# Lab 1: Códigos de Bloco

1st Fernando Gusmão Zanchitta
*Divisão de Ciência da Computação*
*Instituto Tecnológico de Aeronáutica*
São José dos Campos, Brasil
fernando.zanchitta@ga.ta.br

2nd Guilherme Goulart Kowalczuk
*Divisão de Ciência da Computação*
*Instituto Tecnológico de Aeronáutica*
São José dos Campos, Brasil
guilherme.kowalczuk@ga.ita.br

***Abstract*—Nesse relatório, foi implementado na linguagem *Python* a simulação de um sistema de comunicação formado por um canal BSC e codificação de informação dada pelo código de Hamming com taxa $4/7$. Além disso, foi proposto um modelo baseado em Hamming com palavras-código de tamanho maior e taxa de $11/15$. Com essa modelagem, foi simulado uma transmissão de cerca de 1 milhão de bits de informação, e se comparou a probabilidade de erro de bit de informação entre os modelos. Constatou-se que ao aumentar o tamanho da palavra-código – ou diminuir a taxa do sistema – se aumenta o desempenho para baixas probabilidades $p$ de erro de bit de transmissão.**

***Index Terms*—BSC, Hamming, Telecomunicações, Códigos de Bloco, Palavra Código**

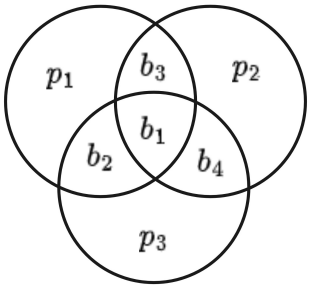


Fig. 1. Diagrama de Venn para o Código de Hamming.

## I. Introdução

Códigos de bloco são um tipo de código de correção de erro usado para codificar mensagens binárias transmitidas entre em canais de comunicação. Formalmente, constitui-se de um subconjunto $C$ de $2^K$ vetores (palavras-código) de $\{0,1\}^N$, com $K < N$. Cada palavra-código se associa a um vetor binário de tamanho $K$ diferente, referido como decodificação da palavra-código, formando um mapa $M : C \to \{0,1\}^K$. Esse mapa $M$ é construído cuidadosamente de modo que, mesmo quando há erros na transmissão dos bits da palavra-código, ainda é possível – sob condições – recuperar a palavra de informação associada.

Já um canal BSC modela um canal não-ideal de transmissão com entrada e saída binárias. Para cada bit transmitido, há um probabilidade $0 < p \leq \frac{1}{2}$ de erro na transmissão desse bit. Aqui, embora o modelo seja de canal não ideal, ainda assumimos que a probabilidade de erro $p$ é constante durante toda a operação do canal e que os eventos de transmissão são independentes entre si. Vale notar, no entanto, que sistemas reais podem violar (e efetivamente violam) esses pressupostos de alguma maneira.

Ao mesclar o modelo estatístico de um canal BSC com uma codificação infeligente de um código de bloco, pode-se modelar o comportamento de um sistema de comunicação simples, alvo desse laboratório. Nesse caso, o objetivo é minimizar a probabilidade de erro de bit de informação $P_b$, dada uma probabilidade de erro $p$ na transmissão pelo canal BSC. Um dos códigos usados para resolver esse problema é o Código de Hamming.

O laboratório envolveu o aprendizado e a implementação de um projeto envolvendo a codificação e decodificação de mensagens que passam em um canal BSC como descrito. Além de levantar a curva de probabilidade de erro de bit de informação $P_b(p)$ para o código de Hamming e para mensagem não codificada, o objetivo do laboratório é propor um código de bloco diferente do código de Hamming, e comparar as curvas $P_b(p)$ entre si.

## II. Metodologia

A implementação do projeto escrita na linguagem *Python* usando conceitos de Programação Orientada a Objetos, e possui 3 arquivos, `Channel.py`, `encoder.py`, `lab1.py`.

No primeiro arquivo (`Channel.py`), criou-se uma classe `Channel`, que serve de `Pai` das classes `BSC` e `IdealChannel`. Na `BSC` foi implementado a soma da palavra de entrada com um sinal `noise` aleatório. `IdealChannel` cria e transmite uma palavra.

No segundo arquivo (`Encoder.py`), foi criada a classe pai `Encoder`, que é pai das classes `HammingEncoder`, `NaiveEncoder`, `AlternativeEncoder`. Cada uma dessas possuí os atributos de `encode` e `decode`. No terceiro arquivo (`Lab1.py`) foi implementada a simulação com a geração de bits e geração de gráficos para comparação de Performance.

### A. Código de Hamming

Para o código de Hamming, consideramos blocos de tamanho $K = 4$, que são codificados em uma palavra-código de tamanho $N = 7$, resultando em uma taxa de bits de $\frac{4}{7}$. Na sua implementação, foi utilizada a notação algébrica considerando a matriz de geração $G \in \{0,1\}^{4\times 7}$, de modo que uma palavra $a \in \{0,1\}^K$ é codificada em $a \mapsto a \cdot G$.

Já para a decodificação, seguiu-se a orientação dada em aula. A matriz de checagem $H$ calcula a síndrome $s = r \cdot H^T$ de uma palavra recebida $r \in \{0,1\}^N$. Assim, foi construído um dicionário `errorDict` de modo a se obter a correção $e$ proposta para cada síndrome possível $s$. A palavra decodificada é então reconstruída ao fazer $c = r \oplus e$[1].

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{1}$$

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$\text{errorDict} = \begin{cases} s = [000] & \Rightarrow \ e = [0000000] \\ s = [001] & \Rightarrow \ e = [0000001] \\ s = [010] & \Rightarrow \ e = [0000010] \\ s = [011] & \Rightarrow \ e = [0001000] \\ s = [100] & \Rightarrow \ e = [0000100] \\ s = [101] & \Rightarrow \ e = [1000000] \\ s = [110] & \Rightarrow \ e = [0100000] \\ s = [111] & \Rightarrow \ e = [0010000] \end{cases} \tag{3}$$

Portanto para o *encode* de uma palavra $u$ na palavra-código $v$ foi feita a operação

$$v = (u \cdot G) \pmod 2 \tag{4}$$

cuja complexidade computacional é uma multiplicação matricial. Já para o *decode* de uma palavra recebida $r$ foram realizadas as operações

$$\begin{aligned} s &= (r \cdot H^T) \pmod 2 \\ e &= \text{errorDict}[s] \\ c &= r \oplus e \end{aligned} \tag{5}$$

cuja complexidade computacional é o de uma multiplicação matricial, uma soma bit a bit e um lookup em um dicionário previamente construído. Assim, retornamos os 4 primeiros bits do resultado $c$.

**A maior dificuldade de implementar o decodificador de Hamming foi na confecção do dicionário de mapeamento de erros, a partir da lógica por trás deste método de decodificação.**[2] Isso porque as operações de multiplicação e soma já são amplamente implementadas em *Python*. Já a construção do dicionário de erros depende da análise das síndromes geradas pelos vetores de erro com peso de Hamming mínimos.

[1]Aqui, $\oplus$ denota a operação XOR bit a bit, ou soma vetorial módulo 2.

[2]Resposta para a pergunta 1 do roteiro.

### *B. Implementação Própria*

A implementação própria envolveu estender a ideia inicial do código de Hamming. Para isso, observe que o código de Hamming pode ser interpretado em um diagrama de Venn como na Figura 1. Nesse caso, os bits de paridade $p_1$, $p_2$ e $p_3$ são escolhidos de modo a tornar a soma dos bits em cada círculo par.

Estendendo essa ideia, ao invés de trabalhar com três bits de paridade, podemos tomar mais um bit de paridade, conforme a Figura 2. Nesse caso, montando o diagrama de Venn com quatro conjuntos, conseguimos transmitir 11 bits de informação em cada bloco de 14 bits de palavra-código. Assim, nossa implementação aumenta a taxa de bits para $11/15 \approx 73\%$.

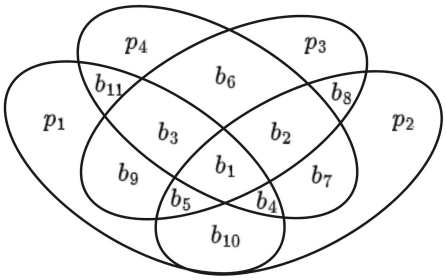


Fig. 2. Diagrama de Venn para implementação alternativa.

A partir da Figura 2 disso, seguindo um padrão lógico de soma de cada círculo resultado em 0, obtém-se as equações abaixo.

$$\begin{aligned} p_1 + b_{11} + b_9 + b_3 + b_5 + b_1 + b_{10} + b_4 &= 0 \\ p_2 + b_8 + b_2 + b_7 + b_1 + b_4 + b_5 + b_{10} &= 0 \\ p_3 + b_8 + b_6 + b_2 + b_3 + b_1 + b_5 + b_9 &= 0 \\ p_4 + b_{11} + b_3 + b_6 + b_1 + b_2 + b_4 + b_7 &= 0 \end{aligned} \tag{6}$$

A partir dessas equações, conseguimos construir as matrizes $G$ de geração da palavra-código e $H$ de checagem de erros. As matrizes para nosso código são bastante grandes, mas podem ser consultadas no.

Os algoritmos de `encode` e `decode` seguem as mesmas operações apresentadas no `Hamming`, uma vez a adição de mais uma equação ao sistema mantêm $N = 2^r - 1$, $r \geq 2$, com tamanho de palavra código $K = 2^r - 1 - r$. Isso garante um formato das matrizes Geradora e de Checagem de erro, seguindo o formato abaixo:

$$A = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

$$G = \left(I_k | A^T\right) \tag{8}$$

$$H = (A | I_{n-k}) \tag{9}$$

Dessa forma, na etapa de codificação utilizamos a Equação 4 para obter a palavra-código $v$ a partir do bloco de informação

a ser transmitida $u$. **Nesse caso, a complexidade computacional é uma multiplicação por matriz.** Já na etapa de decodificação, utilizamos a Equação 5 para obter de volta a informação a partir da palavra recebida. **Nesse caso, a complexidade computacional é uma multiplicação matricial, um lookup em um dicionário já definido e uma operação de XOR bit a bit.**[3]

Observe que essa extensão de $r \geq 2$ torna a codificação possível para comprimentos de bloco de tamanhos maiores ainda, como: 31, 63, 127. Isso torna as taxas de $\approx 0.839$, $\approx 0.905$, $\approx 0.969$. **Assim, é perfeitamente possível estender o método utilizado aqui para tamanhos arbitrariamente grandes de palavra-código, como os citados acima. Entretanto, o tamanho de bloco deve ser da forma $2^r - 1$ com $r \in \mathbb{R}$.**[4]

## III. Análises e Resultados

Foi implementado um codificador e decodificador de canal para o código de Hamming, como descrito nos tópicos acima. Além disso, foi feito um canal BSC com parâmetro $p$ de probabilidade de erro para cada bit transmitido no canal.

Após isso, foi criado um modelo de codificação/ decodificação alternativo para efeito de comparação, como descrito no Tópico B.

Para cada código (Hamming e Alternativo), fizemos a simulação da transmissão de cerca de 1 milhão de bits. Definimos um escopo de probabilidades para teste,

$$p = \begin{bmatrix} 5 \cdot 10^{-1} & 2 \cdot 10^{-1} \\ 1 \cdot 10^{-1} & 5 \cdot 10^{-2} \\ \dots & \dots \\ 1 \cdot 10^{-4} & 5 \cdot 10^{-5} \\ 2 \cdot 10^{-5} & 1 \cdot 10^{-5} \end{bmatrix} \tag{10}$$

sendo $p$ array com probabilidade de erro de bit de informação, utilizando pseudo-divisão por 2.

Com isso, mapeamos esse array $p$ em na criação de canais BSC com probabilidade p. Os canais gerados eram passados pelo processo de codificação e decodificação, e as palavras geradas eram armazenadas em outro array `Channel`.

O `Channel` entra como parâmetro em uma função que mapeia as simulações e o codificador para então ser gerado as novas palavras decodificadas.

Por fim, mapeia-se o resultado das simulações com a palavra inicialmente gerada. Esse resultado é comparado entre os diferentes tipos de codificação e foi gerado o gráfico comparativo de desempenho da probabilidade de erro de um bit.

Observa-se pela Figura 3 que o desempenho gerado pela função código `Naive` é equivalente a um código de bloco unitário, ou seja $P_b$ é atribuído com os mesmos valores de $p$, já que não há codificação além da trivial. A medida em que aumenta-se a palavra código, como nas funções `Hamming` e `Alternative` o desempenho aumenta, uma vez que os valores da probabilidade de erro no bit de informação $P_b$ diminuem em relação a probabilidade de erro na transmissão $p$.

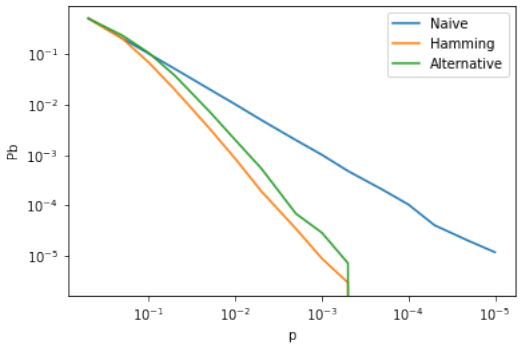


Fig. 3. Gráficos de desempenho da probabilidade de erro de um bit para algoritmos de codificação diferentes.

Comparando o resultado entre as duas ultimas funções, a diferença de desempenho na probabilidade $P_b$ se deve à diferença de tamanho entre os blocos de código e de taxa do sistema. Nesse caso, um acabou compensando o outro, de modo que nosso código *Alternative* teve um desempenho semelhante ao *Hamming*, evidenciado pelas curvas próximas na Figura 3.

Por fim, veja que não há alguns pontos plotados para o código de Hamming nem para o código alternativo para valores pequenos de $p$. Isso acontece porque o gráfico está com eixo em logaritmo, e para probabilidades de $p$ muito baixas, não houve erro de transmissão nos 1 milhão de bits de informação simulados, o que é um ótimo resultado.

## IV. Conclusão

No laboratório, pode-se realizar a implementação simulada de um sistema de comunicações simples, composta por um canal BSC e diferentes códigos de bloco para *encoding*. Pela análise da curva de probabilidade de erro de bit de informação $P_b$ pela probabilidade de erro de transmissão $p$, pode-se identificar dois fatores que modificam a curva para esse tipo de sistema: o tamanho da palavra código e a taxa de transmissão de informação do sistema.

Quando se diminui a probabilidade de erro de bit na transmissão $p$, a probabilidade de erro de bit de informação $P_b$ diminui, já que a decodificação do código precisa corrigir menos erros de transmissão. Uma outra maneira de pensar isso é que o código de Hamming e o nosso código Alternativo são bons em decodificar blocos com zero ou um erros. Conforme $p$ diminui, blocos com dois ou mais erros são cada vez mais raros, o que contribui para um $P_b$ menor.

Outra relação feita é que **à medida que o tamanho $K$ do bloco aumenta – ou que a taxa do sistema $K/N$ diminui – aumenta-se o desempenho da probabilidade de erro de bit. Visualmente, a curva $P_b$ por $p$ fica mais acentuada para valores pequenos de $p$, conforme a Figura 3**[5]. Em suma,

[3]Resposta para a pergunta 4 do relatório.

[4]Resposta para a pergunta 2 do roteiro.

[5]Resposta para a pergunta 3

observa-se uma relação de dependência – *trade-off* – entre taxa do sistema e desempenho, e entre tamanho do bloco de código e desempenho.