Códigos de Verificação de Paridade de Baixa Densidade (LDPC)

O Papel da Teoria de Grafos e Estruturas de Dados na Correção de Erros

Aleksander Da Silva Toth
Arthur Gabriel
Gabriel Giovanini
Isabella Harumi

UTFPR–Apucarana

27 de novembro de 2025

Conteúdo

1 Introdução: O Desafio da Comunicação Digital

Um sistema de comunicação digital tem como objetivo fundamental a transmissão de informação de uma fonte para um destino (Lathi, 2006; Medeiros, 2016). Este processo, aparentemente simples, envolve uma cadeia de etapas complexas: a informação (seja analógica, como a voz, ou já digital) é amostrada, quantizada e convertida em bits (1s e 0s) (Lathi, 2006). Estes bits são então processados por um **codificador de canal**, modulados para se adaptarem ao meio físico, e finalmente transmitidos através de um **canal** (Medeiros, 2016).

[Image of digital communication system block diagram]

[Image of block diagram of a digital communication system]

O "canal"é o elo mais fraco deste sistema. Seja ele um cabo de rede, uma fibra ótica ou o ar (para transmissões Wi-Fi e 5G), ele está inevitavelmente sujeito a imperfeições. O problema central que todo sistema de comunicação enfrenta é o **ruído** (Medeiros, 2016; Lathi, 2006).

O ruído pode "virar"bits, fazendo com que um ‘1‘ enviado seja recebido como ‘0‘. Em sistemas modernos de alta velocidade, como 5G e Wi-Fi 6, a fiabilidade da transmissão face ao ruído não é um luxo, mas uma necessidade. É aqui que entra a **Codificação de Canal para Correção de Erros (FEC)**.

Este trabalho explora a espinha dorsal dos sistemas FEC modernos: os **Códigos de Verificação de Paridade de Baixa Densidade (LDPC)**. Analisamos não só a sua teoria matemática, mas focamos em como a sua implementação prática depende diretamente de conceitos fundamentais de **Estruturas de Dados** e **Teoria de Grafos** (**Cormen2012**; **Szwarcfiter2006**). Demonstramos como a eficiência dos LDPC, que lhes permite operar próximos ao Limite de Shannon, é inseparável da escolha de estruturas de dados eficientes (como listas de adjacência) em vez de simples arrays.

—

2 O Problema da Transmissão: Ruído e Seus Efeitos

Para entender por que os códigos LDPC são necessários, devemos primeiro caracterizar o "inimigo": o ruído.

2.1 O Inimigo: O Que é Ruído?

Em telecomunicações, ruído é definido como qualquer sinal elétrico indesejado que se sobrepõe ao sinal de informação, degradando a sua qualidade (Medeiros, 2016). As suas fontes são vastas e podem ser categorizadas em:

- **Ruído Térmico (Ruído de Johnson-Nyquist):** O ruído "inevitável"gerado pela agitação térmica aleatória dos eletrões em todos os componentes eletrónicos (resistores, cabos, amplificadores). É a base contra a qual todos os sistemas lutam (Medeiros, 2016).
- **Interferência (EMI/RFI):** Ruído gerado pelo homem, proveniente de motores elétricos, luzes fluorescentes, transmissores de rádio vizinhos ou até mesmo um forno micro-ondas. Em redes Wi-Fi, esta é uma fonte de erro dominante (Medeiros, 2016).

- **Diafonia (Crosstalk):** Ocorre em meios cabulados (como cabos de rede) quando o sinal de um par de fios interfere magneticamente com um par vizinho (Medeiros, 2016).

2.2 O Impacto: Canais Analógicos vs. Digitais

O efeito do ruído é fundamentalmente diferente dependendo do tipo de transmissão:

- **Em Transmissão Analógica (ex: Rádio AM/FM antigo):** O ruído adiciona-se diretamente ao sinal. O resultado é uma degradação "graciosa": o ouvinte ouve um "chiado"(hiss) que piora lentamente à medida que o sinal enfraquece (Lathi, 2006; Medeiros, 2016).
- **Em Transmissão Digital (ex: 5G, Wi-Fi):** O ruído não causa "chiado"; ele causa **erros de bit** (Bit Error Rate - BER). O recetor tem de tomar uma decisão: o pulso recebido foi um ‘0‘ ou um ‘1‘? O ruído pode empurrar o sinal para o lado errado da decisão.

Isto leva ao famoso **"Efeito Penhasco"(Cliff Effect)** da transmissão digital. Para um nível de ruído baixo, o sistema de correção de erros (FEC) corrige tudo e a imagem/som é perfeita (100%). Mas se o ruído aumentar ligeiramente para além do que o FEC consegue suportar, o sistema falha catastroficamente (a imagem congela, a chamada cai).

O objetivo dos Códigos LDPC é, portanto, empurrar esse "penhasco"o mais longe possível, permitindo uma comunicação robusta mesmo em canais com muito ruído.

—

3 Fundamentação Teórica: A Solução LDPC

Para combater os erros de bit induzidos pelo ruído, a estratégia de Codificação de Canal (FEC) adiciona **redundância** aos dados. Em vez de enviar apenas a mensagem $\mathbf{u}$, enviamos uma "palavra-código"$\mathbf{v}$ mais longa. Os Códigos LDPC são uma das formas mais eficientes de o fazer.

3.1 Definição e Estrutura (Matriz H)

Um código LDPC é um código de bloco linear definido por sua matriz de verificação de paridade $\mathbf{H}$, uma matriz $m \times n$ com baixa densidade de ’1s’ (Gallager, 1962). Uma palavra-código $\mathbf{v}$ (de n bits) é válida se, e somente se, satisfaz a equação de paridade:

$$\mathbf{H}\mathbf{v}^T = \mathbf{0} \pmod 2 \tag{1}$$

A esparsidade (baixa densidade) de $\mathbf{H}$ é a sua característica definidora.

3.2 A Ponte: Teoria de Grafos e Estrutura de Dados

A Matriz H não é apenas uma ferramenta matemática; ela é uma **estrutura de dados** que descreve um grafo. Esta é a ligação crucial entre os mundos das Telecomunicações e da Ciência da Computação.

A Matriz H é a representação de um **Grafo de Tanner**. Um Grafo de Tanner é um grafo bipartido com dois conjuntos de nós: n "Nós de Variável"(V-Nodes, os bits) e m "Nós de Checagem"(C-Nodes, as equações de paridade) (**Tanner1981**; **YairMZ_BP**).

A Matriz H é, por definição, a **Matriz de Adjacência** deste grafo (**Cormen2012**; **Szwarcfiter2006**). No entanto, o "LD"(Baixa Densidade) em LDPC significa que esta matriz é **esparsa** — talvez 99% dela contenha zeros.

Armazenar esta matriz na memória como um 'array' 2D padrão (uma Matriz de Adjacência) seria um desperdício catastrófico de recursos:

- **Complexidade de Memória (Ruim):** $O(m \times n)$, ou $O(n^2)$ para um código de taxa fixa. Para um código 5G com $n = 10000$, isto exigiria 100 milhões de posições de memória, a maioria para armazenar zeros inúteis (**Cormen2012**).
- **Complexidade de Cálculo (Ruim):** Qualquer algoritmo que iterar sobre esta matriz para encontrar os '1's seria $O(n^2)$.

A solução, ensinada em Estrutura de Dados, é usar uma representação de grafo otimizada para esparsidade: a **Lista de Adjacência** (**Szwarcfiter2006**; **Cormen2012**).

[Image of adjacency matrix vs adjacency list]

- **Complexidade de Memória (Boa):** $O(n+e)$, onde e é o número de arestas (o número de '1's na matriz H). Como e é pequeno (baixa densidade), a memória é $O(n)$.
- **Complexidade de Cálculo (Boa):** O algoritmo de decodificação pode agora aceder diretamente aos vizinhos de um nó em tempo linear, levando a uma complexidade de decodificação global de $O(n)$ por iteração (**Cormen2012**).

3.3 Decodificação Iterativa: O Algoritmo Soma-Produto (SPA)

Esta eficiência de estrutura de dados é o que torna o algoritmo de decodificação **Belief Propagation (BP)** (ou Soma-Produto) viável (**MIT_SPA**).

Este algoritmo funciona precisamente como descrito pelas Listas de Adjacência: é um algoritmo de **passagem de mensagens** onde os nós no Grafo de Tanner trocam "crenças"(probabilidades) sobre os valores dos bits ao longo das arestas (os '1's da matriz H) (**YairMZ_BP**).

1. **V-Node → C-Node:** V-Nodes enviam suas "crenças"(probabilidades) sobre seus próprios valores.
2. **C-Node → V-Node:** C-Nodes calculam a probabilidade de a equação de paridade ser satisfeita e enviam mensagens "extrínsecas"de volta.

O algoritmo opera no domínio logarítmico (LLR) e repete estas passagens até convergir para uma palavra-código válida ($\mathbf{Hv}^T = \mathbf{0}$) ou atingir um limite de iterações.

3.4 Otimizações (MSA) e Codificação (Back-Substitution)

A operação "Produto"do SPA é computacionalmente cara. O **Algoritmo Min-Sum (MSA)** é uma aproximação de baixa complexidade que substitui a operação por uma simples busca pelo mínimo (**FPGA_LDPC**).

Embora a decodificação seja facilitada pela **H** esparsa, a matriz geradora **G** correspondente é tipicamente densa (**Richardson2001_Encoding**). A solução prática é estruturar **H** (ex: QC-LDPC) para que a codificação tenha complexidade linear ($O(n)$) via **"back-substitution"** (**Tavildar_80211n**; **Myers_WiMAX**).

—

4 Resultados Experimentais (Simulação)

4.1 Ambiente de Simulação

A simulação foi conduzida em Python 3.x, utilizando a biblioteca 'ldpc' (de quantumgizmos) (**QuantumGizmos_LDPC**). Esta biblioteca fornece implementações otimizadas (em C++/Cython) de decodificadores BP (incluindo SPA) e funções auxiliares para manipulação de matrizes de paridade.

4.2 Fluxo da Simulação (Canal BSC)

O script (Seção 5) simula um fluxo de comunicação completo:

1. **Construção:** Uma matriz **H** de um código de Hamming (7,4) é usada como exemplo. A matriz geradora **G** é derivada de **H**.
2. **Codificação:** Uma mensagem aleatória **u** é gerada e codificada em $\mathbf{v} = \mathbf{uG} \pmod 2$.
3. **Canal:** Um Canal Binário Simétrico (BSC) é simulado introduzindo ruído **e** com uma taxa de erro de bit (BER) pré-definida. A palavra recebida é $\mathbf{y} = \mathbf{v} \oplus \mathbf{e}$.
4. **Decodificação:** A palavra **y** é convertida em LLRs de entrada. O decodificador 'BpDecoder' (usando o método 'product_sum') (**LDPC_Docs**) é invocado para estimar a palavra-código original, $\hat{\mathbf{v}}$.
5. **Verificação:** A mensagem decodificada $\hat{\mathbf{u}}$ é extraída de $\hat{\mathbf{v}}$ e comparada com **u**.

4.3 Verificação da Decodificação

A simulação verifica dois pontos:

1. **Síndrome da Palavra Decodificada:** $\mathbf{H}\hat{\mathbf{v}}^T$ deve ser $\mathbf{0}$.
2. **Recuperação da Mensagem:** A mensagem original **u** deve ser idêntica à mensagem decodificada $\hat{\mathbf{u}}$.

Para uma análise de desempenho completa (curva BER vs. E_b/N_0), seria necessário executar esta simulação em um loop Monte Carlo, variando a taxa de erro (ou E_b/N_0 para um canal AWGN) e contando os erros de bit.

Placeholder para gráfico: BER vs. SNR

Figura 1: Curva de desempenho simulada (BER) vs. Relação Sinal-Ruído (E_b/N_0).

—

5 Código Python Utilizado

O script a seguir implementa a simulação de ponta a ponta descrita na Seção 4. Requer a instalação da biblioteca 'ldpc' ('pip install ldpc').

Listing 1: Script Python para simulação de LDPC em Canal BSC

```python
# --- C digo de Demonstra  o Completo (Ponta-a-Ponta) ---
#
# Requer: pip install ldpc
# Baseado nos exemplos das bibliotecas [16, 17] e conceitos [18, 12]
# Mapeamento para BibTeX:
# [16] -> QuantumGizmos_LDPC
# [17] -> LDPC_Docs
# [18] -> Richardson2001_Design
# [12] -> MIT_SPA
# ------------------------------------------------------------

import numpy as np
import ldpc.codes # Para construir H (ref: QuantumGizmos_LDPC)
from ldpc.bp_decoder import BpDecoder # Decodificador BP (ref: LDPC_Docs)
from ldpc.utils import construct_generator_matrix # Para codifica  o (ref: LDPC_Docs)

def executar_simulacao_ldpc():
    """
    Executa uma simula  o ponta-a-Ponta de um c digo LDPC (Hamming)
    num canal BSC.
    """

    # --- 1. Constru  o do C digo ---
    print("--- 1. Constru  o do C digo ---")
    # Usamos um c digo de Hamming (3) - um c digo LDPC pequeno e regular
    #    um c digo (7, 4) - 4 bits de msg, 3 de paridade, 7 no total
    H = ldpc.codes.hamming_code(3)
    n_bits = H.shape[1]
    n_paridade = H.shape[0] # n_paridade    n-k, ou H.shape[0]
    n_msg = n_bits - n_paridade

    print(f"Matriz H (m={n_paridade}, n={n_bits}): \n{H}")

    # Para codificar (v=uG), precisamos da Matriz Geradora G
    # (ref: Richardson2001_Design, LDPC_Docs)
    # Esta fun  o converte H para uma G sistem tica
    G, k = construct_generator_matrix(H, systematic=True)
    print(f"\nMatriz G (k={k}, n={G.shape[1]}): \n{G}")
    # Verifica  o de k
    assert k == n_msg

    # --- 2. Codifica  o ---
    print("\n--- 2. Codifica  o ---")
    # Gerar uma mensagem aleat ria de k bits
    mensagem = np.random.randint(0, 2, n_msg)
    print(f"Mensagem original (k={n_msg}): {mensagem}")

    # Codificar: v = uG (mod 2) (ref: Richardson2001_Design)
    palavra_codigo = mensagem @ G % 2
    print(f"Palavra-c digo (n={n_bits}): {palavra_codigo}")
```

```
51
52      # Verifica  o: Um codeword v lido deve ter s ndrome zero (Hv^T =
            0)
53      sindrome_tx = H @ palavra_codigo % 2
54      print(f"S ndrome da palavra-c digo (deve ser zero): {sindrome_tx}")
55
56
57      # --- 3. Simula  o do Canal (BSC) ---
58      print("\n--- 3. Simula  o do Canal (BSC) ---")
59      taxa_erro_canal = 0.05
60      # Gerar ru do (Canal Bin rio Sim trico - BSC)
61      ruido = (np.random.rand(n_bits) < taxa_erro_canal).astype(int)
62
63      # A palavra recebida    o codeword + ru do (mod 2)
64      palavra_recebida = (palavra_codigo + ruido) % 2
65      print(f"Palavra recebida c/ ru do: {palavra_recebida}")
66
67      n_erros = ruido.sum()
68      print(f"Erros introduzidos pelo canal: {n_erros}")
69
70      # --- 4. Decodifica  o (BP/Soma-Produto) ---
71      print("\n--- 4. Decodifica  o (BP/Soma-Produto) ---")
72
73      # O decodificador BP/SPA espera LLRs (Log-Likelihood Ratios), n  o
            bits
74      # Convertemos os bits 0/1 recebidos em LLRs.
75      # 0 -> +L (confiante que    0)
76      # 1 -> -L (confiante que    1)
77      # Assumimos LLRs "fortes" para esta demo
78      confianca_llr = 10.0
79      llrs_canal = (1 - 2 * palavra_recebida.astype(float)) * confianca_llr
80      print(f"LLRs de entrada (arredondado): {np.round(llrs_canal, 1)}")
81
82      # Inicializar o decodificador BP (Soma-Produto) (ref: MIT_SPA,
            LDPC_Docs)
83      decoder = BpDecoder(
84          H,
85          error_rate=taxa_erro_canal,
86          bp_method='product_sum', % Algoritmo Soma-Produto (SPA)
87          max_iter=20 % M ximo de itera  es
88      )
89
90      # Decodificar os LLRs recebidos
91      palavra_decodificada_bits = decoder.decode(llrs_canal)
92
93      # O BpDecoder.decode() retorna os bits 0/1 diretamente
94      print(f"Palavra-c digo decodificada: {palavra_deocdificada_bits}")
95
96      # Extrair a mensagem decodificada (os primeiros k bits)
97      # Em formato sistem tico, a mensagem s  o os primeiros k bits
98      mensagem_decodificada = palavra_decodificada_bits[:n_msg]
99      print(f"Mensagem decodificada: {mensagem_decodificada}")
100
101
102     # --- 5. Verifica  o ---
103     print("\n--- 5. Verifica  o ---")
104     # Verificar se a palavra-c digo decodificada    v lida
105     sindrome_rx = H @ palavra_decodificada_bits % 2
```

```
106     print(f"S ndrome da palavra decodificada (deve ser zero): {
            sindrome_rx}")
107
108     # Verificar se a mensagem foi recuperada
109     sucesso = np.array_equal(mensagem, mensagem_decodificada)
110     print(f"\nDECODIFICA  O BEM-SUCEDIDA: {sucesso}")
111
112 # --- Ponto de Entrada Principal ---
113 if __name__ == "__main__":
114     executar_simulacao_ldpc()
```

---

6 Conclusões

A análise teórica e a simulação prática confirmam o poder e a viabilidade dos códigos LDPC. As principais conclusões deste trabalho são:

- **O Problema (Ruído):** A transmissão de informação digital é fundamentalmente afetada pelo ruído do canal, que introduz erros de bit (BER) (Lathi, 2006; Medeiros, 2016).
- **A Solução (LDPC):** Os códigos LDPC combatem este problema adicionando redundância estruturada, definida pela Matriz **H** (Gallager, 1962).
- **A Eficiência (Estrutura de Dados):** A "baixa densidade"do código não é apenas um conceito matemático, mas uma propriedade de Estrutura de Dados. Ao implementar o Grafo de Tanner (**Tanner1981**) usando Listas de Adjacência (complexidade $O(n+e)$) em vez de Matrizes de Adjacência (complexidade $O(n^2)$), algoritmos de decodificação de complexidade linear, como o Soma-Produto, tornam-se viáveis (**Cormen2012**; **Szwarcfiter2006**).
- **Desempenho e Prática:** Esta combinação de teoria de codificação e implementação eficiente de algoritmos permite que os códigos LDPC operem próximos ao Limite de Shannon (Ryan, 2003), tornando-os a escolha ideal para padrões modernos de alta velocidade como 5G e DVB-S2 (**5G_Coding**; **DVB2002**).

Referências

GALLAGER, Robert. Low-Density Parity-Check Codes. *In:* 1. IRE Transactions on Information Theory. [*S. l.: s. n.*], 1962. v. 8, p. 21–28.

LATHI, B. P. **Sinais e Sistemas Lineares**. 2. ed. Porto Alegre: Bookman, 2006. ISBN 9788536305338.

MEDEIROS, Júlio César de Oliveira. **Princípios de Telecomunicações: Teoria e Prática**. 5. ed. Rio de Janeiro: Érica, 2016. ISBN 9788536507992.

RYAN, William E. An Introduction to LDPC Codes. *In:* 2ND International Symposium on Turbo Codes and Related Topics. Brest, France: [*s. n.*], 2003.