidade de $\log|\tanh(q_i/2)|$ condicionada a $\operatorname{sign}(q_i) = 1$ e $\operatorname{sign}(q_i) = -1$. Primeiro, obtém-se as densidades condicionais

$$p\big(q_i|\operatorname{sign}(q_i)=1\big) = \begin{cases} \frac{p(q_i)}{\int_0^\infty p(q_i)dq_i}, & q_i > 0; \\ 0, & q_i \leq 0. \end{cases}$$

$$p\big(q_i|\operatorname{sign}(q_i)=-1\big) = \begin{cases} 0, & q_i \geq 0; \\ \frac{p(q_i)}{\int_{-\infty}^0 p(q_i)dq_i}, & q_i < 0. \end{cases}$$

Sendo $g_i \triangleq \log|\tanh(q_i/2)|$, calcula-se as densidades condicionais para g_i, que é função de uma variável aleatória. Se $\operatorname{sign}(q_i) = 1$, então $g_i = \log\big(\tanh(q_i/2)\big)$. Como g_i é uma função injetora, a fdp da variável g_i é dada por

$$p(g_i') = p\big(q_i = 2\tanh^{-1}(e^{g_i'})|\operatorname{sign}(q_i)=1\big) \cdot \left|\frac{d}{dg_i'} q_i(g_i')\right|.$$

Portanto,

$$p(g_i') = p\big(q_i = 2\tanh^{-1}(e^{g_i'})|\operatorname{sign}(q_i)=1\big) \cdot \frac{2e^{g_i'}}{1-e^{g_i'}}.$$

De forma análoga, se $\operatorname{sign}(q_i) = -1$ e $g_i = \log\big(\tanh(-q_i/2)\big)$, ou seja,

$$q_i = -2\tanh^{-1}(e^{g_i'}),$$

a função densidade de probabilidade de g_i é dada por

$$p(g_i') = p\big(q_i = -2\tanh^{-1}(e^{g_i'})\operatorname{sign}(q_i)=-1\big) \cdot \frac{2e^{g_i'}}{1-e^{g_i'}}$$

Portanto,

$$p(g_i|\operatorname{sign}(q_i)=1) = p\big(q_i = 2\tanh^{-1}(e^{g_i})\big) \cdot \frac{2e^{g_i}}{1-e^{g_i}}; \tag{5.9}$$

$$p(g_i|\operatorname{sign}(q_i)=-1) = p\big(q_i = -2\tanh^{-1}(e^{g_i})\big) \cdot \frac{2e^{g_i}}{1-e^{g_i}}. \tag{5.10}$$

Na Equação (5.8) existe uma exponencial de uma soma de variáveis aleatórias independentes. A fdp desta soma pode ser encontrada de forma semelhante à Equação (5.5). Primeiramente, calcula-se a transformada de Fourier de (5.9) e (5.10)

$$G_i^+(j\omega) \triangleq \mathcal{F}\big[p(g_i|\operatorname{sign}(q_i)=1)\big];$$

$$G_i^-(j\omega) \triangleq \mathcal{F}\big[p(g_i|\operatorname{sign}(q_i)=-1)\big].$$

Em seguida define-se

$$g_o \triangleq \sum_{i=1}^{d_c-1} g_i.$$

Sendo u o número de mensagens negativas; e $\text{NM}(u)$ representa o evento de existirem u mensagens negativas chegando ao nó de verificação, então

$$p\big(g_o|\text{NM}(u)\big) = \mathcal{F}^{-1}\Big[\big(G_i^+(j\omega)\big)^u\big(G_i^-(j\omega)\big)^{d_c-1-u}\Big]. \tag{5.11}$$

Agora, considera-se a relação entre as variáveis r e g_o para calcular a fdp de r a partir de $p\big(g_o|\text{NM}(u)\big)$. Quando u é par:

$$\prod_{i=1}^{d_c-1} \text{sign}(q_i) = 1 \Rightarrow r = 2\tanh^{-1}\left(e^{g_o}\right).$$

Quando u é ímpar:

$$\prod_{i=1}^{d_c-1} \text{sign}(q_i) = -1 \Rightarrow r = 2\tanh^{-1}\left(e^{-g_o}\right).$$

Portanto, quando u é par a função densidade de probabilidade condicional é dada por

$$p\big(r|\text{NM}(u)\big) = p\big(g_o = \log\tanh\left(r/2\right)|\text{NM}(u)\big) \cdot \left|\frac{d}{dr}\log\tanh\left(\frac{r}{2}\right)\right|; \tag{5.12}$$

para u ímpar, ela é dada por

$$p\big(r|\text{NM}(u)\big) = p\big(g_o = \log\tanh\left(-r/2\right)|\text{NM}(u)\big) \cdot \left|\frac{d}{dr}\log\tanh\left(\frac{-r}{2}\right)\right|, \tag{5.13}$$

em que

$$\left|\frac{d}{dr}\log\tanh\left(\frac{r}{2}\right)\right| = \left|\frac{d}{dr}\log\tanh\left(\frac{-r}{2}\right)\right| = \frac{1-\tanh^2(r/2)}{2\tanh(r/2)}. \tag{5.14}$$

Substituindo (5.14) nas Equações (5.12) e (5.13), obtém-se

$$p\big(r|\text{NM}(u)\big) = \begin{cases} p\big(g_o = \log\tanh\left(r/2\right)|\text{NM}(u)\big) \cdot \frac{1-\tanh^2(r/2)}{2\tanh(r/2)}, & u \text{ par, } r > 0; \\ p\big(g_o = \log\tanh\left(-r/2\right)|\text{NM}(u)\big) \cdot \frac{1-\tanh^2(r/2)}{2\tanh(r/2)}, & u \text{ ímpar, } r < 0; \\ 0, & \text{caso contrário.} \end{cases}$$

Finalmente, tomando a média para todos os valores possíveis para $\text{NM}(u)$, a expressão para a densidade de probabilidade da mensagem de saída do nó de verificação é dada por

$$p(r) = \sum_{u=0}^{d_c-1} \binom{d_c-1}{u} \left(\int_{-\infty}^{0} p(q_i)\right)^u \left(\int_{0}^{\infty} p(q_i)\right)^{d_c-u} p\big(r|\text{NM}(u)\big).$$

Este procedimento é chamado de *density evolution* e permite calcular a probabilidade de erro de mensagens l-ésima iteração, $Pe(l)$. Para um determinado nível de ruído com desvio

padrão σ, é possível executar este algoritmo iterativamente e observar o comportamento da seqüência de probabilidades de erro a cada iteração. Se $Pe(l)$ tende a zero quando l aumenta, para aquele determinado canal e conjunto de códigos LDPC, o algoritmo de decodificação converge. O maior valor de σ para o qual

$$\lim_{l \to \infty} Pe(l) = 0$$

é chamado de limiar de decodificação σ^*.

Calcular a seqüência $Pe(l)$ usando *density evolution* em sua forma integral descrita nesta seção é uma tarefa computacionalmente intensa e de difícil implementação. Na próxima seção, é descrito um método muito mais simples para observar a convergência do algoritmo SP, a análise por aproximação Gaussiana.

5.2 Aproximação Gaussiana

Embora exista uma solução exata para a *density evolution* para o canal RAGB, a complexidade das expressões dificultam a sua implementação. Uma solução encontrada para este problema é aproximar as fdp das mensagens por funções Gaussianas [15]. A aproximação Gaussiana reduz muito a complexidade da *density evolution* e apresenta resultados muito próximos da solução exata.

Uma função densidade de probabilidade Gaussiana é completamente especificada por sua média e variância. É preciso conhecer apenas estes parâmetros para calcular a evolução das densidades das mensagens no decodificador. Existe uma condição importante, chamada *condição de simetria*, que quando satisfeita, é preservada durante a *density evolution* para todas as mensagens. A condição de simetria pode ser expressa como $p(x) = p(-x)e^x$, em que $p(x)$ é a densidade de uma mensagem. Para Gaussianas de média m e variância σ^2, a condição de simetria se reduz a $\sigma^2 = 2m$. Como visto na Equação (5.3), a mensagem q_0 é uma variável aleatória Gaussiana de média $2/\sigma^2$ e variância $4/\sigma^2$, em que σ^2 é a variância do ruído no canal RAGB. Esta densidade satisfaz a condição de simetria. Esta mesma relação entre média e variância é assumida para as demais mensagens q e r.

Como a relação entre a média e a variância das mensagens é bem determinada, podemos calcular a *density evolution* pela determinação de um único parâmetro, a média das distribuições a cada iteração. A partir da Equação (5.4), média da mensagem q na l-ésima iteração é

dada por

$$m_q^{(l)} = m_{q_0} + (d_s - 1)m_r^{(l-1)}. \tag{5.15}$$

O índice i é omitido pois as mensagens r_i, $1 \leq i < d_s$, são independentes e identicamente distribuídas (iid) e possuem a mesma média m_r. Observe que $m_r^{(0)} = 0$ pois todas as mensagens provenientes dos nós de verificação nesta iteração são iguais a zero.

A média $m_r^{(l)}$ atualizada na l-ésima iteração pode ser calculada tomando os valores esperados em ambos os lados da Equação (5.7) para obter

$$E\left[\tanh\left(\frac{r^{(l)}}{2}\right)\right] = E\left[\prod_{i=1}^{d_c-1}\tanh\left(\frac{q_i^{(l)}}{2}\right)\right].$$

Como as variáveis q_i são iid, esta expressão se torna

$$E\left[\tanh\left(\frac{r^{(l)}}{2}\right)\right] = \left(E\left[\tanh\left(\frac{q^{(l)}}{2}\right)\right]\right)^{d_c-1}.$$

Assumindo que a mensagem r possui uma densidade de probabilidade Gaussiana com média m_r e variância $2m_r$, o valor esperado da função $\tanh(r/2)$ é dado por

$$\begin{aligned} E\left[\tanh\left(\frac{r}{2}\right)\right] &= \int_{-\infty}^{\infty}\left(\tanh\frac{r}{2}\right)f(r)dr; \\ &= \int_{-\infty}^{\infty}\left(\tanh\frac{r}{2}\right)\frac{1}{\sqrt{4\pi m_r}}\exp\left[-\frac{(r-m_r)^2}{4m_r}\right]dr; \\ &= \frac{1}{\sqrt{4\pi m_r}}\int_{-\infty}^{\infty}\left(\tanh\frac{r}{2}\right)\exp\left[-\frac{(r-m_r)^2}{4m_r}\right]dr. \end{aligned} \tag{5.16}$$

Define-se a função $\phi(x)$, $x \in [0, \infty)$:

$$\phi(x) \triangleq \begin{cases} 1 - \frac{1}{\sqrt{4\pi x}}\int_{-\infty}^{\infty}\left(\tanh\frac{r}{2}\right)\exp\left[-\frac{(r-x)^2}{4x}\right]dr, & x > 0 \\ 1, & x = 0. \end{cases} \tag{5.17}$$

A função $\phi(x)$ é mostrada na Figura 5.3, de onde é possível concluir que se trata de uma função monotonicamente decrescente.

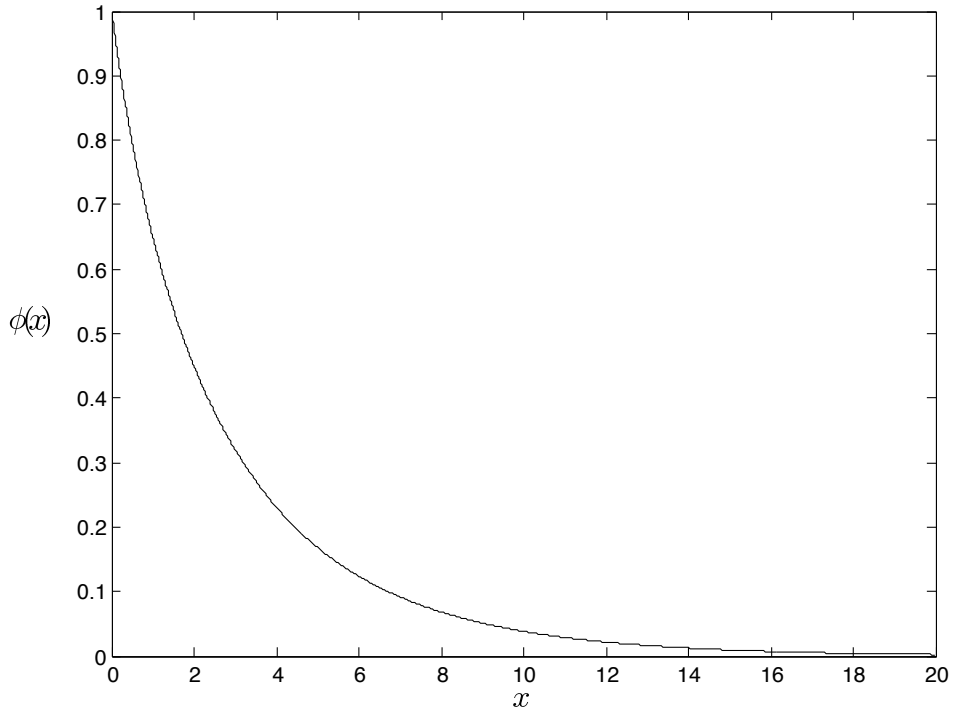


Figura 5.3: *Função $\phi(x)$.*

A partir de

$$\begin{aligned}\phi(m_r^{(l)}) &= 1 - E\left[\tanh\left(\frac{r^{(l)}}{2}\right)\right];\\ \phi(m_q^{(l)}) &= 1 - E\left[\tanh\left(\frac{q^{(l)}}{2}\right)\right],\end{aligned}$$

obtém-se a seguinte relação entre as médias $m_r^{(l)}$ e $m_q^{(l)}$:

$$1 - \phi(m_r^{(l)}) = \left[1 - \phi(m_q^{(l)})\right]^{d_c - 1}. \tag{5.18}$$

Resolvendo a Equação (5.18) para $m_r^{(l)}$,

$$m_r^{(l)} = \phi^{-1}\left(1 - \left[1 - \phi(m_q^{(l)})\right]^{d_c - 1}\right). \tag{5.19}$$

Substituindo (5.19) em (5.15), obtém-se

$$m_q^{(l)} = m_{q_0} + (d_s - 1)\phi^{-1}\left(1 - \left[1 - \phi(m_q^{(l-1)})\right]^{d_c - 1}\right).$$

A decodificação é livre de erros se a média $m_q^{(l)}$ diverge para infinito à medida que o número de iterações cresce. No entanto, é mais conveniente encontrar soluções para uma

recursão que converge para zero. Seja $t^{(l)} \triangleq \phi(m_q^{(l)})$, reescreve-se

$$m_q^{(l)} = m_{q_0} + (d_s - 1)\phi^{-1}\left(1 - \left[1 - t^{(l-1)}\right]^{d_c - 1}\right). \tag{5.20}$$

Aplicando a função ϕ dos dois lados da Equação 5.20

$$\phi(m_q^{(l)}) = \phi\left(m_{q_0} + (d_s - 1)\phi^{-1}\left(1 - \left[1 - t^{(l-1)}\right]^{d_c - 1}\right)\right).$$

Ou seja,

$$t^{(l)} = \phi\left(m_{q_0} + (d_s - 1)\phi^{-1}\left(1 - \left[1 - t^{(l-1)}\right]^{d_c - 1}\right)\right). \tag{5.21}$$

Como a função ϕ é monotonicamente decrescente, a recursão converge para zero a medida que $m_q^{(l)}$ aumenta. Dessa forma é possível visualizar melhor a convergência do algoritmo SP.

A Figura 5.4 ilustra a convergência do algoritmo SP para um conjunto de códigos LDPC $(3, 6)$-regular em um canal RAGB com $\sigma = 0.83$. As setas verticais e horizontais indicam a evolução de $\phi(m_q^{(l)})$ com o número de iterações. Quanto maior o intervalo entre a reta $t^{(l)} = t^{(l-1)}$ e a curva definida por (5.21), mais rápido o algoritmo SP converge. Isso indica que o canal apresenta-se em boas condições, *i.e.*, ruído com baixa potência. A medida que a potência do ruído aumenta, a distância entre a reta e a curva diminui, e a convergência passa a ser mais lenta. O limite para a convergência do algoritmo SP acontece quando a reta tangencia a curva. Neste caso, a probabilidade de erro do algoritmo está limitada a um valor maior que zero. O valor de σ para o qual esta situação ocorre é denominado *limiar* de decodificação e é denotado por σ^*.

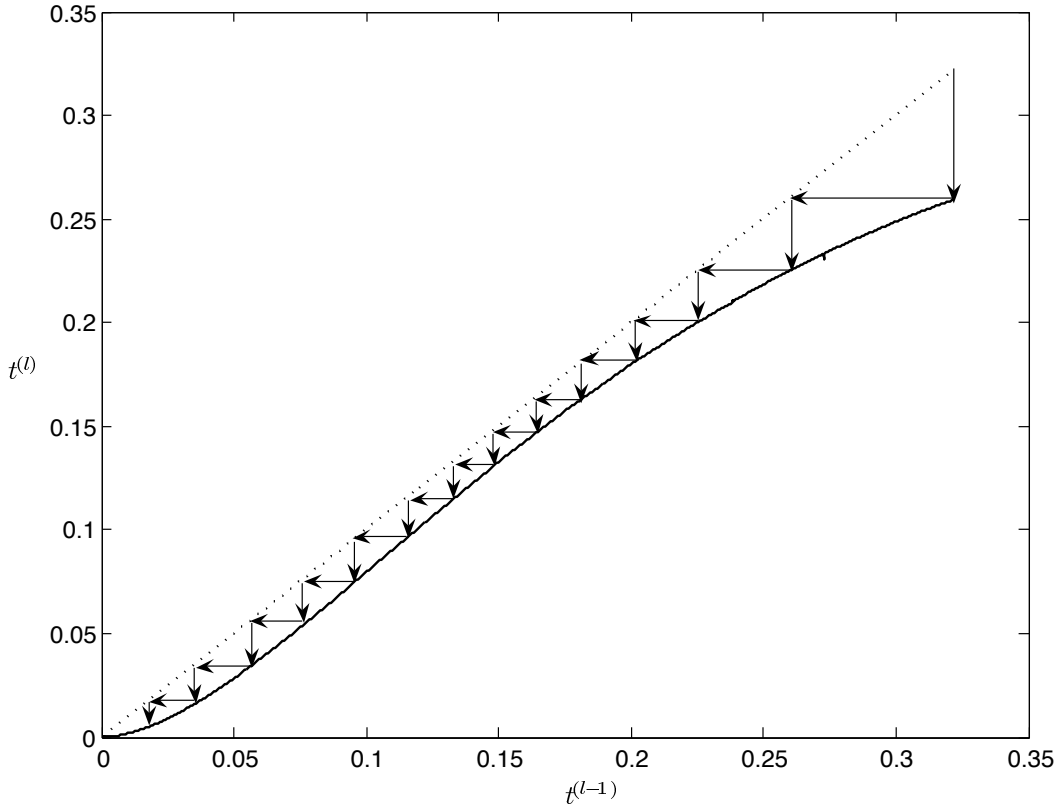


Figura 5.4: *Convergência do algoritmo SP para um conjunto de códigos LDPC* $(3,6)$*-regular em um canal RAGB com* $\sigma = 0.83$.

5.3 Limiares de decodificação para códigos LDPC

Na análise por *density evolution* as funções densidade de probabilidade das mensagens após l iterações são funções do nível de ruído do canal e dos graus dos nós de verificação e de símbolo. Para um dado conjunto de códigos LDPC existe um nível limite de ruído do canal, acima do qual a decodificação iterativa com o algoritmo SP não converge. Este nível de ruído é chamado de limiar e pode ser determinado numericamente por

$$\sigma^* = \sup\{\sigma : t^{(l)} < t^{(l-1)}, \forall t^{(l-1)}\}.$$

Esta situação é ilustrada pela Figura 5.5 para um conjunto de códigos LDPC $(3,6)$-regular em um canal RAGB com $\sigma^* = 0.875$. É possível observar a média m_q^l não tende a infinito e portanto, a probabilidade de erro não tende a zero.

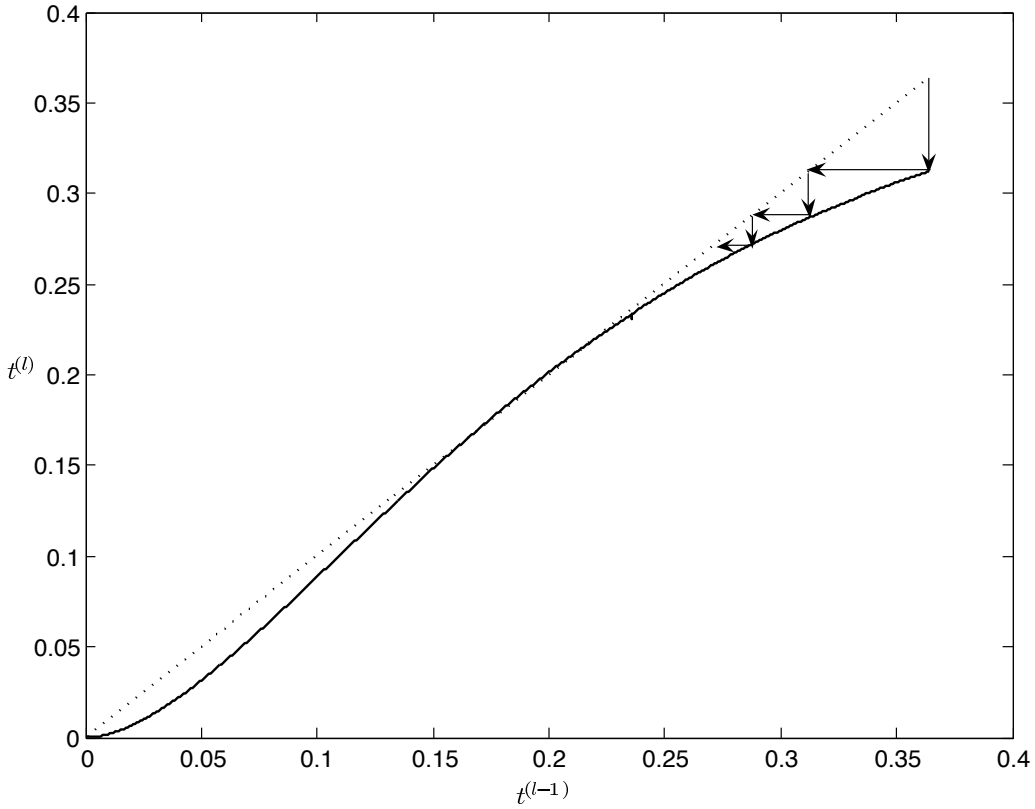


Figura 5.5: *Limiar de decodificação para um conjunto de códigos LDPC* $(3, 6)$*-regular em um canal RAGB.*

A Tabela 5.1 mostra o valor de limiares de decodificação para alguns conjuntos de códigos regulares de taxa $R = 1/2$. O limiar para o conjunto $(3, 6)$-regular implica em um valor limite para seu desempenho livre de erros. Para códigos LDPC $(3, 6)$-regulares infinitamente longos, a probabilidade de erro no canal RAGB pode ser igual a zero apenas se E_b/N_0 é maior que 1.160 dB. Na prática, quando códigos de comprimento finito são utilizados, a distância entre o desempenho do código e o limiar do conjunto aumenta com a diminuição do comprimento do bloco. É possível observar que, para códigos de taxa $R = 1/2$, o aumento no grau dos nós de símbolo implica em um aumento no limiar de decodificação.

Tabela 5.1: *Limiares de decodificação para conjuntos de códigos LDPC regulares calculados com a aproximação Gaussiana para a density evolution.*

Conjunto (d_s, d_c)	Limiar (σ^*)	Eb/N_0 (dB)
$(3, 6)$	0.875	1.160
$(4, 8)$	0.830	1.618
$(5, 10)$	0.785	2.103
$(6, 12)$	0.750	2.499

Para alcançar a capacidade do canal RAGB, o limiar de decodificação do código deve ser igual ao limite de Shannon. Isto é conseguido através de códigos irregulares. As equações derivadas neste capítulo podem ser alteradas para analisar o desempenho de códigos irregulares e então projetar distribuições de grau de modo que o limiar seja tão próximo do limite de Shannon quanto se queira. Então, para obter um código cujo desempenho se aproxima da capacidade basta escolher dentro do conjunto determinado por estas distribuições um código suficientemente longo.

capítulo 6

Algoritmo para decodificação em canais Gilbert-Elliott

O sucesso dos códigos LDPC em canais sem memória motivou a investigação do seu uso em modelos de canais mais complexos, por exemplo, canais com memória [19]. Dentre os canais com memória, o canal Gilbert-Elliott (GE) se destaca por ser capaz de modelar diversas situações práticas tais como canais telefônicos, enlaces de microondas, canais de comunicação via-satélite, sistemas de gravação óptica e magnética, canais de acesso múltiplo por espalhamento espectral, desvanecimento em canais de rádio móvel e sistemas de comunicação com perda de sincronismo [23].

No canal GE, o processo de transmissão é perturbado por erros que ocorrem de forma correlacionada ou em *surtos*. Com o objetivo de eliminar ou diminuir a correlação entre os erros dentro de um um bloco de comprimento N, utiliza-se o *entrelaçamento* de palavras-código. Se o entrelaçador for longo o suficiente, o *canal entrelaçado* (cascateamento do entrelaçador, canal e desentrelaçador) pode ser considerado sem memória, permitindo que codificadores e decodificadores convencionais para canais sem memória sejam utilizados.

Esta abordagem, embora eficaz, limita o desempenho do decodificador que não faz uso da dependência estatística entre os erros introduzidos pelo canal para melhorar a estimativa das palavras-código transmitidas [24]. Ao utilizar um decodificador de decisão suave, como o algoritmo SP, e modificando os LLRs de entrada convenientemente para levar em conta a memória do canal, é possível melhorar o desempenho dos códigos LDPC em um canal GE em

relação ao sistema que utiliza apenas o entrelaçamento.

Neste capítulo, um sistema para *decodificação por decisão com realimentação* (*decision feedback decoder* ou DFD) para decodificação de códigos LDPC em canais GE é proposto. O DFD consiste no cascateamento de um calculador de LLR e em um decodificador de decisão suave. Este sistema foi inicialmente proposto por Mushkin e Bar-David [24]. Um esquema para decodificação de códigos convolucionais foi proposto e analisado por Pimentel e Rêgo [36]. Tal sistema mostrou-se promissor ao apresentar um ganho considerável em relação ao desempenho do sistema que utiliza apenas o entrelaçamento.

O DFD funciona fazendo uso das decisões anteriores para realizar estimações recursivas da probabilidade de ocorrer um erro no próximo símbolo transmitido, condicionada aos erros anteriores do canal. Esta estimativa da probabilidade é utilizada para calcular os LLRs de entrada utilizados pelo algoritmo SP na decodificação de cada palavra recebida.

6.1 O sistema de comunicações

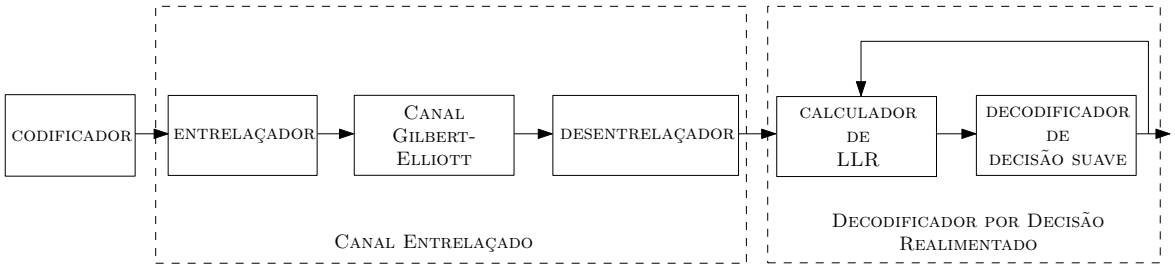


Figura 6.1: *Sistema de comunicações com decodificador por decisão com realimentação.*

A Figura 6.1 ilustra o sistema de comunicações que emprega um DFD para o canal GE. O sistema DFD proposto em [36] utiliza o decoficador de Viterbi de decisão suave. O interesse no presente capítulo é propor um sistema semelhante que usa códigos LDPC e o algoritmo SP. A principal diferença entre os dois sistemas é que enquanto o algoritmo de Viterbi é ótimo, o algoritmo SP é sub-ótimo e iterativo.

Como visto anteriormente, a modelagem da função densidade de probabilidade conjunta *a posteriori* dos símbolos recebidos por meio de grafos fator assume que o canal é *sem memória*. No entanto, o entrelaçador e desentrelaçador fazem com que os erros introduzidos em surtos pelo canal sejam distribuídos entre diferentes palavras-código. Quando o entrelaçador é suficientemente longo, os erros dentro de um mesmo bloco de comprimento N podem ser

considerados *aproximadamente* independentes. Desta forma o decodificador interpreta o canal GE entrelaçado como um canal BSC. Portanto, o entrelaçamento permite a modelagem de códigos por grafos fator. O *calculador de LLR*, processa cada palavra recebida do canal entrelaçado juntamente com a palavra decodificada no instante imediatamente anterior. Uma relação recursiva para a probabilidade de erro condicionada aos erros anteriores é utilizada para estimar a probabilidade de erro em cada uso subseqüente do canal. A partir destas probabilidades, são calculados os logaritmos das razões de verossimilhança que servirão de entrada para o decodificador de decisão suave.

De agora em diante, os subscritos dos vetores são interpretados da seguinte maneira. Por exemplo, no vetor $\mathbf{w}_l \triangleq (w_1, \cdots, w_l)$, os índices das componentes dos vetores indicam instantes de tempo. A variável e_k denota o erro introduzido no k-ésimo uso do canal e q_k denota a probabilidade de ocorrer um erro no instante k, condicionada aos erros anteriores do canal.

6.1.1 O canal Gilbert-Elliott

O canal Gilbert-Elliott (GE) é um canal binário simétrico variante no tempo, cuja probabilidade de transição é determinada pelo estado atual de uma cadeia de Markov de dois estados denominados respectivamente estado *bom* e estado *ruim* [24]. No estado *bom*, denotado por G, os erros ocorrem com um pequena probabilidade g enquanto no estado *ruim*, denotado por B, os erros ocorrem com uma maior probabilidade b. A probabilidade de transição entre os estados da cadeia de Markov são Q e q, como mostrado na figura 6.2.

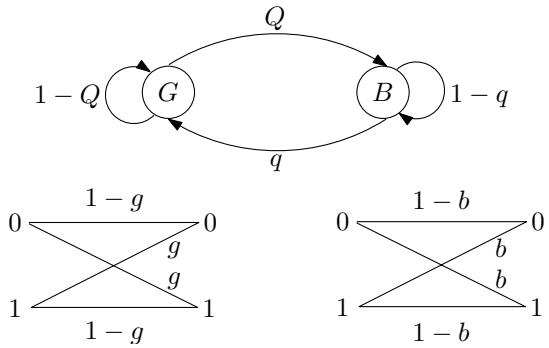


Figura 6.2: *Modelo do canal Gilbert-Elliott.*

Sejam $x_k \in \{0,1\}$, $y_k \in \{0,1\}$ e $e_k \triangleq x_k \oplus y_k$, a entrada, saída e erro do canal GE no instante k, respectivamente. O *processo de erro* do canal $\{e_k\}_{k=1}^{\infty}$ é independente da entrada do canal, *i.e.*, $P(\mathbf{e}_l|\mathbf{x}_l) = P(\mathbf{e}_l)$. O processo de erro no canal depende da seqüência de estados de uma cadeia de Markov (chamada de *processo de estado*) $\{s_k\}_{k=0}^{\infty}$, no qual $s_k = \{G, B\}$. Portanto, o processo de erro possui memória. No entanto, quando condicionado ao processo de estado, o processo de erro não possui memória, *i.e.*,

$$P(\mathbf{e}_l|\mathbf{s}_l) = \prod_{k=1}^{l} P(e_k|s_k).$$

O processo de estado é um processo de Markov estacionário de primeira ordem, *i.e.*, $P(s_l|\mathbf{s}_{l-1}) = P(s_l|s_{l-1})$. Os parâmetros do canal são as probabilidades de transição

$$g \triangleq P(e_k = 1|s_k = G) \quad \text{e} \quad b \triangleq P(e_k = 1|s_k = B),$$

e as probabilidades de transição de estados da cadeia de Markov

$$Q \triangleq P(s_k = G|s_{k-1} = B) \quad \text{e} \quad q \triangleq P(s_k = B|s_{k-1} = G).$$

A distribuição inicial dos estados que garante a estacionariedade da cadeia de Markov é

$$P(s_0 = G) = \frac{q}{q+Q} \quad \text{e} \quad P(s_0 = B) = \frac{Q}{q+Q}.$$

A probabilidade de erro de bit é

$$\begin{aligned} P(e_k = 1) &= P(e_k = 1, s_k = G) + P(e_k = 1, s_k = B); \\ &= P(e_k = 1|s_k = G)P(s_k = G) + P(e_k = 1|s_k = B)P(s_k = B); \\ &= g\frac{q}{q+Q} + b\frac{Q}{q+Q}. \end{aligned} \tag{6.1}$$

A *memória* μ do canal é definida a partir de [24]

$$P(s_k = \xi|s_0 = \xi) - P(s_k = \xi|s_0 \neq \xi) = (1-q-Q)^k, \tag{6.2}$$

em que $\xi = \{G, B\}$. Definindo a memória μ por

$$\mu \triangleq 1 - q - Q,$$

o lado direito da igualdade na Equação (6.2) tende a zero à medida que k cresce, pois $|\mu| < 1$ (assumindo que $0 < q, Q < 1$). Portanto, para um k suficientemente grande a probabilidade da cadeia de Markov estar em um estado ξ no instante de tempo k independe do estado inicial s_0.

Outro parâmetro importante do canal GE é o comprimento médio dos surtos. O número médio de intervalos de tempo em que o canal se encontra no estado B é definido como sendo o comprimento médio dos surtos de erros. Este valor é denotado por λ e é possível mostrar que

$$\lambda = \frac{1}{q}.$$

Seja $q_k(\mathbf{e}_{k-1})$ a probabilidade de ocorrer um erro no instante k, condicionada aos erros anteriores do canal, ou seja,

$$q_k(\mathbf{e}_{k-1}) = P(e_k = 1|\mathbf{e}_{k-1}).$$

É possível obter uma expressão recursiva para a probabilidade q_{k+1} em função de q_k [24]. Primeiramente, considera-se a probabilidade de ocorrer um erro no $k+1$-ésimo uso do canal condicionada aos erros nos k instantes anteriores.

$$\begin{aligned}
q_{k+1}(\mathbf{e}_k) &= P(e_{k+1} = 1|\mathbf{e}_k); \\
&= P(e_{k+1} = 1, s_{k+1} = G|\mathbf{e}_k) + P(e_{k+1} = 1, s_{k+1} = B|\mathbf{e}_k); \\
&= P(e_{k+1} = 1|s_{k+1} = G, \mathbf{e}_k)P(s_{k+1} = G|\mathbf{e}_k) + P(e_{k+1} = 1|s_{k+1} = B, \mathbf{e}_k)P(s_{k+1} = B|\mathbf{e}_k); \\
&= P(e_{k+1} = 1|s_{k+1} = G)P(s_{k+1} = G|\mathbf{e}_k) + P(e_{k+1} = 1|s_{k+1} = B)P(s_{k+1} = B|\mathbf{e}_k); \\
&= gP(s_{k+1} = G|\mathbf{e}_k) + bP(s_{k+1} = B|\mathbf{e}_k); \\
&= g\big(1 - P(s_{k+1} = B|\mathbf{e}_k)\big) + bP(s_{k+1} = B|\mathbf{e}_k); \\
&= g + (b - g)P(s_{k+1} = B|\mathbf{e}_k). \qquad (6.3)
\end{aligned}$$

Manipulando a expressão para a probabilidade da cadeia de Markov estar no estado B no instante $k+1$, dados os erros nos k instantes anteriores, obtém-se

$$\begin{aligned}
P(s_{k+1} = B|\mathbf{e}_k) &= P(s_{k+1} = B, s_k = G|\mathbf{e}_k) + P(s_{k+1} = B, s_k = B|\mathbf{e}_k); \\
&= P(s_{k+1} = B|s_k = G, \mathbf{e}_k)P(s_k = G|\mathbf{e}_k) + P(s_{k+1} = B|s_k = B, \mathbf{e}_k)P(s_k = B|\mathbf{e}_k); \\
&= P(s_{k+1} = B|s_k = G)P(s_k = G|\mathbf{e}_k) + P(s_{k+1} = B|s_k = B)P(s_k = B|\mathbf{e}_k); \\
&= QP(s_k = G|\mathbf{e}_k) + (1 - q)P(s_k = B|\mathbf{e}_k); \\
&= Q\big(1 - P(s_k = B|\mathbf{e}_k)\big) + (1 - q)P(s_k = B|\mathbf{e}_k); \\
&= Q + (1 - q - Q)P(s_k = B|\mathbf{e}_k); \\
&= Q + \mu P(s_k = B|\mathbf{e}_k). \qquad (6.4)
\end{aligned}$$

A substituição da Equação 6.4 em 6.3 resulta na seguinte expressão:

$$\begin{aligned} q_{k+1}(\mathbf{e}_k) &= g + (b-g)\big(Q + \mu P(s_k = B|\mathbf{e}_k)\big) \\ &= g + Q(b-g) + \mu(b-g)P(s_k = B|\mathbf{e}_k). \end{aligned} \tag{6.5}$$

Aplicando a regra de Bayes para a expressão da probabilidade da cadeia de Markov estar no estado $s_k = \xi$ dados os erros $\mathbf{e}_k$, obtém-se

$$\begin{aligned} P(s_k = \xi|\mathbf{e}_k) &= \frac{P(s_k = \xi, e_k|\mathbf{e}_{k-1})}{P(e_k|\mathbf{e}_{k-1})}; \\ &= \frac{P(e_k|s_k = \xi, \mathbf{e}_{k-1})P(s_k = \xi|\mathbf{e}_{k-1})}{P(e_k|\mathbf{e}_{k-1})}; \\ &= \frac{P(e_k|s_k = \xi)P(s_k = \xi|\mathbf{e}_{k-1})}{P(e_k|\mathbf{e}_{k-1})}. \end{aligned} \tag{6.6}$$

Considerando a expressão para $q_k(\mathbf{e}_{k-1})$ obtida à partir de (6.3):

$$q_k(\mathbf{e}_{k-1}) = g + (b-g)P(s_{k-1} = B|\mathbf{e}_{k-1}) \tag{6.7}$$

e resolvendo (6.7) para $P(s_k = B|\mathbf{e}_{k-1})$, obtém-se

$$P(s_k = B|\mathbf{e}_{k-1}) = \frac{q_k(\mathbf{e}_{k-1}) - g}{b-g}. \tag{6.8}$$

Fazendo $\xi = B$ e substituindo a Equação (6.8) em (6.6), obtém-se

$$\begin{aligned} P(s_k = B|\mathbf{e}_k) &= \frac{P(e_k|s_k = B)P(s_k = B|\mathbf{e}_{k-1})}{P(e_k|\mathbf{e}_{k-1})}; \\ &= \frac{P(e_k|s_k = B)}{P(e_k|\mathbf{e}_{k-1})} \cdot \frac{q_k(\mathbf{e}_{k-1}) - g}{b-g}. \end{aligned} \tag{6.9}$$

Finalmente, a substituição de (6.9) em (6.5) leva a

$$q_{k+1}(\mathbf{e}_k) = g + Q(b-g) + \mu\big(q_k(\mathbf{e}_{k-1}) - g\big)\frac{P(e_k|s_k = B)}{P(e_k|\mathbf{e}_{k-1})}. \tag{6.10}$$

Portanto, se $e_k = 1$:

$$\begin{aligned} q_{k+1}(\mathbf{e}_k) &= g + Q(b-g) + \mu\big(q_k(\mathbf{e}_{k-1}) - g\big)\frac{P(e_k = 1|s_k = B)}{P(e_k = 1|\mathbf{e}_{k-1})}; \\ &= g + Q(b-g) + \mu\big(q_k(\mathbf{e}_{k-1}) - g\big)\frac{b}{q_k(\mathbf{e}_{k-1})}; \\ &= g + Q(b-g) + \frac{\mu\big(q_k(\mathbf{e}_{k-1}) - g\big)b}{q_k(\mathbf{e}_{k-1})}; \end{aligned} \tag{6.11}$$

se $e_k = 0$:

$$\begin{aligned} q_{k+1}(\mathbf{e}_k) &= g + Q(b-g) + \mu\big(q_k(\mathbf{e}_{k-1}) - g\big)\frac{P(e_k = 0|s_k = B)}{P(e_k = 0|\mathbf{e}_{k-1})}; \\ &= g + Q(b-g) + \mu\big(q_k(\mathbf{e}_{k-1}) - g\big)\frac{1-b}{1 - q_k(\mathbf{e}_{k-1})}; \\ &= g + Q(b-g) + \frac{\mu\big(q_k(\mathbf{e}_{k-1}) - g\big)(1-b)}{1 - q_k(\mathbf{e}_{k-1})}. \end{aligned} \tag{6.12}$$

A relação recursiva para q_{k+1} no canal GE é dada por:

$$q_{k+1} = \begin{cases} g + Q(b-g) + \frac{\mu(q_k - g)b}{q_k}, & \text{se } e_k = 1; \\ g + Q(b-g) + \frac{\mu(q_k - g)(1-b)}{1-q_k}, & \text{se } e_k = 0. \end{cases} \tag{6.13}$$

O valor inicial $q_1 = P(e_1 = 1)$ é dado pela Equação (6.1).

6.1.2 O entrelaçador

A função de um entrelaçador é permutar os símbolos de uma seqüência de acordo com uma regra ou mapeamento *reversível*, de modo que seja possível re-obter a seqüência original a partir do mapeamento inverso [13]. O entrelaçamento é aplicado para reduzir a correlação entre dois erros consecutivos em uma mesma palavra-código transmitida através de um canal com memória, fazendo com que o canal entrelaçado possa ser considerado sem memória. Na realidade, o processo de entrelaçamento, por ser reversível, não remove a memória e sim a transforma em uma memória fragmentada latente que não interfere na operação usual de correção de erros [24].

Os entrelaçadores podem ser convolucionais ou de bloco. Neste capítulo serão considerados apenas os entrelaçadores de bloco. Este tipo de entrelaçador funciona da seguinte maneira. Uma seqüencia de $I_d \cdot N$ símbolos codificados $\{x_1, x_2, x_3, \cdots, x_{I_d \cdot N}\}$ é armazenada em uma matriz de dimensões $I_d \times N$, preenchendo-a linha por linha. A seqüência de saída do entrelaçador é obtida pela leitura dos elementos da matriz coluna por coluna. O valor de N representa o comprimento dos blocos a serem entrelaçados e I_d é a *profundidade* do entrelaçamento.

Exemplo 6.1 (Entrelaçador de bloco) *Considerando um entrelaçador de blocos com comprimento $N = 7$ e de profundidade $I_d = 5$. Seja a seqüência de entrada de 35 símbolos $\{x_1, x_2, x_3, \cdots, x_{35}\}$ deste entrelaçador. A Figura 6.3 ilustra como esta seqüência preenche o entrelaçador.*

x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}
x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	x_{21}
x_{22}	x_{23}	x_{24}	x_{25}	x_{26}	x_{27}	x_{28}
x_{29}	x_{30}	x_{31}	x_{32}	x_{33}	x_{34}	x_{35}

Figura 6.3: *Entrelaçador de bloco.*

A seqüência de saída é obtida lendo os símbolos pelas colunas e é igual a

$$\{x_1, x_8, x_{15}, x_{22}, x_{29}, x_2, x_9, x_{16}, x_{23}, x_{30}, \cdots, x_7, x_{14}, x_{21}, x_{28}, x_{35}\}.$$

Um entrelaçamento suficientemente longo de palavras-código transmitidas pelo canal GE faz com que o canal entrelaçado se comporte como um canal BSC cuja probabilidade de erro é igual a probabilidade de erro média do canal GE dada pela Equação 6.1. O teorema do processamento de dados [20] garante que o entrelaçamento reversível não implica em uma redução da capacidade do canal entrelaçado em relação à capacidade do canal original. Como o canal GE apresenta uma capacidade maior do que a do canal BSC equivalente (obtido pelo entrelaçamento), é possível melhorar o desempenho do decodificador utilizando a memória latente do sistema [24].

Supondo que a seqüência de saída de um entrelaçador de bloco seja transmitida através de um canal GE. A seqüência de saída do canal $\{y_1, y_2, y_3, \cdots, y_{I_d \cdot N}\}$ é desentrelaçada sendo armazenada em uma matriz de dimensões $I_d \times N$, coluna por coluna e lendo os símbolos linha por linha.

Exemplo 6.2 (Desentrelaçador de bloco) *Considerando o arranjo de desentrelaçamento na Figura 6.2 de uma seqüência de comprimento 35 que foi entrelaçada e transmitida através de um canal GE, é possível observar que dois símbolos recebidos consecutivamente são corrompidos por erros separados por exatamente* $I_d = 5$.

$x_1 \oplus e_1$	$x_2 \oplus e_6$	$x_3 \oplus e_{11}$	$x_4 \oplus e_{16}$	$x_5 \oplus e_{21}$	$x_6 \oplus e_{26}$	$x_7 \oplus e_{31}$
$x_8 \oplus e_2$	$x_9 \oplus e_7$	$x_{10} \oplus e_{12}$	$x_{11} \oplus e_{17}$	$x_{12} \oplus e_{22}$	$x_{13} \oplus e_{27}$	$x_{14} \oplus e_{32}$
$x_{15} \oplus e_3$	$x_{16} \oplus e_8$	$x_{17} \oplus e_{13}$	$x_{18} \oplus e_{18}$	$x_{19} \oplus e_{23}$	$x_{20} \oplus e_{28}$	$x_{21} \oplus e_{33}$
$x_{22} \oplus e_4$	$x_{23} \oplus e_9$	$x_{24} \oplus e_{14}$	$x_{25} \oplus e_{19}$	$x_{26} \oplus e_{24}$	$x_{27} \oplus e_{29}$	$x_{28} \oplus e_{34}$
$x_{29} \oplus e_5$	$x_{30} \oplus e_{10}$	$x_{31} \oplus e_{15}$	$x_{32} \oplus e_{20}$	$x_{33} \oplus e_{25}$	$x_{34} \oplus e_{30}$	$x_{35} \oplus e_{35}$

Figura 6.4: *Desentrelaçador de bloco.*

6.1.3 O calculador de LLR

O índice temporal k utilizado na seção anterior é agora dividido nos índices i e j, de modo que $k = j + (i-1) \cdot I_d$, em que $1 \leq i \leq I_d$ e $1 \leq j \leq N$. Portanto, $x(i,j)$, $y(i,j)$ e $e(i,j)$ representam o símbolo transmitido, o símbolo recebido e o erro no k-ésimo uso do canal, respectivamente.

O calculador de LLR é o principal elemento do sistema DFD para códigos LDPC. Ele tem a função de processar a informação recebida do canal $\mathbf{y}$ e a informação realimentada sobre a palavra anteriormente decodificada. Os LLRs calculados são fornecidos ao algoritmo SP que os utilizam no passo de iniciação como $L(y|x)$. A Figura 6.5 ilustra o funcionamento do calculador de LLR descrito nesta seção.

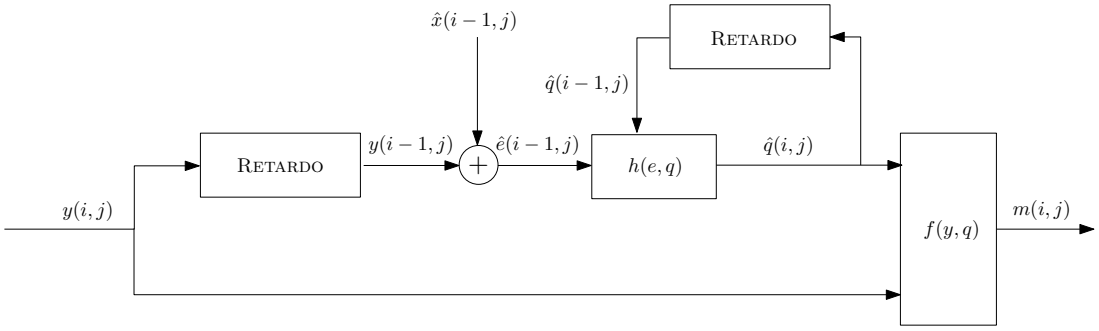


Figura 6.5: *Calculador de LLR.*

Seja $\mathbf{e}(i,j) = [e(i-1,j), \cdots, e(i,j)]$ uma seqüência de erros consecutivos na coluna acima da (i,j)-ésima posição da matriz do desentrelaçador. O logaritmo da razão de verossimilhança para o símbolo recebido $y(i,j) = x(i,j) \oplus e(i,j)$ é definido como

$$m(i,j) = f(y,q) \triangleq \begin{cases} \log \frac{q(i,j)}{1-q(i,j)}, & \text{se } y(i,j) = 1; \\ \log \frac{1-q(i,j)}{q(i,j)}, & \text{se } y(i,j) = 0, \end{cases} \tag{6.14}$$

Em que $q(i,j) \triangleq P\big(e(i,j) = 1 | \mathbf{e}(i,j)\big)$. Este LLR leva em consideração a memória latente do canal já que considera a correlação dos erros nas colunas do desentrelaçador. Faz-se então o uso das relações recursivas para $q(i,j)$ obtidas anteriormente:

$$q(i,j) = h(e,q) \triangleq \begin{cases} \beta + \frac{\mu(q(i-1,j)-g)b}{q(i-1,j)}, & \text{se } e(i-1,j) = 1; \\ \beta + \frac{\mu(q(i-1,j)-g)(1-b)}{1-q(i-1,j)}, & \text{se } e(i-1,j) = 0, \end{cases} \tag{6.15}$$

em que $\beta \triangleq g + Q(b-g)$. Os LLRs $m(i,j)$, $1 \leq j \leq N$ são passados para o decodificador SP que gera uma estimativa $\hat{\mathbf{x}}$ da palavra-código transmitida. Como o erro do canal anterior não

é conhecido pelo DFD, é necessário estimá-lo fazendo $\hat{e}(i-1,j) = \hat{x}(i-1,j) \oplus y(i-1,j)$ que é usado em 6.15 para calcular uma *estimativa* $\hat{q}(i,j)$ da probabilidade $q(i,j)$.

Ao utilizar os LLRs $m(i,j)$ como entradas para o algoritmo SP, espera-se um melhor desempenho por incorporar a memória do canal. É importante observar que os LLRs utilizados são aproximações e são atualizados a cada bloco de N símbolos a ser decodificado. Portanto, é necessário analisar a sensibilidade do algoritmo SP em relação à propagação de erros por utilizar LLRs com valores aproximados. Um outro fator a ser estudado é se é ou não necessário *treinar* o decodificador utilizando uma seqüencia conhecida de I_p blocos de N símbolos transmitidos para estimar corretamente os erros do canal e obter LLRs mais confiáveis ao custo de uma perda em taxa de transmissão.

6.2 Algoritmo SP para o canal GE

Supondo que o entrelaçamento de palavras de um código LDPC transmitidas por um canal GE seja suficientemente longo para considerá-lo um BSC. As mensagens iniciais do algoritmo SP são os LLRs denotados por $L(y_n|x_n)$, $n = 1, \cdots, N$. Como mencionado anteriormente, estas mensagens são calculadas a partir do modelo do canal. Sendo q a probabilidade de transição de um canal BSC, calcula-se seguinte o logaritmo da razão de verossimilhança por

$$L(x|y) = \log\left(\frac{P(x=0|y)}{P(x=1|y)}\right) = \log\left(\frac{P(y|x=0)\cdot P(x=0)}{P(y|x=1)\cdot P(x=1)}\right).$$

Assumindo que $P(x=0) = P(x=1)$,

$$L(x|y) = \log\left(\frac{P(y|x=0)}{P(y|x=1)}\right).$$

Portanto,

$$L(y|x) = \begin{cases} \log\frac{q}{1-q}, & \text{se } y = 1; \\ \log\frac{1-q}{q}, & \text{se } y = 0. \end{cases} \tag{6.16}$$

Observa-se que as Equações 6.16 e 6.14 são essencialmente iguais. Portanto, ao utilizar os LLRs calculados a partir de 6.14, a memória do canal é considerada mantendo as mesmas regras de atualização para o algoritmo SP descrito no capítulo 3. O único aumento de complexidade do sistema é introduzido pelo calculador de métrica que requer alguns elementos de memória e um somador binário antes de calcular o logaritmo das razões de verossimilhança.

Exemplo 6.3 (Decodificando no canal GE) *Para ilustrar o funcionamento do sistema DFD para decodificação de códigos LDPC foi considerado um canal GE com probabilidade de erro média* $P(e_k) = 0.05$, $g = 10^{-3}$, $b = 0.4$, $q = 0.02$ *e* $Q = 2.809 \cdot 10^{-3}$. *Este canal apresenta surtos de comprimento médio igual a* $\lambda = 50$. *Foi utilizado um código LDPC* $(3, 6)$*-regular de comprimento 256 com 128 equações de paridade e taxa* $R = 1/2$, *um entrelaçador de bloco de profundidade* $I_d = 250$ *e o decodificador SP com 10 iterações. O padrão de erros obtido na saída do canal GE é mostrado na Figura 6.6.*

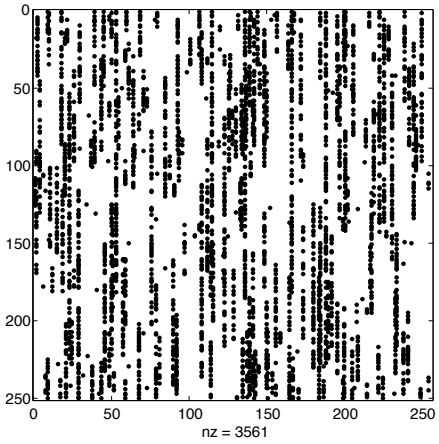


Figura 6.6: *Padrão de erros no desentrelaçador.*

A Figura 6.7 compara o erro residual no esquema que utiliza o DFD e o esquema que utiliza apenas o entrelaçamento. Existe uma redução muito grande nos erros quando o esquema com DFD é utilizado.

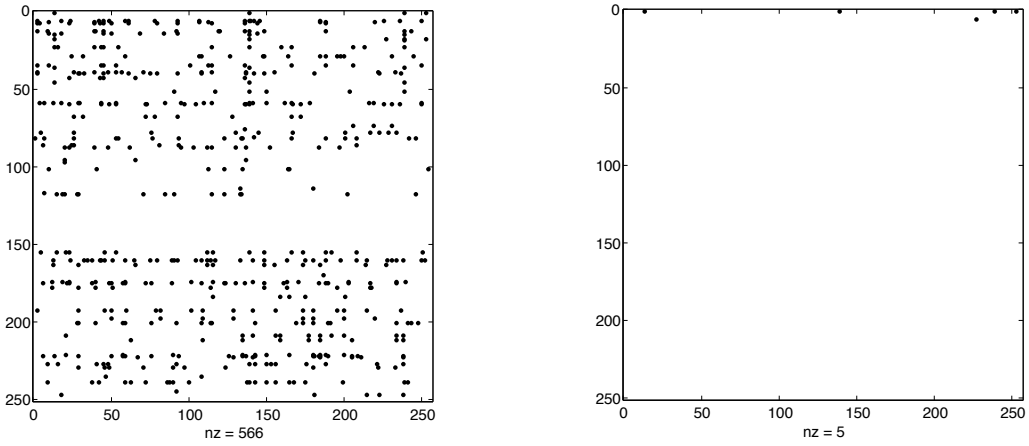


Figura 6.7: *Erro residual no esquema sem DFD e com DFD.*

capítulo 7

Conclusões

Este capítulo descreve brevemente as contribuições desta dissertação, apresenta algumas conclusões e propõe alguns trabalhos para futuras investigações.

7.1 Sumário de contribuições

O principal objetivo desta dissertação é relatar um estudo realizado sobre códigos LDPC e o algoritmo de decodificação SP. Apesar da abundante literatura disponível, a notação é muito dispersa. Foi realizado um esforço em apresentar uma notação uniforme e uma seqüência lógica na apresentação destes assuntos.

Primeiramente, foi apresentada a representação de códigos de bloco lineares baseada em grafos. O algoritmo SP foi estudado em sua forma integral e a sua principal simplificação prática, o algoritmo MIN-SUM foi derivado. Em seguida, a estrutura dos códigos LDPC foi discutida. Apresentou-se exemplos de códigos regulares e irregulares e também foram apresentadas algumas técnicas de projeto. A partir de simulações computacionais, o desempenho de diversos códigos LDPC de comprimento curto a médio foram analisados.

Após a análise experimental, partiu-se para a investigação analítica. A técnica de análise assintótica *density evolution* foi derivada em forma integral. Verificou-se a dificuldade de implementação da *density evolution*, o que motivou o estudo de técnicas aproximadas que viabilizassem a sua implementação computacional. Uma destas técnicas, a análise por aproximação Gaussiana, também foi derivada. A aproximação Gaussiana permitiu o cálculo de limiares de decodificação para diversos conjuntos de códigos LDPC regulares e ajudou a

visualizar a convergência no processo de decodificação graficamente. A análise assintótica também pode ser utilizada para obter distribuições de grau de códigos cujo desempenho se aproxima do limite de Shannon.

Finalmente, uma contribuição foi feita na direção de sugerir um novo esquema de decodificação para códigos LDPC em canais com memória. O esquema proposto utiliza uma idéia relativamente antiga sugerida em [24] e que foi analisada em [36] para o caso de códigos convolucionais. A decodificação para códigos LDPC funciona tão bem que torna a determinação de seu desempenho por curvas de taxa de erro de bit muito difícil. Algumas simulações preliminares foram realizadas e verificou-se que mesmo com um baixo número de iterações do algoritmo SP, a taxa de erro de bit residual é muito menor do que a solução que consi dera apenas o entrelaçamento de palavras-código e LLRs usuais. Em princípio, credita-se o sucesso dos códigos LDPC nesta abordagem pela sua estrutura semi-aleatória que emula um entrelaçamento interno de seus símbolos [35]. Outra possibilidade, é que a operação do algoritmo SP acomoda muito bem o processamento adaptativo imposto pelo calculador de LLRs. Como o aumento de complexidade é muito pequeno em relação ao do algoritmo SP original, é importante analisar cuidadosamente este esquema e estudar seus detalhes. Uma questão aqui levantada é se com apenas ferramentas desenvolvidas para canais sem memória é possível atingir um desempenho próximo da capacidade de um canal com memória, amplamente utilizado para modelar um grande número de situações práticas.

7.2 Trabalhos futuros

Uma continuação natural deste trabalho é o estudo de técnicas de codificação eficiente de códigos LDPC [37], assunto que não foi discutido nesta dissertação. Um outro tópico de investigação é o projeto de matrizes de verificação de paridade irregulares utilizando o algoritmo PEG e a avaliação de seu desempenho por meio de simulações. A implementação de um *software* que utilize a *density evolution* ou versões aproximadas, como a aproximação Gaussiana, para obter distribuições de grau que definam conjuntos de códigos LDPC com limiares de decodificação próximos da capacidade do canal RAGB, seria uma importante contribuição. O estudo de técnicas de análise que consideram conjuntos de códigos LDPC de comprimento finito e a análise por tabelas EXIT [38] são pertinentes. Quanto ao esquema DFD para o canal Gilbert-Elliott, seria interessante determinar curvas de desempenho relacionando taxa de erro de bit, comprimento médio de surtos, grau de entrelaçamento, probabilidade

média de erro e outros parâmetros do canal. A comparação deste esquema com o esquema proposto por Eckford em [19] é necessária já que este é atualmente o melhor sistema de decodificação de códigos LDPC no canal GE. Por fim, sugere-se estudar a possibilidade de definir uma técnica de análise semelhante à *density evolution* para canais GE usando DFD e projetar códigos cujo desempenho se aproxime da capacidade do canal GE.

Referências

[1] D. J. C. Mackay, Good error correcting codes based on very sparse matrices, *IEEE Transacations on Information Theory*, v. 45, n. 2, p. 399–431, Março, 1999.

[2] R. G. Gallager, **Low Density Parity Check Codes**. MIT Press, 1963.

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

[4] E. Berlekamp, R. McEliece, & H. van Tilborg, On the inherent intractability of certain coding problems, *IEEE Transactions on Information Theory*, v. 24, n. 3, p. 384–386, Maio, 1978.

[5] C. Berrou, A. Glavieux, & P. Thitimajshima, Near Shannon limit error-correcting coding and decoding: Turbo codes, In: **1993 IEEE International Conference on Communications**, Genebra, Suíça, 1993, p. 1064–1070.

[6] C. Heegard & S. B. Wicker, **Turbo Coding**. Kluwer Academic Publishers, 1999.

[7] R. J. McEliece, D. J. C. Mackay, & J.-F. Cheng, Turbo decoding as an instance of Pearl's belief propagation algorithm, *IEEE Journal on Selected Areas in Communications*, v. 16, n. 2, p. 140–152, Fevereiro, 1998.

[8] D. J. C. Mackay & R. M. Neal, Near Shannon limit performance of low density parity check codes, *Electronics Letters*, v. 32, p. 1645–1646, 1996.

[9] M. Sipser & D. A. Spielman, Expander codes, *IEEE Transactions on Information Theory*, v. 42, n. 6, p. 1710–1722, Novembro, 1996.

[10] N. Wiberg, Codes and decoding on general graphs, Tese, Linköping University, 1996.

[11] R. M. Tanner, A recursive approach to low complexity codes, *IEEE Transactions on Information Theory*, v. 27, p. 533–547, Setembro, 1981.

[12] F. R. Kschischang, B. J. Frey, & H.-A. Loeliger, Factor graphs and the sum-product algorithm, *IEEE Transactions on Information Theory*, v. 47, n. 2, p. 498–519, Fevereiro, 2001.

[13] S. Lin & D. J. C. Jr, **Error Control Coding**. Prentice Hall, 2004.

[14] T. J. Richardson & R. L. Urbanke, The capacity of low-density parity-check codes under message-passing decoding, *IEEE Transactions on Information Theory*, v. 47, n. 2, p. 599–618, Fevereiro, 2001.

[15] S.-Y. Chung, T. J. Richardson, & R. L. Urbanke, Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation, *IEEE Transactions on Information Theory*, v. 47, n. 2, p. 657–670, Fevereiro, 2001.

[16] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, & D. A. Spielman, Improved low-density parity-check codes using irregular graphs, *IEEE Transactions on Information Theory*, v. 47, n. 2, p. 924–934, Fevereiro, 2001.

[17] T. J. Richardson, M. A. Shokrollahi, & R. L. Urbanke, Design of capacity approaching irregular low-density parity-check codes, *IEEE Transactions on Information Theory*, v. 47, n. 2, p. 619–637, Fevereiro, 2001.

[18] S.-Y. Chung, J. G. D. Forney, T. J. Richardson, & R. L. Urbanke, On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit, *IEEE Communications Letters*, v. 5, n. 2, p. 58–60, Fevereiro, 2001.

[19] A. W. Eckford, Low-density parity-check codes for Gilbert-Elliott and Markov-modulated channels, Tese, University of Toronto, 2004.

[20] R. G. Gallager, **Information Theory and Reliable Communication**. John Wiley and Sons, 1968.

[21] E. N. Gilbert, Capacity of a burst-noise channel, *Bell System Technical Journal*, v. 39, p. 1253–1265, Setembro, 1960.

[22] E. O. Elliott, Estimates of error rates for codes on burst-noise channels, *Bell System Technical Journal*, v. 42, p. 1977–1997, Setembro, 1963.

[23] C. Pimentel, Enumeration techniques for finite state channels, Tese, University of Waterloo, 1996.

[24] M. Mushkin & I. Bar-David, Capacity and coding for the Gilbert-Elliott channels, *IEEE Transactions on Information Theory*, v. 35, n. 6, p. 1277–1290, Novembro, 1989.

[25] F. R. Kschischang, Codes defined on graphs, *IEEE Communications Magazine*, v. 41, n. 8, p. 118–125, Agosto, 2003.

[26] D. J. C. Mackay, **Information Theory, Inference, and Learning Algorithms**. Cambridge University Press, 2003.

[27] C. B. Schlegel & L. C. Pérez, **Trellis and Turbo Coding**, ser. IEEE Series on Digital & Mobile Communication, J. B. Anderson, Ed. Wiley Interscience, 2004.

[28] J. Hagenauer, E. Offer, & L. Papke, Iterative decoding of binary block and convolutional codes, *IEEE Transactions on Information Theory*, v. 42, p. 429–445, Março, 1996.

[29] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier, & X. Y. Hu, Reduced-complexity decoding of LDPC codes, *IEEE Transactions on Communications*, v. 53, n. 8, p. 1288–1299, Agosto, 2005.

[30] D. J. C. Mackay, Online database of low-density parity-check codes, Website, acessado em abril de 2006. [Online]. Disponível: http://www.inference.phy.cam.ac.uk/mackay/codes/data.html

[31] W. E. Ryan, **CRC Handbook for Coding and Signal Processing for Recording Systems**. CRC Press, 2004, cap. An Introduction to LDPC Codes.

[32] X.-Y. Hu, E. Eleftheriou, & D. M. Arnold, Regular and irregular progressive edge growth Tanner graphs, *IEEE Transactions on Information Theory*, v. 51, n. 1, p. 386–398, Janeiro, 2005.

[33] S. Litsyn & V. Shevelev, On ensembles of low-density parity-check codes: asymptotic distance distributions, *IEEE Transactions on Information Theory*, v. 48, n. 4, p. 887–908, Abril, 2002.

[34] M. Ardakani, Efficient analysis, design and decoding of low-density parity-check codes, Tese, University of Toronto, 2004.

[35] J. Hou, P. H. Siegel, & L. B. Milstein, Performance analysis and code optimization of low density parity-check codes on rayleigh fading channels, *Selected Areas in Communications, IEEE Journal on*, v. 19, n. 5, p. 924–934, 2001.

[36] C. Pimentel & L. C. Rêgo, Analysis of soft decision decoding of interleaved convolutional codes over burst channels, In: **IEEE Wireless Communications and Networking Conference**, v. 3, 1999, p. 1090–1094.

[37] T. J. Richardson & R. L. Urbanke, Efficient encoding of low-density parity-check codes, *Information Theory, IEEE Transactions on*, v. 47, n. 2, p. 638–656, Fevereiro, 2001.

[38] S. ten Brink, Convergence of iterative decoding, *IEE Electronics Letters*, v. 35, p. 806–808, Maio, 1999.

Sobre o Autor

O autor nasceu no Recife, Pernambuco, no dia 31 de outubro de 1980. Estudou no Colégio Equipe de 1982 a 1997. Se formou em Engenharia Elétrica, modalidade Eletrônica, pela Universidade Federal de Pernambuco em 2003. É membro da Sociedade Brasileira de Telecomunicações e do *Institute of Electrical and Electronics Engineers*. Seus interesses de pesquisa incluem códigos para controle de erros, processamento digital de sinais, teoria da informação, combinatória e matemática discreta.

Endereço: Praça Prof. Fleming, 145/301
Jaqueira
Recife – PE, Brasil
C.E.P.: 52.050 – 180

e-mail: `mmv@ee.ufpe.br`

Esta dissertação foi diagramada usando LaTeX 2$_\varepsilon$[1] pelo autor.

[1] LaTeX 2$_\varepsilon$ é uma extensão do LaTeX. LaTeX é uma coleção de macros criadas por Leslie Lamport para o sistema TeX, que foi desenvolvido por Donald E. Knuth. TeX é uma marca registrada da Sociedade Americana de Matemática ($\mathcal{AMS}$). O estilo usado na formatação desta dissertação foi escrito por Dinesh Das, Universidade do Texas. Modificado em 2001 por Renato José de Sobral Cintra, Universidade Federal de Pernambuco, e em 2005 por André Leite Wanderley.