UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE TELEINFORMÁTICA

GEORGE HARINSON MARTINS CASTRO

UMA ABORDAGEM EXPLORATÓRIA DE CÓDIGOS DE CORREÇÃO DE ERROS BASEADOS EM HAMMING MATRICIAL

FORTALEZA

2021

GEORGE HARINSON MARTINS CASTRO

UMA ABORDAGEM EXPLORATÓRIA DE CÓDIGOS DE CORREÇÃO DE ERROS BASEADOS EM HAMMING MATRICIAL

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Teleinformática do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Mestre em Engenharia de Teleinformática. Área de concentração: Sinais e Sistemas.

Orientador: Prof. Dr. Jarbas Aryel Nunes da Silveira.
Coorientador: Prof. Dr. César Augusto Missio Marcon.

FORTALEZA

2021

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

C351a Castro, George Harinson.
UMA ABORDAGEM EXPLORATÓRIA DE CÓDIGOS DE CORREÇÃO DE ERROS BASEADOS EM HAMMING MATRICIAL / George Harinson Castro. – 2021.
66 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia de Teleinformática, Fortaleza, 2021.
Orientação: Prof. Dr. Jarbas Aryel Nunes da Silveira.
Coorientação: Prof. Dr. César Augusto Missio Marcon.

1. Memórias. 2. Decodificadores. 3. Código de Hamming. 4. Erros. 5. Códigos Corretores de Erros. I. Título.
CDD 621.38

GEORGE HARINSON MARTINS CASTRO

UMA ABORDAGEM EXPLORATÓRIA DE CÓDIGOS DE CORREÇÃO DE ERROS BASEADOS EM HAMMING MATRICIAL

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Teleinformática do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Mestre em Engenharia de Teleinformática. Área de concentração: Sinais e Sistemas.

Aprovada em: 28/06/2021.

BANCA EXAMINADORA

_____

Prof. Dr. Jarbas Aryel Nunes da Silveira (Orientador)

Universidade Federal do Ceará (UFC)

_____

Prof. Dr. César Augusto Missio Marcon (Coorientador)

Pontifícia Universidade Católica do Rio Grande do Sul (PUC-RS)

_____

Profa. Dra. Lirida Naviner de Barros

Institut Polytechnique de Paris

_____

Prof. Dr. Giovanni Cordeiro Barroso

Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

A Deus por ter me guiado a alcançar os objetivos durante minha vida.

À família de um neófita pai, base de tudo que me deu motivação para a conclusão deste trabalho no meio de uma pandemia.

Ao meu orientador, professor Jarbas Silveira, pela confiança, amizade e oportunidade na realização do mestrado. Confiança dada desde 2011. Obrigado por tudo!

Ao meu coorientador, professor César Marcon, pelos conselhos, direcionamentos e dedicação que contribuiu para este trabalho. Ajudas tão valorosas de ideias de como prosseguir no trabalho. O meu muito obrigado!

Aos professores participantes da banca examinadora, pelo tempo dedicado à avaliação deste trabalho e pelas valorosas colaborações e sugestões.

À Universidade Federal do Ceará, pela formação.

“Tudo o que um homem imaginar, outros homens poderão fazer.”

(Júlio Verne)

RESUMO

Nos últimos anos, com o crescente processo de compactação dos componentes eletrônicos, houve o desenvolvimento de semicondutores com bilhões de transistores. Tais avanços tecnológicos propiciaram o avanço de sistemas cada dia mais eficazes, sendo alguns desses destinados a aplicações espaciais, sendo o ambiente bastante hostil para componentes eletrônicos. Além disso, a evolução tornou as memórias mais rápidas e com escalas nanométricas, operando em altas frequências e baixo consumo de energia, trazendo preocupações aos projetistas desses sistemas, pois erros podem acometer esses componentes. Os erros podem ser de um único *bit* ou erros em rajadas, o que torna o conjunto vulnerável, se não for aplicada nenhuma técnica de tolerância a falhas. Assim, os códigos corretores de erros (CCE), se mostram bastante eficientes para a correção de erros em *bits* e com custo de realização menor. Desse modo, o presente trabalho propõe a exploração de organização, algoritmos e capacidade de correção de erros por meio de códigos organizados espacialmente, usando o código de *Hamming*. Objetivou-se saber como se pode obter melhores taxas de correção de erros em *bits*, variando organizações das palavras de código mais uma redundância associada nas matrizes organizacionais. Usando códigos de *Hamming* (8,4) e (13,8) e diferentes organizações de dados, desenvolveu-se a codificação e diferentes formas de decodificação, usando a linguagem de programação Java, para melhorar as taxas de correção de erros em *bits*. Os resultados foram melhores taxa de correções de dados com 100% de correção para erros de 2 (dois) e 3 (três) *bits* de dados em alguns ordenamentos. A última organização de dados chegou a corrigir quase 100% para erros em 4 (quatro) *bits* de dados, o que mostra ser possível avançar ainda mais para que se possa obter melhores taxas de correções de erros em *bits* de dados. Além de melhores taxas de correção de erros, a exploração trouxe novas contribuições sobre a organização espacial dos dados e também sobre como se pode avançar os estudos exploratórios usando códigos matriciais.

Palavras-chave: memórias; decodificadores; código de *hamming*; erros; códigos corretores de erros.

ABSTRACT

In recent years, with the increasing miniaturization process of electronic components, semiconductors with billions of transistors have been developed. Such technological advances have led to the advancement of more and more effective systems, some of which are intended for space applications, in which the environment is totally hostile to electronic components. Evolution has also made memories faster and with nanometric scales operating at high frequencies and low energy consumption, bringing concerns to the designers of these systems, as errors can affect these components. Such errors can be single bit or burst errors, which makes the whole set vulnerable, if no fault tolerance technique is applied. Thus, error correction codes (ECC) prove to be quite efficient for the correction of errors in bits, with a low cost of execution. Hence, the present work proposes the exploration of organization, algorithms and error correction capability through spatially organized codes using the Hamming code. The study aimed to find out how to obtain better error correction rates in bits by varying the organization of code words plus an associated redundancy in organizational matrices. Using Hamming codes (8.4) and (13.8) and different data organizations, coding and different forms of decoding have been developed, using the Java programming language, to improve the error correction rates in bits. The results were better data correction rates with 100% correction for errors of 2 (two) and 3 (three) bits of data in some orderings. In addition, the last data organization corrected almost 100% for errors in 4 (four) bits of data, which shows that we can go even further in order to obtain better error correction rates in data bits. Besides better error correction rates, the exploration brought a lot of knowledge with respect to the spatial organization of the data and also to how we can advance exploratory studies using matrix codes.

Keywords: memories; decoders; hamming code; errors; error correcting codes.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CCE	Códigos Corretores de Erros
BCH	Bose-Chaudhuri-Hocquenghem
CLC	*Column-Line-Code*
CR	*Checkbits*
DR	Dados Redundantes
ECC	*Error Correction Code*
IDE	*Integrated Development Environment*
LET	*Linear Energy Transfer*
LPC	Line Product Code
MBU	*Multiple Bit Upset*
SEC-DED	*Single Error Correction - Double Error Detection*
SEE	*Single Event Effects*
SEU	*Single Event Upset*

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, com a crescente miniaturização dos componentes eletrônicos, houve o desenvolvimento de chips com bilhões de transistores. Tais avanços tecnológicos propiciaram o desenvolvimento de sistemas cada dia mais eficazes. Alguns desses sistemas têm sido destinados a aplicações espaciais, e a confiabilidade se torna não só necessária, mas um importante requisito para garantir a integridade dos dados durante as missões espaciais. É importante destacar que as memórias representam uma parcela significativa dos circuitos, sendo esses componentes mais sensíveis a radiações causadas pelo ambiente hostil (LI *et al.*, 2018). Além disso, hoje, há também uma preocupação dos projetistas, pois com a redução e integração dos semicondutores, estes passaram a operar em altas frequências e baixos níveis de tensão, o que aumenta as chances de falhas transitórias ou permanentes em circuitos modernos (GRANHAUG; AUNET, 2008).

Desse modo, erros podem ser induzidos por radiação, ocasionando um SEU (*single event upset*), quando ocorre uma inversão de um *bit* apenas ou múltiplas inversões de *bits* MBU (*multiple bit upset*) (Baumann, 2005). Tais erros, nos *bits* de dados ou de controle, podem trazer consequências severas aos diversos sistemas computacionais, sendo necessário o uso de técnicas de tolerância a falhas.

Essas técnicas de tolerância a falhas devem ser utilizadas nos diversos sistemas para tratar possíveis condições de erros. O entendimento dos conceitos sobre **defeito, erro e falha** são essenciais para se trabalhar no desenvolvimento desses sistemas. Assim, é fundamental se ter técnicas de tolerância a falhas necessárias e capazes, não só para garantir a correção nos dados, como também detectar dados defeituosos, além de se ter um Código Corretor de Erro (ECC – *Error Correction Code*) capaz de lidar com erros múltiplos em *bits* e que possa se adaptar às condições de erros e interferências do meio. Os códigos corretores de erros (CCEs) podem garantir, de certa forma, uma proteção às memórias, usando os denominados SEC-DED (*Single Error Correction – Double Error Detection*) (GHERMAN *et al.*, 2011).

Desse modo, este trabalho apresenta uma exploração de algoritmos de correção de erros a partir de códigos organizados espacialmente, baseados em *Hamming*. Para isso, serão apresentadas propostas de organizações das palavras de código nas matrizes mais uma redundância associada, que poderá melhorar a correção dos dados na presença de erros múltiplos. Assim, pode-se verificar até onde se pode avançar nos códigos corretores de erros para diversos tipos de organizações.

1.1 Objetivos

O objetivo principal deste trabalho é explorar organizações de ECCs aplicadas a dados organizados espacialmente. Para essa exploração, são utilizadas variações de ECCs baseados em *Hamming*.

Assim, na realização deste trabalho, destacam-se os seguintes objetivos específicos.

a) Implementar organizações de dados e códigos de correção que permitam explorar limiares de taxa de correção de erro versus proporção de redundância aplicada;
b) Explorar combinações de códigos Hamming e dados: (i) Hamming com diferentes tamanhos; i.e., Hamming (8,4) e Hamming (13,8); (ii) Hamming estendido; (iii) Hamming estendido e redundância de dados;
c) Desenvolver diversas formas de decodificações usando o código de *Hamming* para a correção e detecção de *bits* de erros;
d) Avaliar os códigos desenvolvidos com geração de erros exaustiva.

1.2 Estrutura do trabalho

Este trabalho está organizado em 6 capítulos. O capítulo 2 trata da fundamentação teórica acerca dos assuntos abordados neste trabalho. O capítulo 3 trata de trabalhos existentes relacionados a esse tema. O capítulo 4 detalha a metodologia. No capítulo 5, são apresentados e discutidos os resultados. Por fim, o capítulo 6 apresenta as conclusões desta dissertação e as sinalizações para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo trata dos aspectos ligados ao domínio da pesquisa sobre erros e os códigos corretores de erros que usam o código de *Hamming* em sua codificação. Inicialmente, nesta seção, são abordados os conceitos de defeito, erro e falha para uma uniformização dos termos usados neste trabalho.

2.1 Falhas em circuitos integrados

2.1.1 Falha, erro e defeito

Técnicas de tolerância a falhas são aplicadas, atualmente, em diversos sistemas computacionais, sendo muito utilizadas para tratar possíveis erros durante o processo de envio e recebimento de informações. Assim, deve-se considerar e entender três conceitos, sendo eles defeito, erro e falha (AVIZIENIS; LAPRIE; RANDELL, 2001). Um **defeito** é um desvio do comportamento do sistema das especificações pré-estabelecidas. Já o **erro** está para a diferença do resultado atingido para o resultado esperado, podendo causar um defeito (um resultado errôneo). E por último a **falha**, a qual é a causa, normalmente física, que pode resultar em um erro de um componente do sistema[1].

Figura 1 – Relacionamento falha, erro e defeito

Fonte: Avizienis, Laprie e Randell (2001)

Assim, a tolerância às falhas se torna importante para os sistemas, e sua estratégia mais comum se dá pelo uso de redundância nos diferentes níveis (software, hardware e no tempo), sendo a replicação um importante mecanismo para que o sistema se recupere na presença de falhas.

[1] Este trabalho usa os conceitos de Falha, Erro e Defeito tendo como base Avizienis, Laprie e Randell (2001), os quais sinalizam que o erro é parte de um estado errôneo, sendo um desvio do especificado.

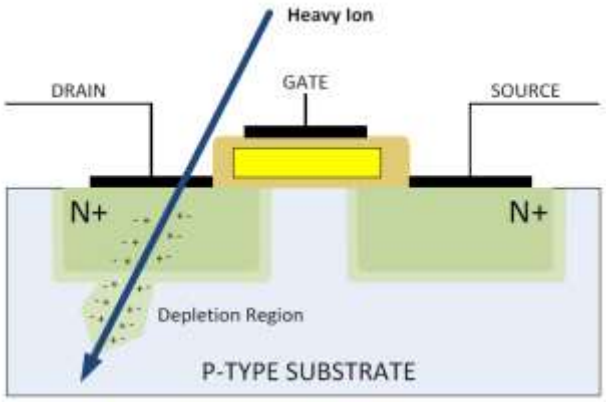
2.1.2 Erros em bits

Os avanços tecnológicos tornaram os diversos circuitos integrados capazes de funcionar em alta frequência e em baixo nível de tensão. Porém, tais evoluções tecnológicas, em especial o baixo nível de tensão, podem deixar os circuitos mais susceptíveis à radiação em ambientes hostis, causando inversão de *bits*. Desse modo, quando cadeias de partículas fortemente carregadas, sejam de forma elétrica ou cinética, atingem os semicondutores, podem resultar em alterações nas cargas dos dispositivos. Assim, impactos de radiação ionizante podem afetar os circuitos, trazendo como consequência falhas, sendo esses eventos de inversão de um único *bit* (*Single Event Effects* – SEE) ou alterações em vários *bits*, causando *Multiple Bit Upset* (MBU) (RADAELLI *et al.*, 2005; LI *et al.*, 2018).

Para Baumann (2005), são preocupantes os efeitos "soft" (suave) de *single-event effects* (SEEs), como também os erros do tipo "hard" (severos) SEEs em ambientes hostis, como o espaço, podendo causar danos maiores, como falhas em componentes do sistema. Desse modo, o autor descreve um erro suave quando um determinado evento (radiação) causa uma perturbação suficiente para inverter ou causar reversão no estado de dados de uma célula de memória.

Os efeitos singulares de erros são ocasionados por um íon que colide com o semicondutor, ocasionando o erro por uma ionização direta (*Linear Energy Transfer* – LET) com carga de energia depositada suficiente para a ocorrência da inversão de *bit* (BROSSER; MILH, 2014). A Figura 2 mostra a colisão de uma partícula carregada (ionização direta) e sua reação no material.

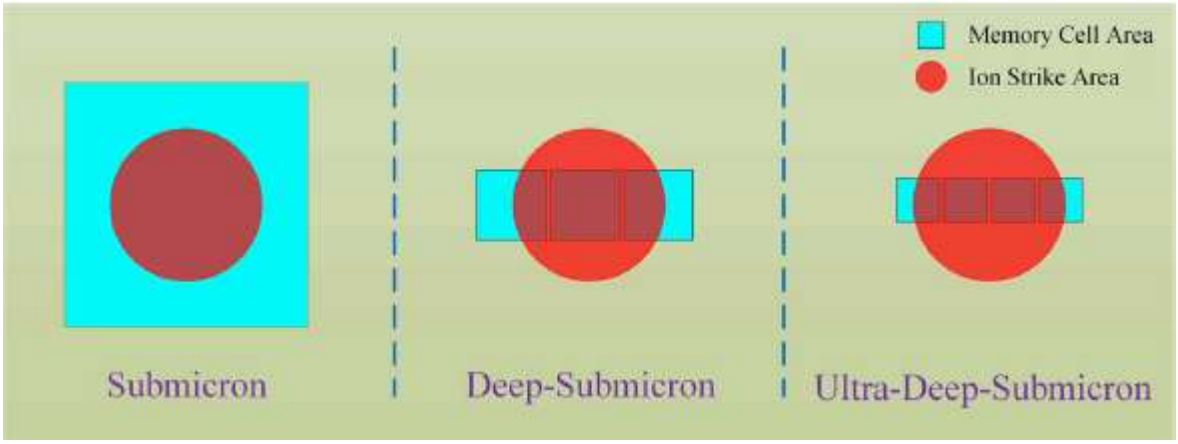
Figura 2 – Ionização direta



Fonte: Brosser e Milh (2014)

Em aplicações espaciais, o impacto em memórias se apresenta bem maior, pois o processo de redução dos circuitos integrados permitiu uma compactação das células de memórias, e, assim, uma única carga pode alterar grande quantidade de células. É possível visualizar a aproximação e o efeito da radiação nas células de memórias pela Figura 3.

Figura 3 – Efeito da radiação com a variação da tecnologia em células de memória



Fonte: Li *et al.* (2018)

É importante salientar que, para proteger determinados circuitos, o processo de tolerância a falhas não é somente uma preocupação em ambientes hostis, como o ambiente espacial, mas uma preocupação de projetistas quando se fala sobre a miniaturização dos semicondutores.

2.2 Fundamentos de códigos corretores de erros

Os códigos corretores de erros são amplamente difundidos e utilizados para realizar a correção de erros em *bits* de dados e possuem um custo de realização menor (CHEN; HSIAO, 1984). No processo, a palavra de dados é codificada apresentando paridades que serão utilizadas no processo de verificação durante a decodificação. Assim, é possível realizar a correção dos dados, podendo corrigir erros únicos de *bits* ou, quando os algoritmos são mais complexos, podem corrigir erros de *bits* em rajadas ou realizar uma redução significativa de erros nos *bits*.

Desse modo, é preciso ter um entendimento bem consolidado sobre os fundamentos que levam aos códigos corretores de erros. Nessa esteira, este trabalho, para dar uma melhor fundamentação, começa com o entendimento do código de bloco linear, definindo a estrutura de uma palavra de código e, para exemplificar, fornece um exemplo genérico do código em bloco (6,3).

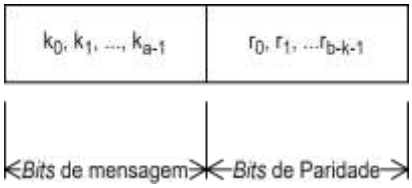
Posteriormente, aborda-se a matriz geradora, em seguida a matriz de verificação de paridade e, por último, a matriz de síndrome. Toda a fundamentação utiliza o código genérico, para dar o entendimento necessário e, em seguida, o estudo continua com o código de *Hamming*, na seção 2.3, sendo este um código de bloco linear.

2.2.1 Estrutura em bloco de ECC

Nos códigos de blocos, a informação é segmentada em blocos, sendo que cada um tem ***k*** *bits* de informações e cada conjunto de dados é independente um do outro. Ao adicionar ***r*** *bits* redundantes em cada bloco de mensagem, obtém-se **n** = ***k*** + ***r*** de comprimento no total e, assim, **n** denomina-se palavra de código (FOROUZAN, 2008; SHU, 1983). Essas palavras de código são o resultado do processo de codificação que transforma cada segmento em um bloco maior, seguindo uma regra previamente definida. Consequentemente, esse bloco maior denomina-se palavra de código, conforme pode ser observado na Figura 4.

Cada segmento irá conter k *bits* de informações binárias e possuirá 2^k blocos de mensagens diferentes, e com n *bits* será possível criar 2^n combinações distintas, o que resulta em número bem maior quando comparado às 2^k possibilidades. Tais combinações são possíveis, porque **n**>***k***, ou seja, o número de *bits* que pode ser alterado é maior (FOROUZAN, 2008), assim pode-se ter mais variações nas palavras de códigos. Cada bloco linear é, então, constituído pelo espaço vetorial de $2^{\mathbf{n}}$ possibilidades.

Figura 4 – Palavra de código



Fonte: elaborada pelo autor.

Na Tabela 1, é possível observar um exemplo de um código de bloco que contém 3 (três) *bits* de informação, o qual, seguindo determinada regra, transforma os *bits* de dados em uma palavra de código que tem 6 (seis) *bits*.

Tabela 1 – Exemplo de código em bloco (6,3)

Mensagem	Palavras de Código
0 0 0	0 0 0 0 0 0
0 0 1	0 0 1 1 0 1
0 1 0	0 1 0 0 1 1
0 1 1	0 1 1 1 1 0
1 0 0	1 0 0 1 1 0
1 0 1	1 0 1 0 1 1
1 1 0	1 1 0 1 0 1
1 1 1	1 1 1 0 0 0

Fonte: Gomes (2011)

2.2.2 Matriz geradora

A matriz geradora **G** é aquela na qual se permite auferir os vetores de códigos $\boldsymbol{e}$, por meio da combinação linear de palavras que representam as informações $\boldsymbol{m}$, sendo obtidos com base em $\boldsymbol{e} = \boldsymbol{m} \,.\, \boldsymbol{G}$. Assim, a matriz $\boldsymbol{G}$ deve ser o fruto direto de um subespaço vetorial com dimensão $k\ x\ n$ linearmente independentes, conforme em (1) (HAYKIN; MOHER, 2011; GOMES, 2011).

$$G = \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{k-1} \end{bmatrix} = \begin{bmatrix} g_{0,0} & g_{0,1} & g_{0,2} & \cdots & g_{0,b-1} \\ g_{1,0} & g_{1,1} & g_{1,2} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ g_{k-1,0} & g_{k-1,1} & g_{k-1,2} & \cdots & g_{k-1,n-1} \end{bmatrix} \qquad (1)$$

Assim, os vetores geradores são $g_0, g_1, \ldots g_{a-1}$ e usando $\boldsymbol{e} = \boldsymbol{m} \,.\, \boldsymbol{G}$. Observa-se, logo em seguida, na Equação (2), a combinação linear dos vetores que estão associados unicamente às mensagens originais.

$$\boldsymbol{e} = m \cdot G = \left(m_0, m_1, \ldots, m_{k-1}\right) \cdot \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{k-1} \end{bmatrix} = m_0 g_0 + m_1 g_1 + \cdots + m_{k-1} g_{k-1} \quad (2)$$

Desse modo, para se obter o **código de bloco (*n*, *k*)**, com ***k*** *bits* de informações e ***n*** *bits* de comprimento total, a partir do exemplo apresentado na Tabela 1, código de bloco (6,3), tem-se:

$$G = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{matrix} 1\ 0\ 0\ 1\ 1\ 0 \\ 0\ 1\ 0\ 0\ 1\ 1 \\ 0\ 0\ 1\ 1\ 0\ 1 \end{matrix} \quad (3)$$

Logo, se a palavra $\boldsymbol{m} = 011$, então:

$$\boldsymbol{e} = (0 \quad 1 \quad 1) \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = 0.g_1 + 1.g_2 + 1.g_3 \quad (4)$$

$$\boldsymbol{e} = 0.(1\ 0\ 0\ 1\ 1\ 0) + 1.(0\ 1\ 0\ 0\ 1\ 1) + 1.(0\ 0\ 1\ 1\ 0\ 1)$$
$$\boldsymbol{e} = (0\ 0\ 0\ 0\ 0\ 0) + (0\ 1\ 0\ 0\ 1\ 1) + (0\ 0\ 1\ 1\ 0\ 1)$$
$$\boldsymbol{e} = \quad 0\ 1\ 1\ 1\ 1\ 0 \quad (5)$$

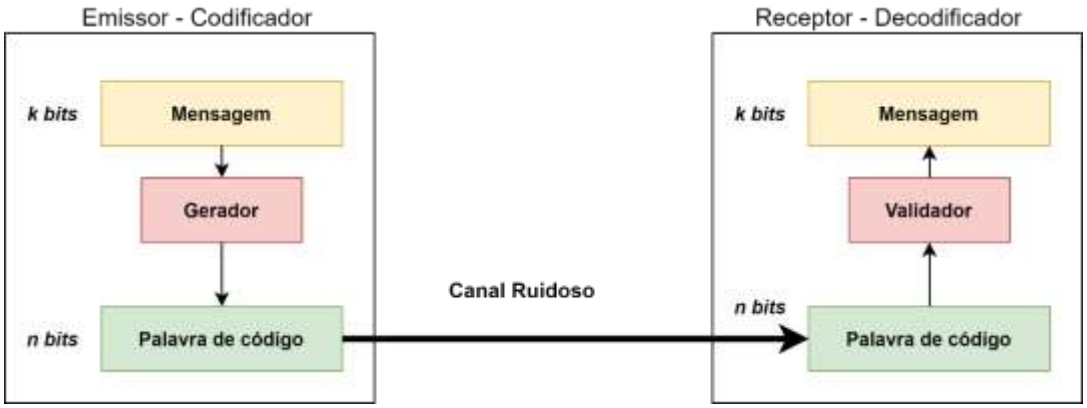
A palavra de código será $\boldsymbol{e} = \quad 0\ 1\ 1\ 1\ 1\ 0$.

É possível notar que um codificador poderá ser implementado por meio de uma lógica puramente combinacional, o que facilita a construção de circuitos codificadores e decodificadores quando for utilizada uma linguagem de descrição de *hardware*. Um outro aspecto importante é que não existe um sistema de memória envolvida para o processo de codificação, dependendo somente da informação atual, isso garante também que um possível circuito em *hardware* não necessite de blocos específicos para construção de memória.

2.2.3 Matriz de verificação de paridade

Uma etapa importante, no processo de envio de informações, é saber se realmente a mensagem transmitida, a partir de um canal de comunicação, chegou ao seu destino sem ter sofrido nenhuma alteração que possa acarretar erros, conforme pode ser visto na Figura 5. Assim, um decodificador se torna necessário para verificar se a mensagem recebida pode ser válida ou não.

Figura 5 – Codificador e decodificador



Fonte: elaborada pelo autor.

Desse modo, uma matriz de verificação de paridade possibilita que cada bloco de paridade seja gerado por uma soma linear dos *bits* a serem enviados, e a matriz de identidade propicia que a parte da mensagem seja duplicada em seguida, conforme forma sistemática em (6).

$$\boldsymbol{G} = \left[\boldsymbol{P}_{k\times(n-k)} | \boldsymbol{I}_{n-k}\right] = \left[\begin{array}{cccc|ccc} P_{00} & p_{01} & \ldots & p_{0,n-k-1} & 1 & 0 & 0 \\ p_{10} & p_{11} & \ldots & p_{1,n-k-1} & 0 & 1 & \cdots 0 \\ \vdots & \vdots & \ldots & \vdots & \vdots & \vdots & \ldots \vdots \\ p_{k-1,0} & p_{k-1,1} & \ldots & p_{k-1,n-k-1} & 0 & 0 & 1 \end{array}\right] = \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{k-1} \end{bmatrix} \quad (6)$$

Se uma matriz $\boldsymbol{H}_{n-k\times n}$ for verificadora de paridade, ela será ortogonal a $\boldsymbol{G}$, matriz geradora; se corresponder a um subespaço vetorial, ou seja, se $\boldsymbol{H}$ é um subespaço dual ao subespaço concebido por $\boldsymbol{G}$, logo os vetores de $\boldsymbol{G}$ serão ortogonais aos de $\boldsymbol{H}$, conforme em (7), (8) e (9) (HAYKIN; MOHER, 2011; GOMES, 2011).

$$\boldsymbol{H} = \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_{n-k-1} \end{bmatrix} = \begin{bmatrix} h_{0,0} & h_{0,1} & h_{0,2} & \ldots & h_{0,b-1} \\ h_{1,0} & h_{1,1} & h_{1,2} & \ldots & h_{1,n-1} \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ h_{n-k-1,0} & h_{n-k-1,1} & h_{n-k-1,2} & & h_{n-k-1,n-1} \end{bmatrix} \quad (7)$$

$$\boldsymbol{G} \cdot \boldsymbol{H}^{\boldsymbol{T}} = 0. \quad (8)$$

$$\boldsymbol{e} \cdot \boldsymbol{H}^{\boldsymbol{T}} = \boldsymbol{0}. \quad (9)$$

Para verificar tal consequência, pode-se dizer que, se determinado código linear foi concebido por uma matriz geradora, uma palavra de código $\boldsymbol{e}$ será válida se e somente se $\boldsymbol{eH^T} = 0$. A matriz $\boldsymbol{H}$ é chamada de Verificação de Paridade, assim, se a matriz que constitui um código estiver na forma de $\boldsymbol{G} = \left[\boldsymbol{I}_{k\times k}|\boldsymbol{P}_{k\times(n-k)}\right]$, conforme a matriz geradora de um código sistemático em (10), a formação da matriz de verificação é relacionada diretamente em (11) (GOMES, 2011).

$$\boldsymbol{G'} = \left[\boldsymbol{I}_{k\times k}|\boldsymbol{P}_{k\times(n-k)}\right] = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & P_{00} & p_{01} & \cdots & p_{0,n-k} \\ 0 & 1 & \cdots 0 & p_{10} & p_{11} & \cdots & p_{1,n-k} \\ \vdots & \vdots & \cdots \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 1 & p_{1,n-k} & p_{2,n-k} & \cdots & p_{k-1,n-k} \end{array}\right] \quad (10)$$

$$\boldsymbol{H} = \left[\boldsymbol{P^T}|\boldsymbol{I}_{(n-k)}\right] = \left[\begin{array}{ccc|ccc} P_{00} & p_{01} & \cdots & p_{k1} & 1 & 0 & 0 \\ p_{10} & p_{11} & \cdots & p_{k2} & 0 & 1 & \cdots 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots \vdots \\ p_{1,n-k} & p_{2,n-k} & \cdots & p_{k,n-k} & 0 & 0 & 1 \end{array}\right] \quad (11)$$

Sendo $\boldsymbol{P^T}$uma matriz que representa a transposta de coeficientes $\boldsymbol{P}$ e $\boldsymbol{I_{n-k}}$ uma matriz identidade. Dessa forma, o universo de $\boldsymbol{G}$ será o espaço nulo de $\boldsymbol{H}$**.** Assim, determinado bloco linear específico pode ser gerado tanto por $\boldsymbol{H}$ ou por $\boldsymbol{G}$.

2.2.4 Matriz de síndrome

Parte-se agora para o exemplo do código de bloco (6,3) a fim de obter sua matriz verificadora de paridade $\boldsymbol{H}$. A seguir, apresenta-se $\boldsymbol{G}$, conforme visto em (3), dessa maneira, para obter $\boldsymbol{H} = \left[\boldsymbol{P^T}|\boldsymbol{I}_{(n-k)}\right]$, usa-se (11):

$$G = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{array}{cccccc} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{array} \quad (12)$$

$$H = \begin{matrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{matrix} \quad (13)$$

De acordo com (8) e (9), pode-se verificar a ortogonalidade utilizando a palavra código e a matriz verificadora de paridade, assim, tem-se:

$$e \cdot H^T = (0\ 1\ 1\ 1\ 1\ 0)\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (14)$$

$$= 0(1\ 1\ 0) + 1(0\ 1\ 1) + 1(1\ 0\ 1) + 1(1\ 0\ 0) + 1(0\ 1\ 0) + 0(0\ 0\ 1) = 0\ 0\ 0$$

Esse resultado corrobora a Equação 9, considerando o receptor da mensagem, conforme Figura 5, que possivelmente poderá receber um vetor adulterado pelo processo de transmissão de dados por um canal não confiável. Cumpre destacar que a principal finalidade do decodificador é recuperar a mensagem a partir do vetor $\boldsymbol{e} = e_1, e_2, \ldots e_n$ enviado. Nessa esteira, caso o vetor $\boldsymbol{r} = r_1, r_2, \ldots r_n$ seja recebido pela transmissão não confiável, tem-se:

$$r = e + err. \quad (15)$$

Sendo que $\boldsymbol{err}$ é um vetor erro que pode ter sido ocasionado durante a transmissão da palavra código ou ter sido introduzido pelo canal de comunicação. Para tanto, considera-se Síndrome $\boldsymbol{S_{n-k}}$, definida pela equação (16).

$$S = r \cdot H^T \quad (16)$$

$$S = (e + err) \cdot H^T = e \cdot H^T + err \cdot H^T \quad (17)$$

Considerando a equação (9), tem-se:

$$\boldsymbol{S = err \cdot H^T} \quad (18)$$

A síndrome é usada no processo de decodificação e detecção de possíveis erros. Assim, $\boldsymbol{S}$ está diretamente ligada a um padrão de erro, sendo que cada parâmetro de erros é ligado a uma síndrome particular (HAYKIN; MOHER, 2011). Uma consequência direta da equação (18) é que a matriz de verificação de paridade $\boldsymbol{H}$ de determinado código propicia o cálculo da síndrome dependendo apenas do padrão de erro ($\boldsymbol{err}$).

O exemplo a seguir detalha a utilização do código (6,3). A palavra código $\boldsymbol{e} = 0\ 1\ 1\ 1\ 1\ 0$ pertence ao conjunto da Tabela 1, cuja mensagem é igual a $\mathbf{0\ 1\ 1}$, assim, sua síndrome será igual a 0 (zero) de acordo com (19).

$$S = (0\ 1\ 1\ 1\ 1\ 0)\begin{bmatrix}1 & 1 & 0\\ 0 & 1 & 1\\ 1 & 0 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} = (0 \quad 0 \quad 0) \qquad (19)$$

Porém, se a palavra código recebida pela transmissão não confiável fosse igual a ($\boldsymbol{e} = 0\ 1\ 1\ 1\ 1\ \mathbf{1}$), sendo que não pertence ao conjunto de palavras disponíveis, então, a síndrome não deverá ser igual a 0 (zero) (GOMES, 2011; HAYKIN; MOHER, 2011).

$$S = (0\ 1\ 1\ 1\ 1\ 0)\begin{bmatrix}1 & 1 & 0\\ 0 & 1 & 1\\ 1 & 0 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} = (0 \quad 0 \quad 1) \qquad (20)$$

$$= 0(1\ 1\ 0) + 1(0\ 1\ 1) + 1(1\ 0\ 1) + 1(1\ 0\ 0) + 1(0\ 1\ 0) + \mathbf{1}(0\ 0\ 1)$$

$$= 0\ 0\ 1 \qquad (21)$$

2.3 Código de *Hamming*

Hamming é um código binário com capacidade de detectar e corrigir erros simples (Hamming, 1950). *Hamming* é muito conhecido e aplicado para correção de erro, sendo o primeiro ECC a ser usado em aplicações que exigiam confiabilidade durante o processo de

envio ou recebimento de informações. A seguir apresentam-se as características da codificação de *Hamming*, a qual foi usada neste trabalho (MOON, 2005; HAMMING, 1950; FOROUZAN, 2008; MOREIRA; FARRELL, 2006; NEUBAUER; FREUDENBERGER; KÜHN, 2007):

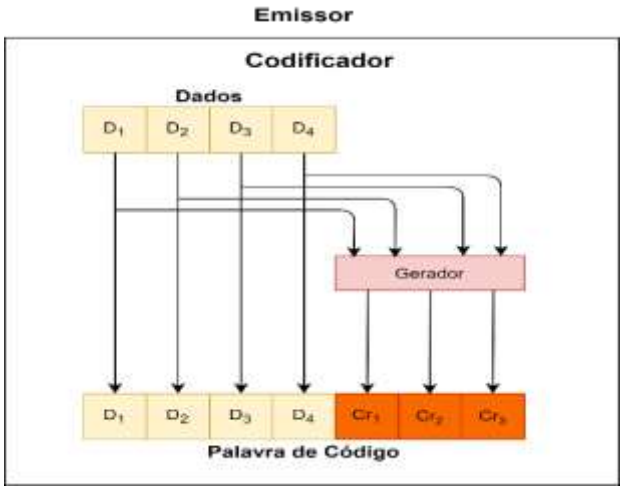
- O código codifica ***M*** *bits* de dados em ***N*** *bits* totais *Hamming* (N, M). E.g., *Hamming* (7,4) codifica 4 *bits* de dados e em 7 *bits*, sendo 3 *bits* de verificação (*checkbits* - C_r);
- $\boldsymbol{N}$ pode ser determinado pela equação $N = \mathrm{M} + \boldsymbol{K}$, sendo $\boldsymbol{k}$ o número de *checkbits* (C_r) gerados na codificação, conforme ilustra a Figura 6;
- O código de *Hamming* deve seguir as equações a seguir:

$$2^k \geq k + M + 1 \tag{22}$$

$$2^M \geq \frac{2^N}{N+1} \tag{23}$$

- A taxa de código $\boldsymbol{r}$ estabelece uma relação entre a quantidade de *bits* de informação (M) a serem codificados e a palavra de código resultante do processo de codificação (N), representando a eficiência do código ($\boldsymbol{r} = M/N$).
- O número total de *bits* (*checkbits)* e o número total de *bits* da informação gera o *overhead* do código.

Figura 6 – Codificador de *Hamming* (7,4) - estrutura



Fonte: elaborada pelo autor.

A Figura 6 evidencia que, no processo de codificação de *Hamming*, foram introduzidos os ***checkbits*** **(Cr)**, ou seja, paridades geradas a partir dos dados de informações representados por D_1, D_2, D_3 e D_4. A paridade irá informar, por meio do seu *bit*, se o número de 1s será par ou ímpar. Assim, se o resultado for par, será igual a 0 (zero) ou, se o resultado for ímpar, será igual a 1 (um) (FOROUZAN, 2008).

2.3.1 Código de Hamming (7,4)

No processo de codificação de *Hamming* (7,4), a matriz geradora $\boldsymbol{G}$ é dada por (24) (NEUBAUER; FREUDENBERGER; KÜHN, 2007):

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{24}$$

Sendo o número de *bits* de dados igual a 4 (quatro) e o número de *bits* de paridade igual a 3 (três) e uma $d_{mín} = 3$, tem-se que uma mensagem $\boldsymbol{m} = \ 0\ 0\ 1\ 1$, codificada por meio da matriz geradora, será igual a (25), conforme ocorre a seguir.

$$\boldsymbol{e} = (0\ 0\ 1\ 1) \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} = (0\ 0\ 1\ 1\ 0\ 0\ 1) \tag{25}$$

Já o processo de decodificação da matriz de verificação de paridade H é dado por (26):

$$H = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{26}$$

Considerando (9), logo

$$e \cdot H^T = (0\ 0\ 1\ 1\ 0\ 0\ 1)\begin{bmatrix}0 & 1 & 1\\ 1 & 0 & 1\\ 1 & 1 & 0\\ 1 & 1 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} = (0\ 0\ 0) \qquad (27)$$

Desse modo, pode-se considerar que a mensagem $\boldsymbol{m} = 0\ 0\ 1\ 1$ não apresenta erro, porque o vetor em (27) foi nulo; caso contrário, $\boldsymbol{m}$ apresentaria erro, e a síndrome representaria um dos vetores de correção apresentados na tabela a seguir (HAYKIN; MOHER, 2011).

Tabela 2 – Decodificação código de Hamming (7,4)

Síndrome	Padrão de erro
000	0000000
100	1000000
010	0100000
001	0010000
110	0001000
011	0000100
111	0000010
101	0000001

Fonte: Haykin e Moher (2011)

2.3.1.1 Compondo a codificação de Hamming (7,4)

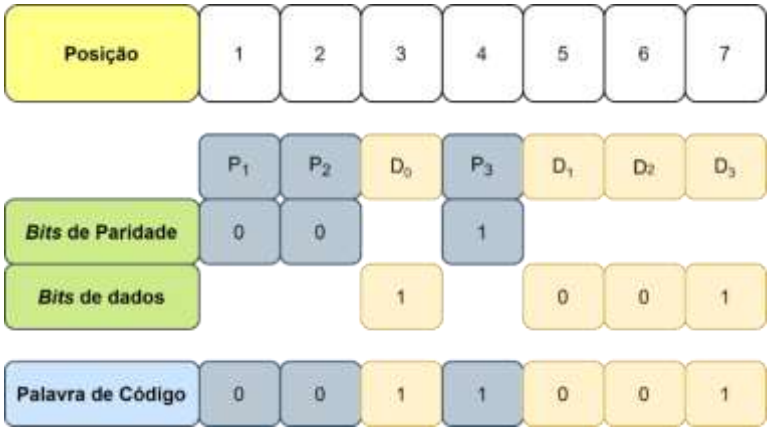
A Figura 7 exemplifica o processo de codificação considerando o *Hamming* (7,4). Assim, dado um conjunto de dados $D_1, D_2, D_3\ e\ D_4$, o processo de codificação se dará com o uso da operação XOR para compor as paridades que farão parte da palavra de código. E, para $\boldsymbol{m} = 1\ 0\ 0\ 1$, tem-se (28), (29) e (30) como a composição dos *bits* de paridade da mensagem, por exemplo.

$$P_0 = D_0 \oplus D_1 \oplus D_3 = 1 \oplus 0\ \oplus 1 = 0 \qquad (28)$$

$$P_1 = D_0 \oplus D_2 \oplus D_3 = 1 \oplus 0\ \oplus 1 = 0 \qquad (29)$$

$$P_2 = D_1 \oplus D_2 \oplus D_3 = 0 \oplus 0 \oplus 1 = 1 \quad (30)$$

Figura 7 – Codificação de *Hamming* (7,4) com paridade par

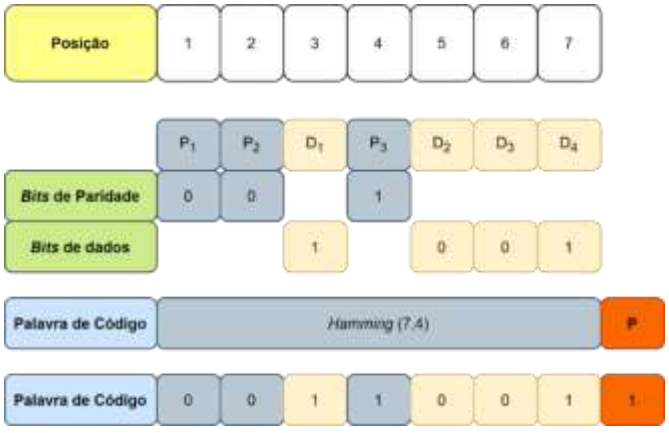


Fonte: elaborada pelo autor.

2.3.1.2 Compondo a codificação de Hamming (8,4) estendido

A codificação do *Hamming* Estendido é dada por *Hamming* (8,4), ou seja, é acrescentado um *bit* a mais se for obtida a paridade da palavra de código. Sendo caracterizado por (*N*+1, *M*), considera-se *N* o comprimento total da palavra de código e *M* o tamanho da mensagem. Essa adição de paridade amplia a distância de *Hamming* para $d_{mín} = 4$. Desse modo, o código terá a capacidade de corrigir um erro e de detectar até dois erros. A Figura 8 exemplifica o processo de codificação de forma estendida.

Figura 8 – Codificação de *Hamming* + Paridade (8,4)



Fonte: Elaborada pelo autor.

Considerando o mesmo exemplo do *Hamming* (7,4), porém acrescentando o *bit* extra de paridade e definindo *Hamming* (8,4), as equações abaixo evidenciam as operações realizadas para as paridades P_0, P_1 e P_2, respectivamente (31), (32) e (33), e para a paridade ***P*** no final do processo de codificação em (34).

$$P_0 = D_0 \oplus D_1 \oplus D_3 = 1 \oplus 0 \oplus 1 = 0 \quad (31)$$

$$P_1 = D_0 \oplus D_2 \oplus D_3 = 1 \oplus 0 \oplus 1 = 0 \quad (32)$$

$$P_2 = D_1 \oplus D_2 \oplus D_3 = 0 \oplus 0 \oplus 1 = 1 \quad (33)$$

$$P = D_0 \oplus D_1 \oplus D_2 \oplus D_3 \oplus P_0 \oplus P_1 \oplus P_2 = 1 \quad (34)$$

O processo de decodificação do *Hamming* (8,4) leva a quatro possibilidades diferentes que são compreendidas na Tabela 3, a qual evidencia a situação da síndrome, da paridade e a compreensão dos estados. Essa tabela foi utilizada para o desenvolvimento dos códigos.

Tabela 3 – Compreensão da Síndrome e do *bit* de paridade do código de *Hamming* (8,4)

Síndrome (S)	**Paridade (P)**	**Compreensão**
S=0 (zero)	P=0 (zero)	Não houve erro na matriz
S≠0 (zero)	P=1 (um)	Ocorreu um erro que pode ser corrigido
S≠0 (zero)	P=0 (zero)	Ocorreu um erro duplo, porém não pode ser corrigido
S=0 (zero)	P=1 (um)	Ocorreu um erro no *bit* de paridade **P**
S=0 (zero)	P=1 (um)	Ocorreu um possível erro triplo que não pode ser corrigido (**anomalia**)

Fonte: elaborada pelo autor.

2.4 Distância mínima de *Hamming*

A distância de *Hamming* é uma medida utilizada para os algoritmos de detecção e correção de erros, sendo a distância mínima a menor entre todos os pares possíveis em um conjunto de palavras. Assim, denomina-se a distância de *Hamming* a diferença entre os *bits* de duas palavras, a qual pode facilmente ser obtida por uma operação XOR (⊕), possuindo sempre um resultado maior que 0 (zero) (FOROUZAN, 2008).

$$d_{mín}(000, 0111\ ) = 2 \tag{35}$$

$$000\ \oplus 011 = 011 \tag{36}$$

$$d_{mín}(00000, 01011\ ) = 3 \tag{37}$$

$$00000\ \oplus 01011 = 01011 \tag{38}$$

Durante o processo de codificação, sempre é necessário se definir o tamanho da palavra de código, o tamanho da mensagem a ser enviada e a distância mínima de *Hamming*. Assim, considerando (35), (36), (37) e (38), percebe-se que a distância entre as palavras é obtida pela operação XOR entre os *bits* das palavras, ou seja, em (35), tem-se a $d_{mín} = 2$ e em (37) tem-se a $d_{mín} = 3$, porque cada uma das palavras difere uma da outra em 2 (dois) e 3 (três) 1s, respectivamente.

Tal medida determina a capacidade de correção de erros na palavra código. Assim, para assegurar a detecção de até $\boldsymbol{x}$ erros, a distância deve ser $d_{mín} = s + 1$. E para todas as demais situações para a correção de até $\boldsymbol{y}$ erros, a diferença mínima deve ser de $d_{mín} = 2y + 1$ em determinado código de bloco (FOROUZAN, 2008). Outros autores relacionam a capacidade de correção de erros para um código de bloco linear com o número de erros que se pode corrigir para cada palavra de código relacionado em (39) (HAYKIN; MOHER, 2011; MOREIRA; FARRELL, 2006; NEUBAUER; FREUDENBERGER; KÜHN, 2007):

$$t = \left\lfloor \frac{d_{mín} - 1}{2} \right\rfloor \tag{39}$$

Sendo $\lfloor x \rfloor$ o maior inteiro ($inteiro \leq x$) que não pode exceder o valor de x, o código que corrige as situações de t erros deve reparar erros de $t + 1$. Assim, para o exemplo da Tabela 1, a $d_{mín} = 3$, ou seja, o potencial de correção é de 1 (um) erro, podendo-se detectar até 2 (dois) erros.

3 TRABALHOS RELACIONADOS

Neste capítulo, é apresentada uma revisão de literatura acerca do assunto de códigos corretores de erros.

Para se corrigir erros de três *bits*, Klockmann, Georgakos e Goessel (2017) propõem, em um artigo, um novo código de correção de erros de três *bits* comparando-se com o código de *Hamming*, o qual consegue corrigir apenas um único *bit*. Os autores trazem, nessa proposta, a correção utilizando apenas dois *bits* de verificação e a correção de até 3 erros adjacentes, o que pode ser facilmente implementado usando portas lógicas do tipo XOR. Conforme relatado, o código pode ser útil para correção de erros em circuitos, como memórias, registradores e matrizes de registradores. Apesar de ser um código simples e bem direto, nada é comentado sobre se ter redundância de dados para possíveis correções, porém tal proposta não é objetivo do trabalho, o qual pode ser ampliado para outras explorações.

Em Das e Touba (2020), é apresentada uma nova classe de códigos corretores de erros em rajada, usando uma decodificação paralela. A metodologia evidenciada se utiliza de códigos já existentes como o *Hamming* e o código BCH (Bose-Chaudhuri-Hocquenghem). A ideia principal da proposta é a utilização de submatrizes identidades na parte superior da matriz de verificação de paridade, sendo apresentada uma nova metodologia de decodificação. Um ponto bastante positivo e relevante da referência é o acréscimo mínimo de redundância se comparado com códigos de *Hamming* para correção de erros de rajada em geral, ocasionando também uma melhor latência no decodificador, quando comparado com outros tipos de soluções. A ideia do trabalho é muito boa, pois reduz a complexidade do decodificador para quase nenhum aumento na redundância, pode-se, inclusive, chamar a atenção pelo fato de os testes serem para tamanho de rajadas até 7 erros; porém os autores não abordam o percentual de correção dos erros. Assim, seria louvável realizar um teste exaustivo com vários tipos de erros. Seu uso é indicado para memórias cujo processo de desempenho é sensível a latências.

Os autores Pfeifer e Vierhaus (2016) exploram novas maneiras de correção direta de erros em *bits* e mostram um passo a passo para correção de erro único, sendo viável apenas se os erros duplos e triplos forem detectados e mapeados, mostrando como expandir para a detecção de múltiplos erros em *bits*. Apesar de falar no Código de *Hamming*, o artigo explora o código Hsiao repetidamente e cita que é possível corrigir erros de 2 *bits*, usando um código matricial. Como resultado principal, é possível detectar erros de 5 *bits*. É importante frisar que a exploração de códigos se faz necessária para este trabalho, a fim de obter dados concretos, aprendizados e futuros códigos e estudos mais aprofundados no trabalho em questão.

Segundo Richter, Oberlaender e Goessel (2008), é possível melhorar a correção de erros triplos usando códigos de correção SEC-DED especiais, sendo necessário um código que reduza a probabilidade de falhas na correção para incidência de erros em rajadas de três *bits*, pois, conforme a referência, uma falha na correção pode ser decorrente de erro na decodificação, ocasionado por uma palavra de código errada que não sinalize tal erro corretamente. Assim, a escolha das colunas da matriz de verificação pode diminuir a probabilidade de erros durante o processo de correção dos *bits*. Os autores relatam que falhas na correção de erros triplos podem ser reduzidas por um algoritmo interativo. Assim, as informações apresentadas trazem uma melhoria na correção de erros em rajadas de três *bits*, porém não garantem a correção de 100% desses *bits*, apenas evidenciam um novo código que reduz erros adjacentes como também erros triplos. É possível perceber que há uma diferença na organização das matrizes, bem como as implementações são para códigos com maiores números de *bits* e redundância associada.

Em Castro *et al*. (2016) e Silva *et al*. (2018), são criados, implementados e avaliados o *Column-Line-Code* (CLC) e o CLC Estendido, os quais figuram como poderosos algoritmos para a detecção e correção de erros em rajadas em memórias, combinando o código de *Hamming* estendido com *bits* de paridade. O CLC apresenta o mesmo número de *bits* na palavra de dados que este trabalho, porém se enquadra em um código matricial com paridades nas linhas e colunas da matriz.

Em um artigo, Freitas *et al*. (2020) propõem um código corretor de erro (ECC) denominado de *Line Product Code* (LPC), o qual utiliza um código matricial a partir do uso do *Hamming* com paridades em linhas e colunas para memórias 3D confiáveis. O código é implementado usando uma palavra de dados de 16 *bits*, como a usada neste trabalho, porém usa um código matricial para a detecção e correção de erros em *bits*. Seu formato de matriz apresenta duas síndromes para cada linha e coluna que amplia a correção de erros duplos de *bits*. Existem casos em que o erro duplo não pode ser corrigido, mas se mostra uma alternativa a mais para explorações e implementações de outros códigos corretores de erros.

4 MATERIAIS E MÉTODOS

Este capítulo descreve os procedimentos utilizados para explorar ECCs lineares com redundância. Para o desenvolvimento e análise da capacidade de correção dos ECCs, emprega-se a linguagem de programação Java com a IDE do Eclipse (FOUNDATION, 2020). O capítulo também descreve a metodologia para verificação da capacidade de correção de erros de cada código explorado.

4.1. A pesquisa exploratória e os códigos utilizados

A pesquisa exploratória visa a obter dados em torno de um determinado problema e, assim, aumentar a experiência do pesquisador para obter resultados consistentes em torno de um determinado assunto para proporcionar uma maior familiaridade com o problema, visando a construir hipóteses. Para tanto, a pesquisa exploratória constitui um trabalho preliminar para outros tipos de pesquisas mais aprofundadas (MARCONI, 2017). Esta seção apresenta como se pode explorar os códigos corretores de erros com redundância usando o Código de *Hamming* para obter melhores taxas de correção em *bits* de dados.

Na Tabela 4, observam-se os códigos para detecção e correção de erros em *bits* de dados com redundância, desenvolvidos utilizando a linguagem de programação Java, para a exploração das diversas taxas de correção de dados com diferentes tipos de organização e de algoritmos. Assim, foram desenvolvidas diversas formas de decodificação para cada tipo de organização dos dados, mas antes será detalhada como a redundância foi implementada para prover uma maior correção e alternativas para corrigir os dados.

É importante destacar que, na literatura, encontra-se a nomenclatura *Hamming* (7,4) ou apenas *Ham* (7,4), por exemplo. Desse modo, para um melhor entendimento do trabalho, adotou-se uma nomenclatura que incluísse também a redundância[2], conforme exemplificado na Tabela 4.

[2] **{Dados} (8,4)** – Refere-se ao *Ham* (8,4) estendido, **{Dados} (8,4) + Redundância** está relacionado ao *Ham* (8,4) estendido mais uma redundância de dados que não faz parte da codificação, **{Dados + Redundância} (13,8)** está associado ao *Ham* (13,8) e, nesse caso, a redundância faz parte da codificação dos dados e **{Dados} (8,4) + {Redundância} (8,4)** é referente ao *Ham* (8,4) dos dados mais o *Ham* (8,4) da redundância, ou seja, a redundância está protegida pelo código de *Hamming* (8,4).

Tabela 4 – ECCs implementados e redundâncias de dados

ECCs	**Número de informações (*Bits*)**			
	Dados	***CheckBits***	**Dados Redundantes**	**Paridade**
{Dados} (8,4)	4	3	-	1
{Dados} (8,4) + Redundância	4	3	4	1
{Dados + Redundância} (13,8)	4	4	4	1
{Dados} (8,4) + {Redundância} (8,4)	4	6	4	2

Fonte: elaborada pelo autor.

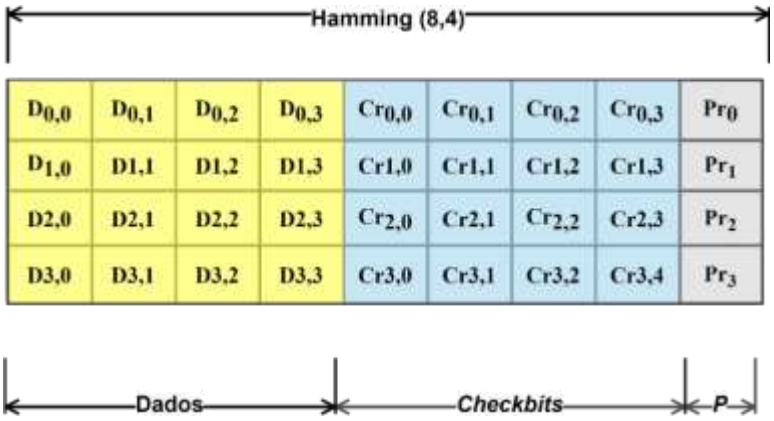
4.2 Organizações

4.2.1 Primeira organização

A primeira organização é um código linear que utiliza o *Hamming* (8,4) estendido, sendo organizada como uma matriz de 16 *bits* de dados que servirá como referência para a comparação da redundância mínima. A Figura 9 mostra a matriz de dados com *checkbits* e paridade.

A matriz para o *Hamming* (8,4) estendido foi definida como referência por apresentar, em sua codificação, o *bit* extra de paridade que será usado também nos outros códigos e tipos de decodificações distintas. Para o processo de codificação, foi usado como referência o portal HAMMING CODE (KOREN, 2021), o qual desenvolve a codificação dos dados, gerando todo o esquema de paridade como também a matriz de síndrome, o que facilitou a padronização da codificação de todas as organizações propostas.

Figura 9 – Dados (8,4) (Referência)

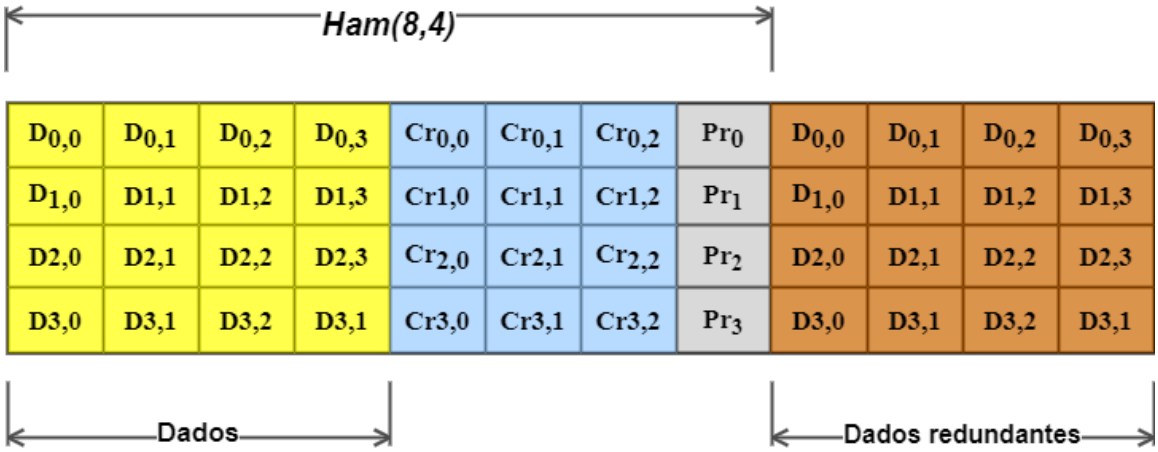


Fonte: elaborada pelo autor.

4.2.2 Segunda organização

A segunda organização é um código linear que utiliza o *Hamming* (8,4) mais dados redundantes, usados para explorar correções alternativas, por exemplo, quando o código detecta erros duplos, porém não consegue informar onde é e nem o corrigir. Na Figura 10, pode-se observar a matriz que contém 16 *bits* de dados + *checkbits* + paridade + 16 *bits* de dados redundantes para explorar a primeira decodificação proposta no trabalho.

Figura 10 – {Dados} (8,4) + Redundância



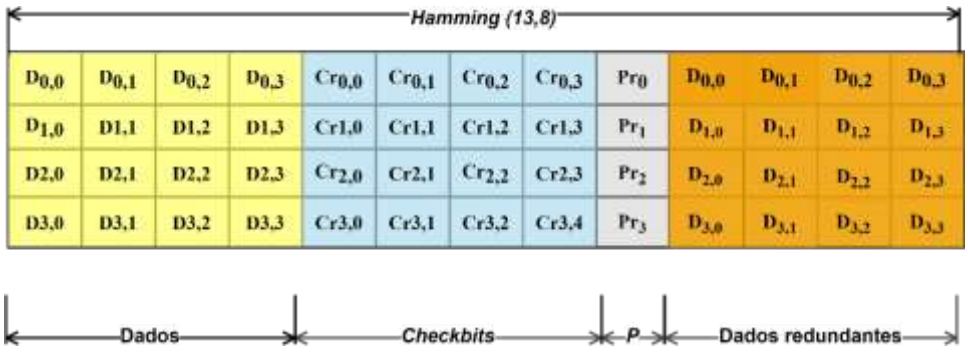
Fonte: elaborada pelo autor.

Essa organização deve propiciar um ganho significativo na correção de erros duplos, quando se utilizam os dados redundantes para a correção dos dados que, possivelmente, podem apresentar erros. Um aspecto relevante a ser considerado é que os dados redundantes não apresentam nenhuma codificação a mais, são apenas a cópia dos dados de forma redundante.

4.2.3 Terceira organização

A terceira organização da matriz de dados mais redundância aplica codificação para a área de dados redundantes junto aos dados. Nesse caso, existem alternativas a serem exploradas, pois a correção ocorre em cada linha de dados e redundância. Assim, estão codificados, inicialmente, em cada linha, conforme a Figura 11.

Figura 11 – {Dados + Redundância} (13,8)



Fonte: elaborada pelo autor.

Na Figura 12 é possível observar a forma como os dados mais os dados redundantes foram codificados, usando o código de *Hamming* (13,8). A redundância entrou na codificação sendo considerada como dados (**D_4, D_5, D_6 e D_7**). Assim, no processo de codificação, foi utilizado **D_0, D_1, D_2, D_3, D_4, D_5, D_6 e D_7**, sendo, os primeiros quatro *bits* de dados, e os quatro *bits* seguintes de dados redundantes.

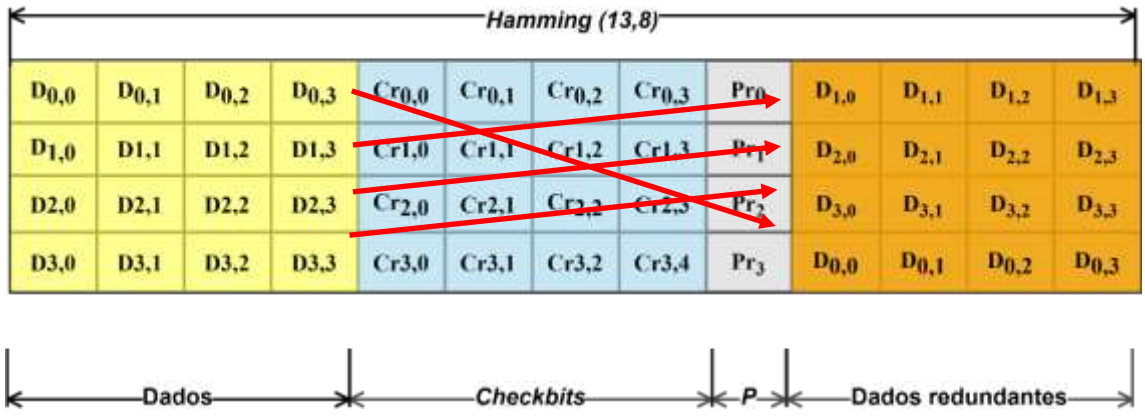
Figura 12 – Codificação do *Hamming* (13,8)

D_0	D_1	D_2	D_3	$Cr_{0,0}$	$Cr_{0,1}$	$Cr_{0,2}$	$Cr_{0,3}$	Pr_0	D_4	D_5	D_6	D_7

Fonte: elaborada pelo autor.

A Figura 13 mostra a codificação do *Hamming* (13,8), porém a organização dos dados redundantes muda um pouco, pois houve o deslocamento entre as linhas de acordo com a indicação das setas.

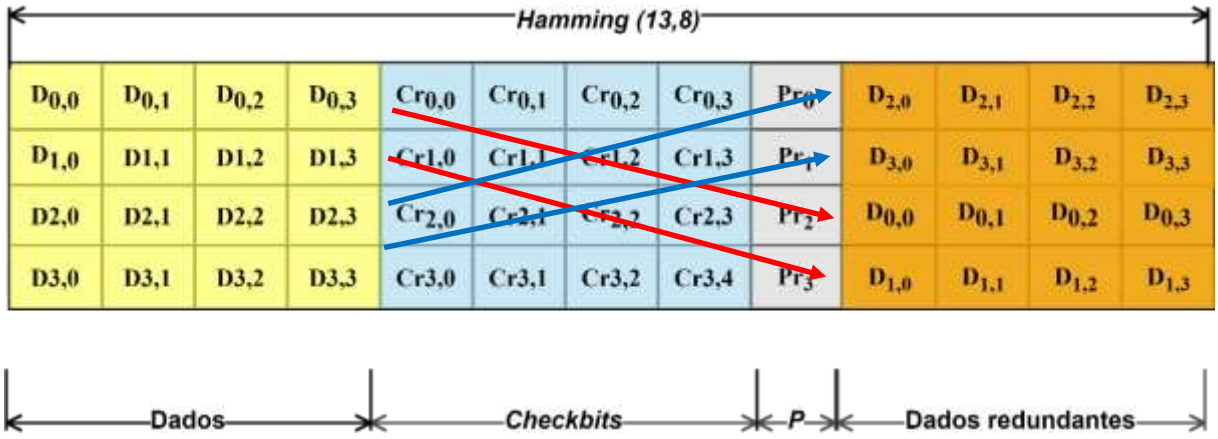
Figura 13 – {Dados + Redundância} (13,8) com deslocamento entre linhas (a)



Fonte: elaborada pelo autor.

Cabe salientar que a codificação *Hamming* (13,8), neste caso, pode proteger mais *bits*, visto que usa quatro *bits* de verificação dos dados. Na organização evidenciada pela Figura 14, também há o deslocamento entre as linhas dos dados redundantes de acordo com a indicação das setas, o que poderá trazer resultados melhores, visto que os dados redundantes, de certa forma, estão a uma distância maior.

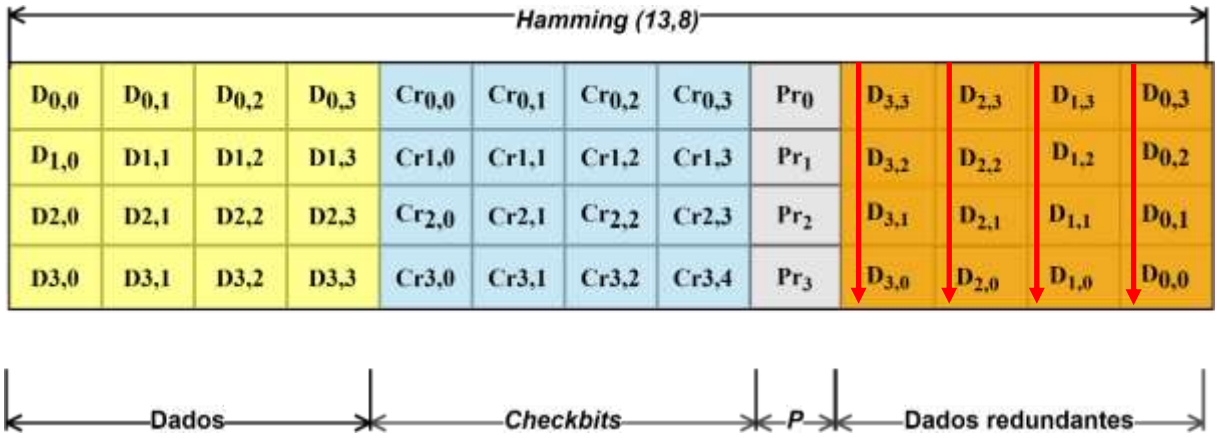
Figura 14 – {Dados + Redundância} (13,8) com deslocamento entre linhas (b)



Fonte: elaborada pelo autor.

Na Figura 15 apresenta-se a organização em forma da matriz **transposta** para os dados redundantes, considerando que a organização está começando de baixo para cima. Assim, cada linha virou coluna nessa disposição dos dados redundantes.

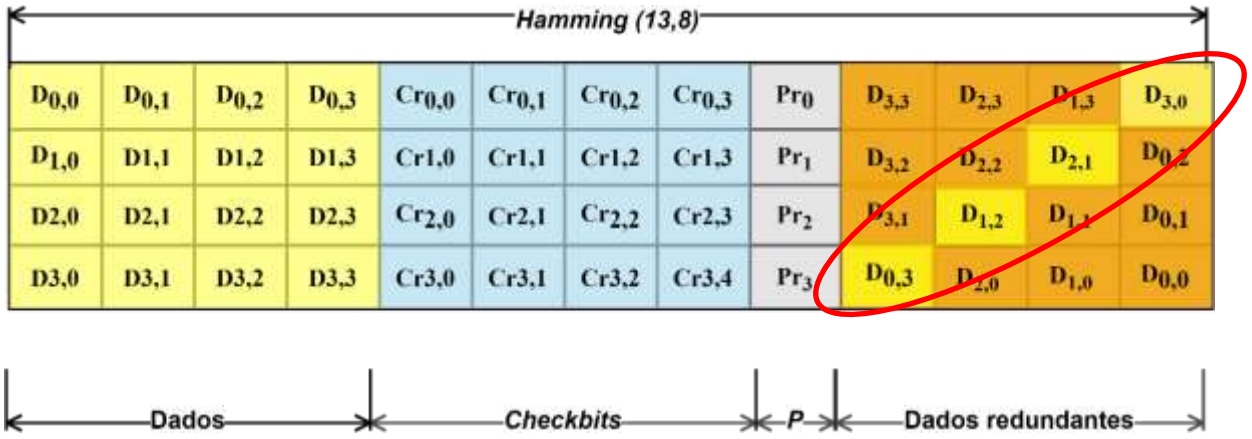
Figura 15 – {Dados + Redundância} (13,8) com matriz transposta



Fonte: elaborada pelo autor.

Também foi realizada uma modificação da Figura 15, conforme pode ser observado na Figura 16. Assim, apresenta-se a organização em forma da matriz **transposta** com a rotação da diagonal secundária, considerando que a organização está começando de baixo para cima. Assim, cada linha virou coluna nessa disposição dos dados redundantes.

Figura 16 – {Dados + Redundância} (13,8) com matriz transposta e rotação

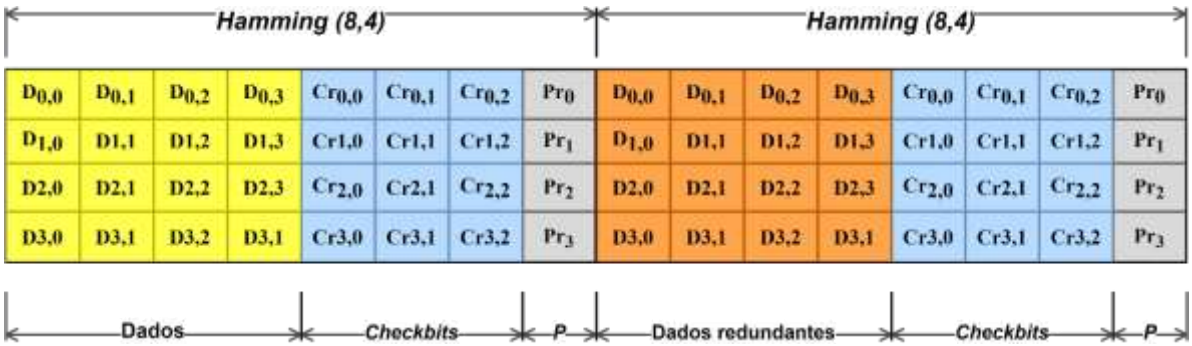


Fonte: elaborada pelo autor.

4.2.4 Quarta organização

A quarta e última organização a ser explorada propõe uma redundância em nível de dados e codificação e deve ser uma forma de organização que aumenta a capacidade de detecção e correção de erros. Na Figura 17, apresenta-se a disposição dos dados mais os dados redundantes codificados, bem como cada um com o *Hamming* (8,4) estendido. Observa-se que há um provável acréscimo de área e codificação, deixando o algoritmo mais complexo, o que poderá acarretar aumento no uso de recursos computacionais do sistema.

Figura 17 – {Dados} (8,4) + {Redundância} (8,4)



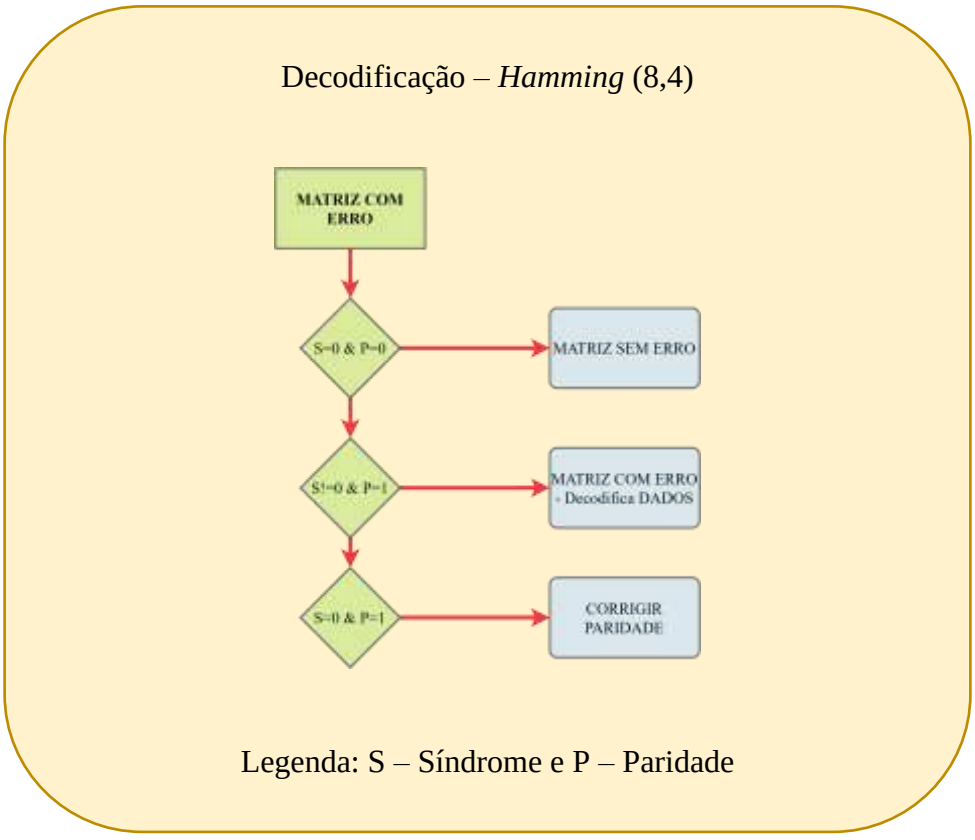
Fonte: elaborada pelo autor.

4.3 Decodificações dos códigos propostos

Neste trabalho, foram utilizados diversos tipos de decodificações para chegar aos resultados esperados para a exploração dos códigos de *Hamming* lineares com redundância. Assim, é preciso entender como cada código se comporta na presença de vários tipos de erros em *bits* de dados ou redundância para chegar nos resultados pretendidos.

Nessa esteira, apresenta-se como cada algoritmo se comportou na parte da decodificação dos dados com sua redundância. De início, parte-se da primeira organização, conforme tópico anterior; pode-se visualizar (Figura 18) o processo decisório durante a etapa de decodificação dos dados, logo após a inserção de vários tipos de erros na matriz. O fluxograma de interação parte da matriz sem erro, indo para a matriz que apresenta erro, nesse caso, realiza-se a verificação dos dados pela matriz de síndrome e, caso haja correspondência, alteram-se os *bits*. Para facilitar o processo de verificação dos dados, usou-se uma matriz de 16 *bits* de dados iguais a 0 (zero). No último processo decisório, tem-se a correção da paridade. Essa decodificação é realizada para o *Hamming* (8,4) estendido.

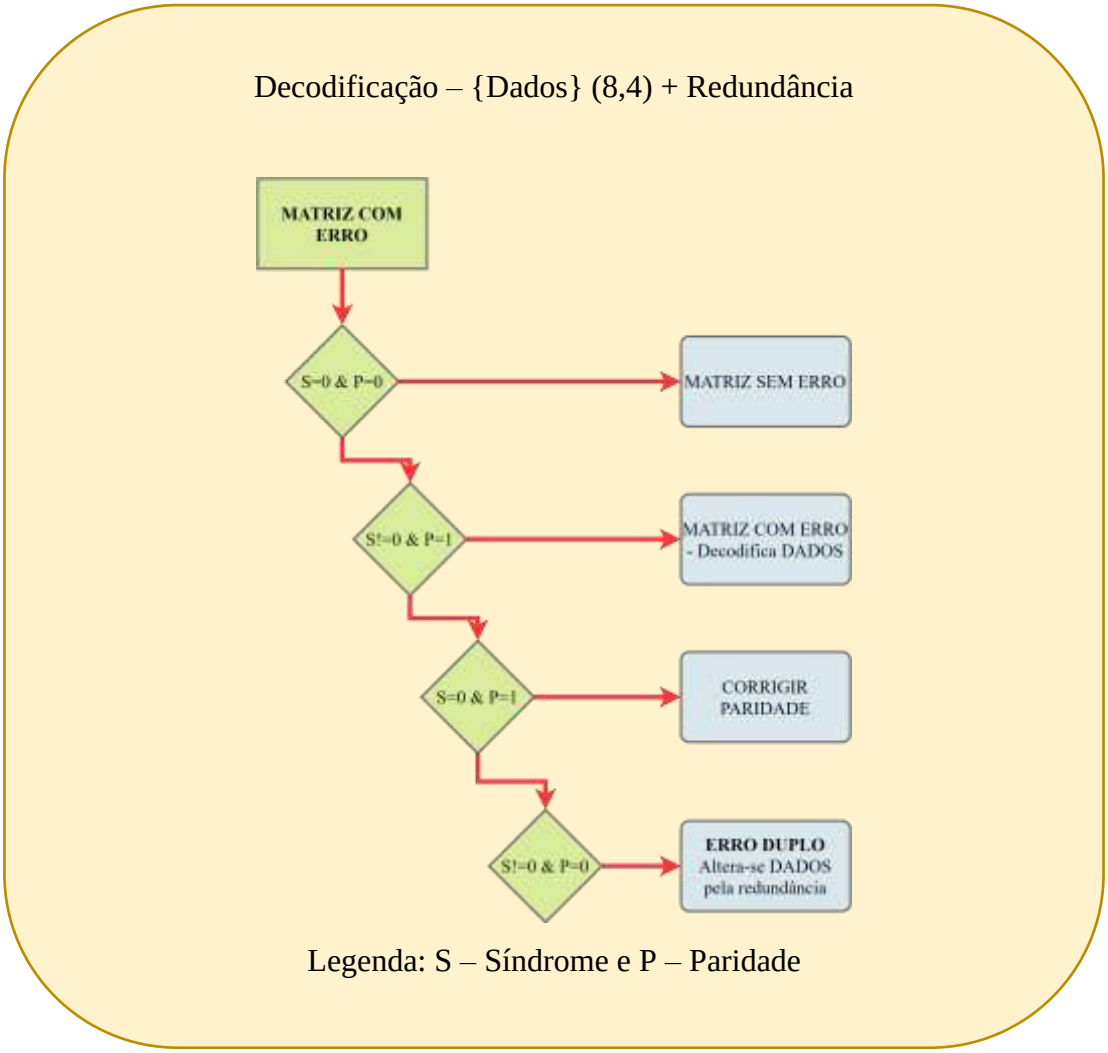
Figura 18 – Decodificação *Hamming* (8,4) estendido



Fonte: elaborada pelo autor.

O próximo código de decodificação é o da segunda organização. Nesse caso, tem-se, inicialmente, a verificação da existência de erros. Posteriormente, considerando que a síndrome é diferente de 0 (zero) e a paridade é igual a 1 (um), aplica-se a decodificação por meio da matriz de síndrome. No próximo processo decisório, verifica-se o erro na paridade e, caso haja erro, altera-se o *bit* de paridade. O último processo é a verificação da condição de erro duplo, aquele que altera dois *bits* nos dados ou *checkbits* + paridade. Conforme a Figura 19, a condição será quando a síndrome for diferente de 0 (zero) e a paridade for igual a 0 (zero) como mostrado na Tabela 5. Nesse caso, altera-se a linha de dados pelos dados redundantes durante a interação do algoritmo.

Figura 19 – Decodificação {Dados} (8,4) + Redundância

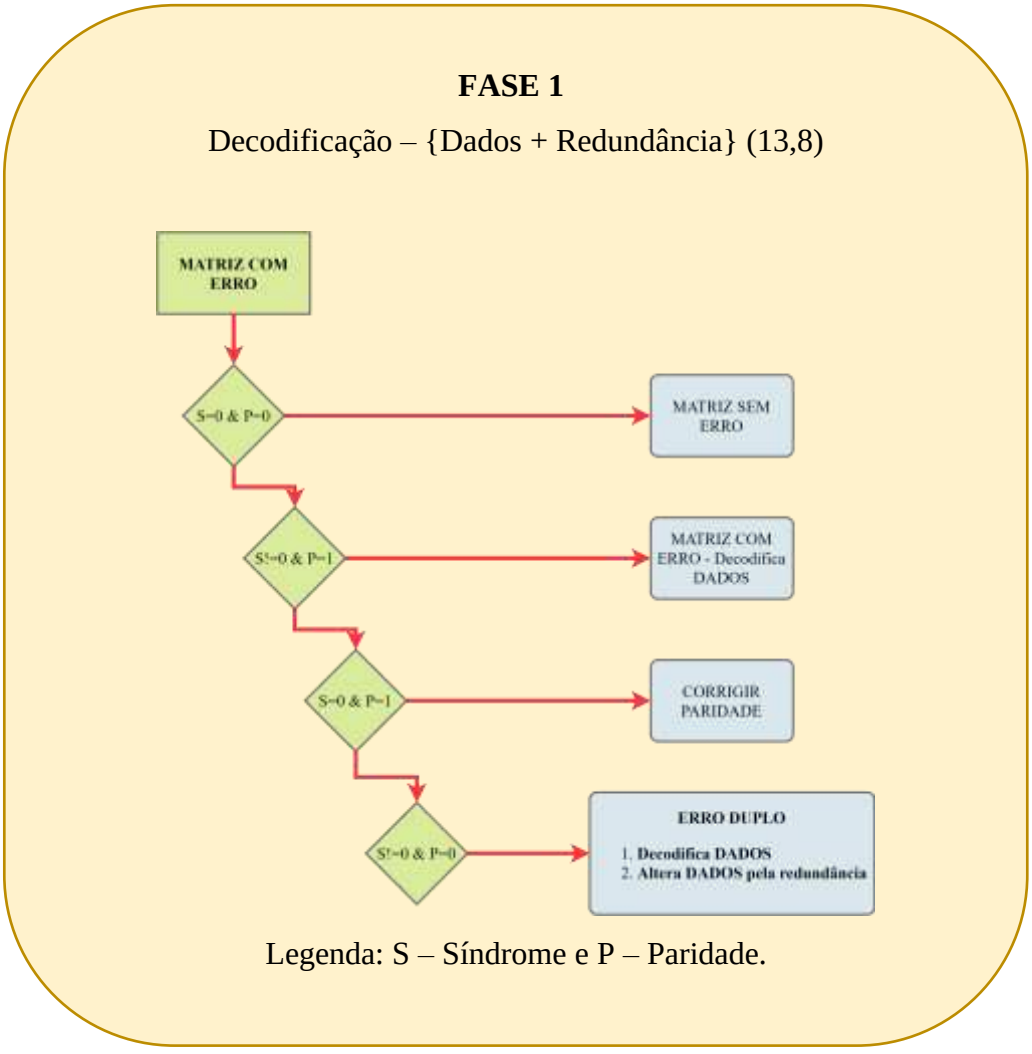


Fonte: elaborada pelo autor.

No processo de decodificação da terceira organização, houve a divisão da decodificação em duas fases distintas. Nesse caso, inicialmente, há a verificação da existência

de erros, conforme observado na Figura 20; posteriormente, considerando que a síndrome é diferente de 0 (zero) e a paridade é igual a 1 (um), há a possível correção pela matriz de síndromes e, em seguida, tem-se a correção da paridade. No próximo passo, tem-se a verificação da existência de erros duplos e, caso haja erro duplo, primeiro se decodificam os dados pela matriz de síndrome e, posteriormente, alteram-se os dados pela redundância (**FASE 1)**.

Figura 20 – Decodificação {Dados + Redundância} (13,8) – FASE 1

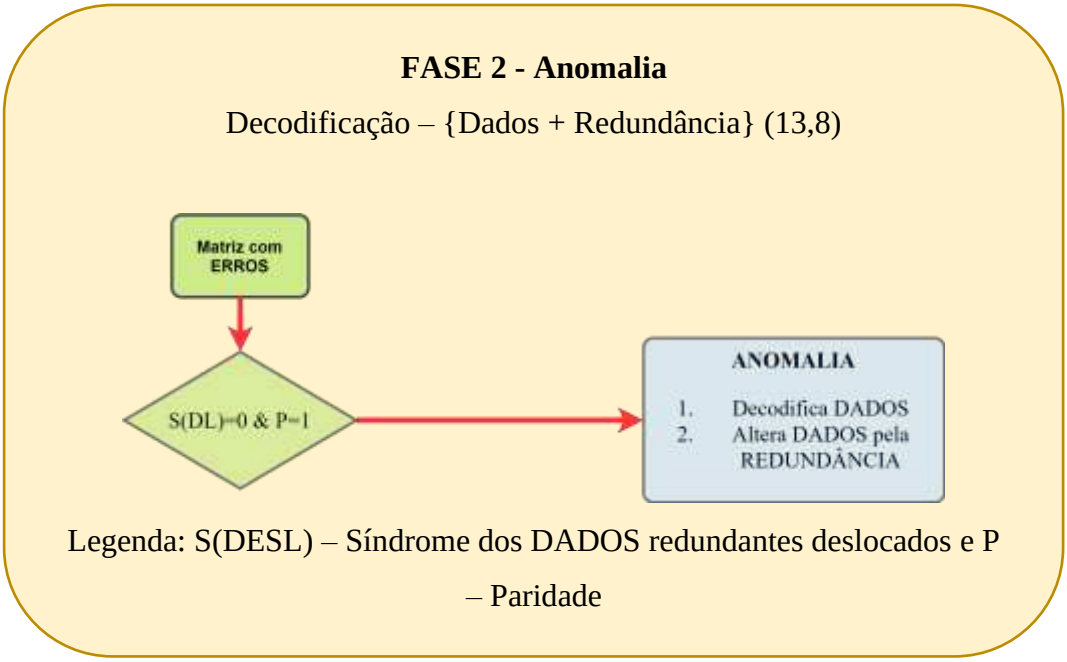


Fonte: elaborada pelo autor.

Na **FASE 2**, conforme a Figura 21, tem-se a interação do processo de decodificação dos dados mais a redundância, considerando o caso de **anomalia** (Tabela 3), que é o caso do possível erro triplo que deixa os *checkbits* dos dados iguais a 0 (zero) e a paridade igual a 1 (um). Nesse ciclo, verificam-se os *checkbits* dos dados deslocados, os quais correspondem aos dados da linha, analisando-se se são iguais a 0 (zero), o que irá garantir que os dados redundantes estão livres de erros; e se a paridade da linha é igual a 1 (um), condição para a

ocorrência de anomalia. Nesse caso, o processo leva primeiro à decodificação, novamente, dos dados e a posterior alteração dos dados pela redundância, o que garante os dados novamente corretos.

Figura 21 – Decodificação – {Dados + Redundância} (13,8) – FASE 2

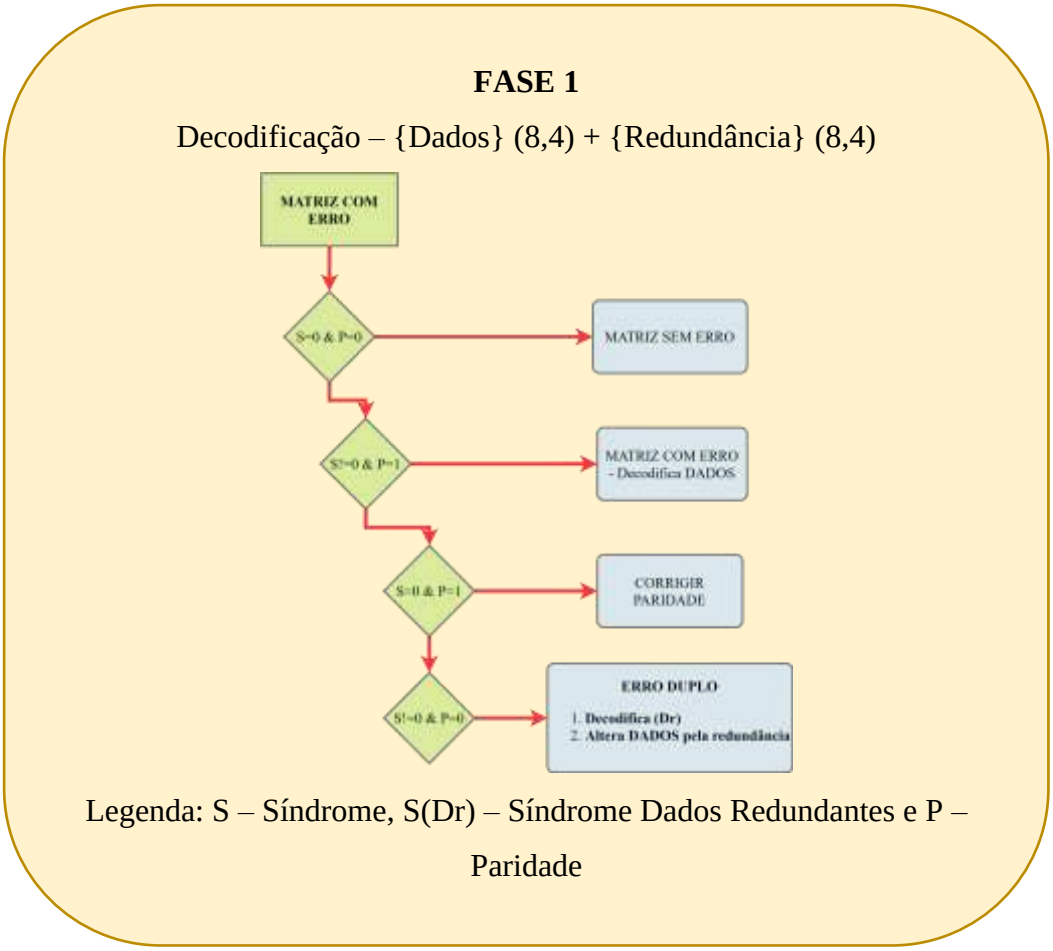


Fonte: elaborada pelo autor.

O último código de decodificação é a quarta organização que também foi dividida em duas fases. Nesse caso, inicialmente, há a verificação da existência de erros; posteriormente, considerando que a síndrome é diferente de 0 (zero) e a paridade é igual a 1 (um), aplica-se a decodificação por meio da matriz de síndrome na matriz, conforme Figura 17 e o fluxograma na Figura 22, e a correção da paridade posteriormente. Em seguida, tem-se a verificação da existência do erro duplo. Nesse caso, o algoritmo realiza, primeiramente, a decodificação pela síndrome dos dados redundantes (DR), parte direta da matriz; e, posteriormente, realiza-se a alteração dos dados pelos dados redundantes.

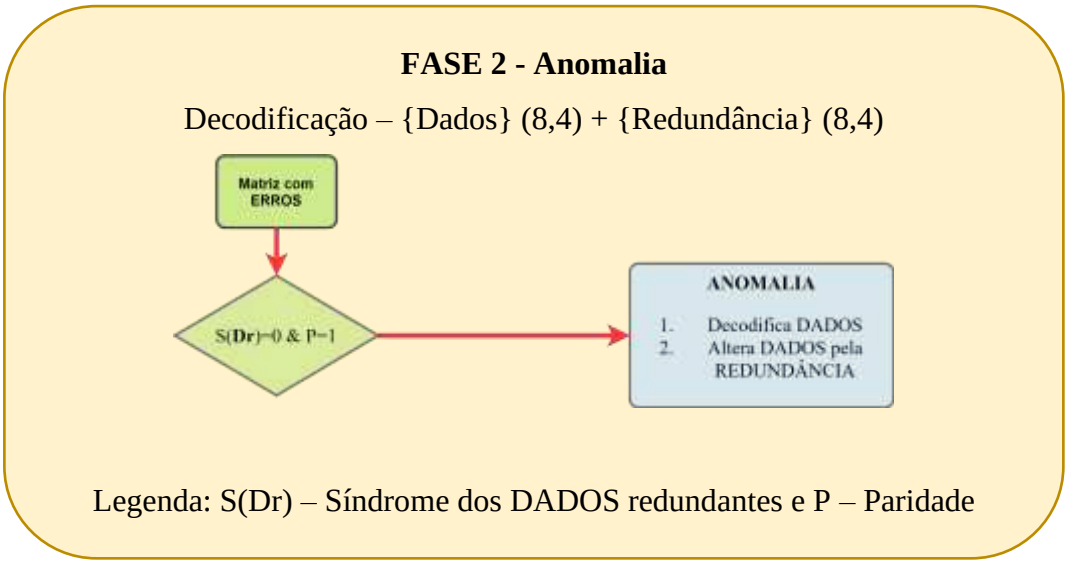
Na sequência **(FASE 2**) de verificação, é testada a condição de anomalia, porém considera-se a seguinte situação: síndrome dos dados redundantes (*Hamming* (8,4) redundante) é igual a 0 (zero) – parte direita; e a paridade da linha é igual a 1 (um) – parte esquerda (Figura 17), caso seja verdade. Primeiramente, decodificam-se os dados redundantes e depois alteram-se os dados pelos dados redundantes (Figura 23). Dessa forma, garante-se que os dados redundantes estão livres de erros.

Figura 22 – Decodificação - {Dados} (8,4) + {Redundância} (8,4) – FASE 1



Fonte: elaborada pelo autor.

Figura 23 – {Dados} (8,4) + {Redundância} (8,4) – FASE 2

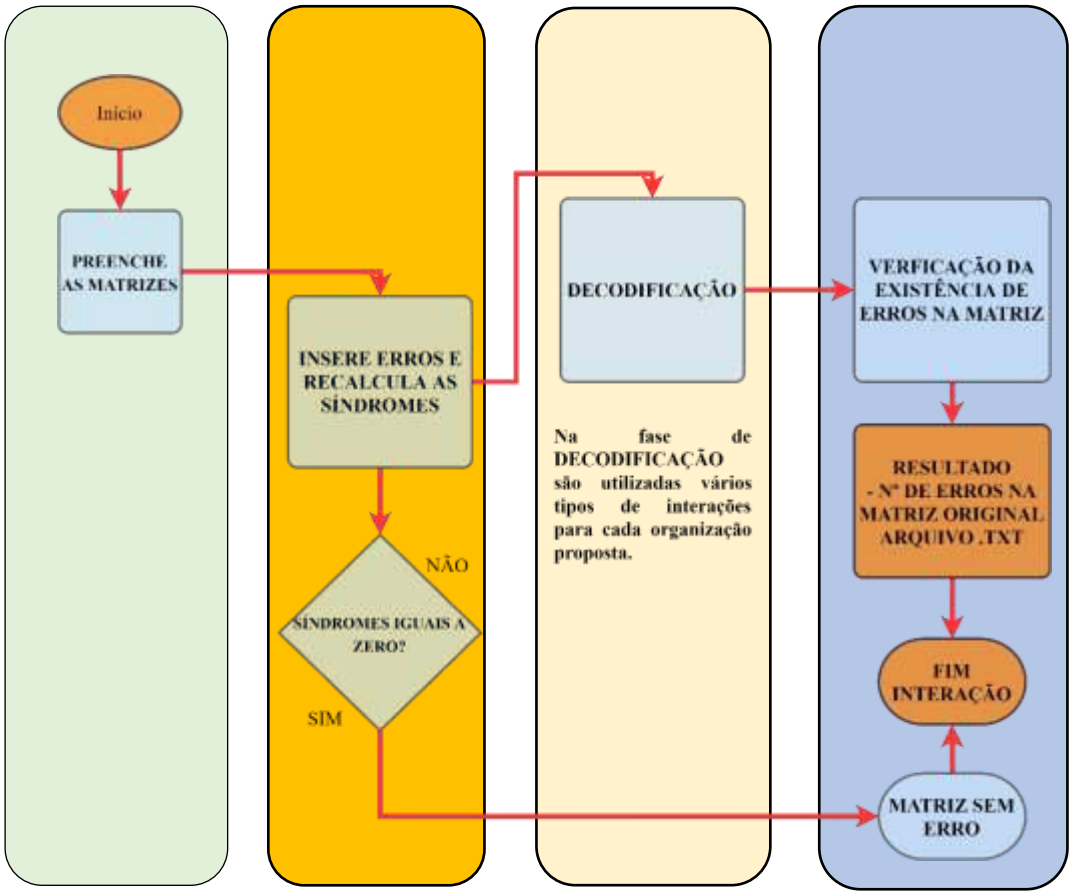


Fonte: elaborada pelo autor.

4.4 Simulação e avaliação da exploração dos códigos corretores de erros

A simulação dos dados foi realizada de acordo com o fluxograma da Figura 24, assim, inicialmente, todos os dados são preenchidos na matriz de simulação, considerando cada tipo de organização, bem como atentando para as variações apresentadas na terceira organização de dados para se verificar a mudança na taxa de correção de dados na presença de vários tipos de erros.

Figura 24 – Fluxograma geral de testes da matriz de dados



Fonte: elaborada pelo autor.

Os testes realizados foram exaustivos, assim, para cada disposição da matriz de dados mais redundância, foram considerados todos os possíveis erros nos dados, nos *checkbits*, na paridade e na redundância, pois, por meio de um método recursivo, pôde-se inserir os erros em cada posição na matriz, após o processo de codificação de dados. As simulações consideraram erros variando entre 1 (um) e 10 (dez) erros.

Logo depois, tem-se a etapa de decodificação (Figura 24) que varia de acordo com a organização dos dados. Para exemplificar como o teste exaustivo foi executado, são mostradas abaixo três figuras contendo um erro e dois erros na matriz de dados. Na Figura 25, visualiza-se que o erro está na posição $D_{0,0}$ na matriz, assim, o próximo erro será inserido na posição $D_{0,1}$ em seguida em $D_{0,2}$ e assim por diante. Já para a inserção de dois erros na matriz, fixa-se primeiro um erro e depois insere-se o outro erro na próxima posição, variando até o final da matriz quando o próximo erro fixo será na posição $\mathbf{D_{0,1}}$ e assim por diante, conforme Figuras 26 e 27. Os outros tipos de erros seguem o mesmo padrão para o teste exaustivo.

Figura 25 – Inserção de um erro na matriz de dados

D0,0	D0,1	D0,2	D0,3
D1,0	D1,1	D1,2	D1,3
D2,0	D2,1	D2,2	D2,3
D3,0	D3,1	D3,2	D3,3

Fonte: elaborada pelo autor.

Figura 26 – Inserção de dois erros na matriz de dados (a)

D0,0	D0,1	D0,2	D0,3
D1,0	D1,1	D1,2	D1,3
D2,0	D2,1	D2,2	D2,3
D3,0	D3,1	D3,2	D3,3

Fonte: elaborada pelo autor.

Figura 27 – Inserção de dois erros na matriz de dados (b)

D0,0	D0,1	D0,2	D0,3
D1,0	D1,1	D1,2	D1,3
D2,0	D2,1	D2,2	D2,3
D3,0	D3,1	D3,2	D3,3

Fonte: elaborada pelo autor.

5 RESULTADOS E DISCUSSÃO

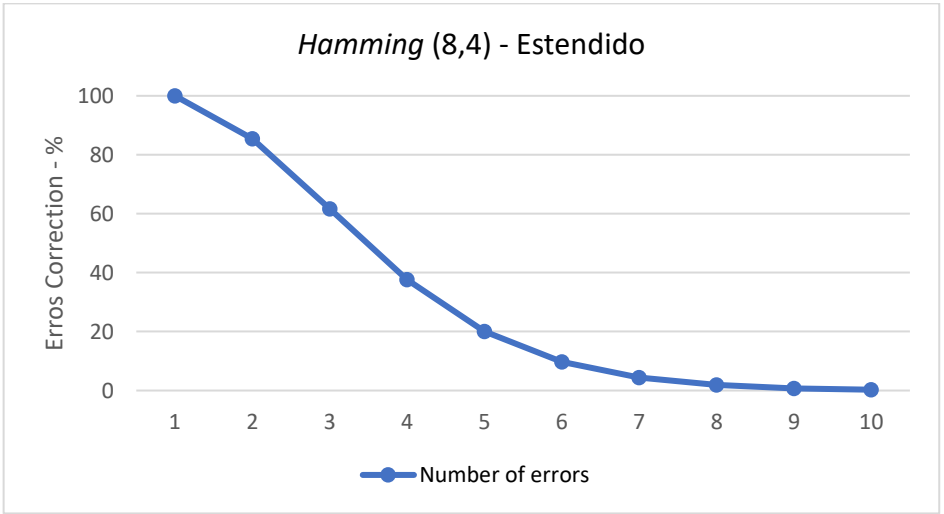
Este capítulo descreve os resultados extraídos das simulações realizadas nas diferentes organizações para a exploração de organização, códigos e capacidade de correção de erros em baseados em *Hamming*.

5.1 Código {Dados} (8,4)

O resultado da primeira organização proposta neste trabalho (Figura 9) é evidenciado no Gráfico 1, sendo a simulação realizada apenas utilizando a configuração *Hamming* (8,4) estendido. A inserção do número de erros variou de 1 até 10 erros para uma simulação exaustiva, ou seja, teste que combina todas as possíveis combinações de erros tanto na parte dos dados quanto na parte das paridades, seguindo $C_{n,p} = \frac{n!}{p!(n-p)!}$. Assim, o número de combinações para a organização número 1 (um) para dois erros será igual 496 possíveis combinações considerando o número de *bits* totais da matriz.

Esse gráfico representa a referência inicial para este trabalho para que se possa avançar na exploração, assim, é possível perceber que a organização está considerando o número de erros pela taxa de correção que seguirá para os outros resultados deste trabalho. Desse modo, é possível perceber que, com o aumento do número de erros, a taxa de correção diminui rapidamente chegando a 0 (zero) quando se tem 10 (dez) erros, o que já era esperado para o *Hamming* (8,4) estendido.

Gráfico 1 – {Dados} (8,4) + Redundância – Referência

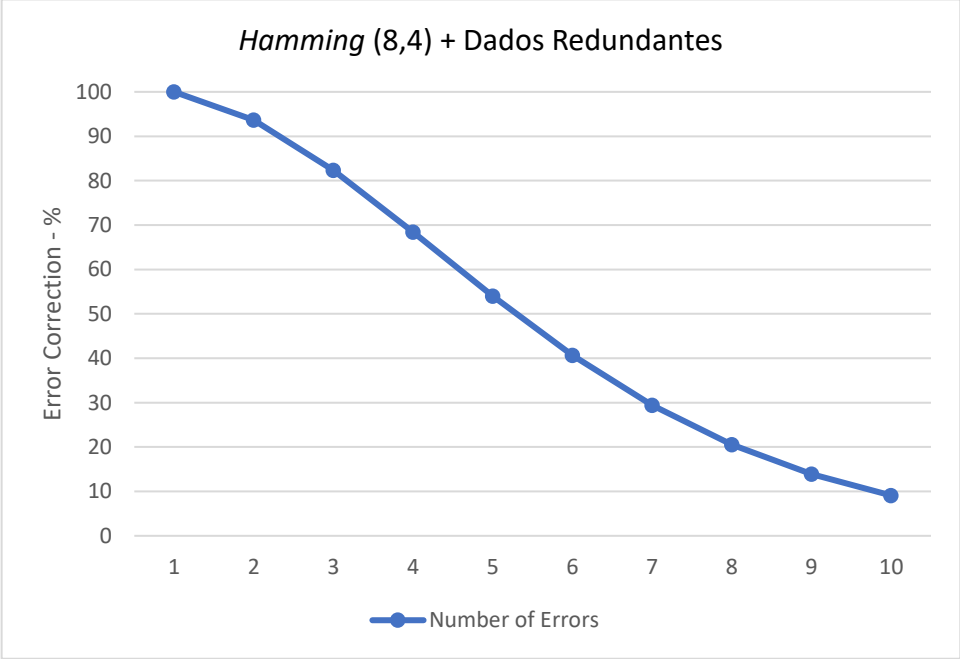


Fonte: elaborado pelo autor.

5.2 Código {Dados} (8,4) + Redundância

O resultado da segunda organização proposta neste trabalho, pela Figura 10, é evidenciado no Gráfico 2, sendo a simulação realizada com o *Hamming* (8,4) estendido, porém, nesse caso específico, a organização dos dados apresenta os dados redundantes linha a linha. A inserção do número de erros variou de 1 a 10 durante a simulação.

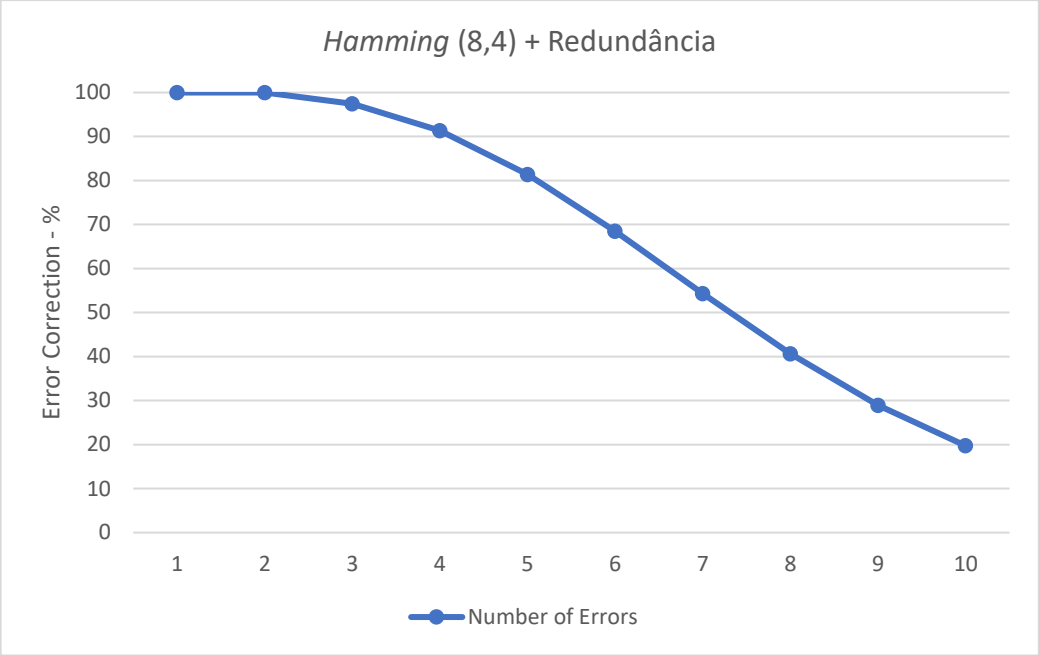
Gráfico 2 – {Dados} (8,4) + Redundância (a)



Fonte: elaborado pelo autor.

No Gráfico 2, observa-se o primeiro resultado para essa variação, sendo um gráfico descendente e de referência, pois ainda não foi aplicada nenhuma modificação no algoritmo de decodificação para correção de erros. Assim, pode-se prosseguir para o Gráfico 3, que nos evidencia uma modificação no algoritmo que usa o código de *Hamming* em sua codificação e decodificação. Desse modo, no processo de decodificação, foi possível detectar e corrigir 100% de erros duplos, conforme Gráfico 3 e de acordo com a Tabela 3.

Gráfico 3 – Gráfico {Dados} (8,4) + Redundância (b)

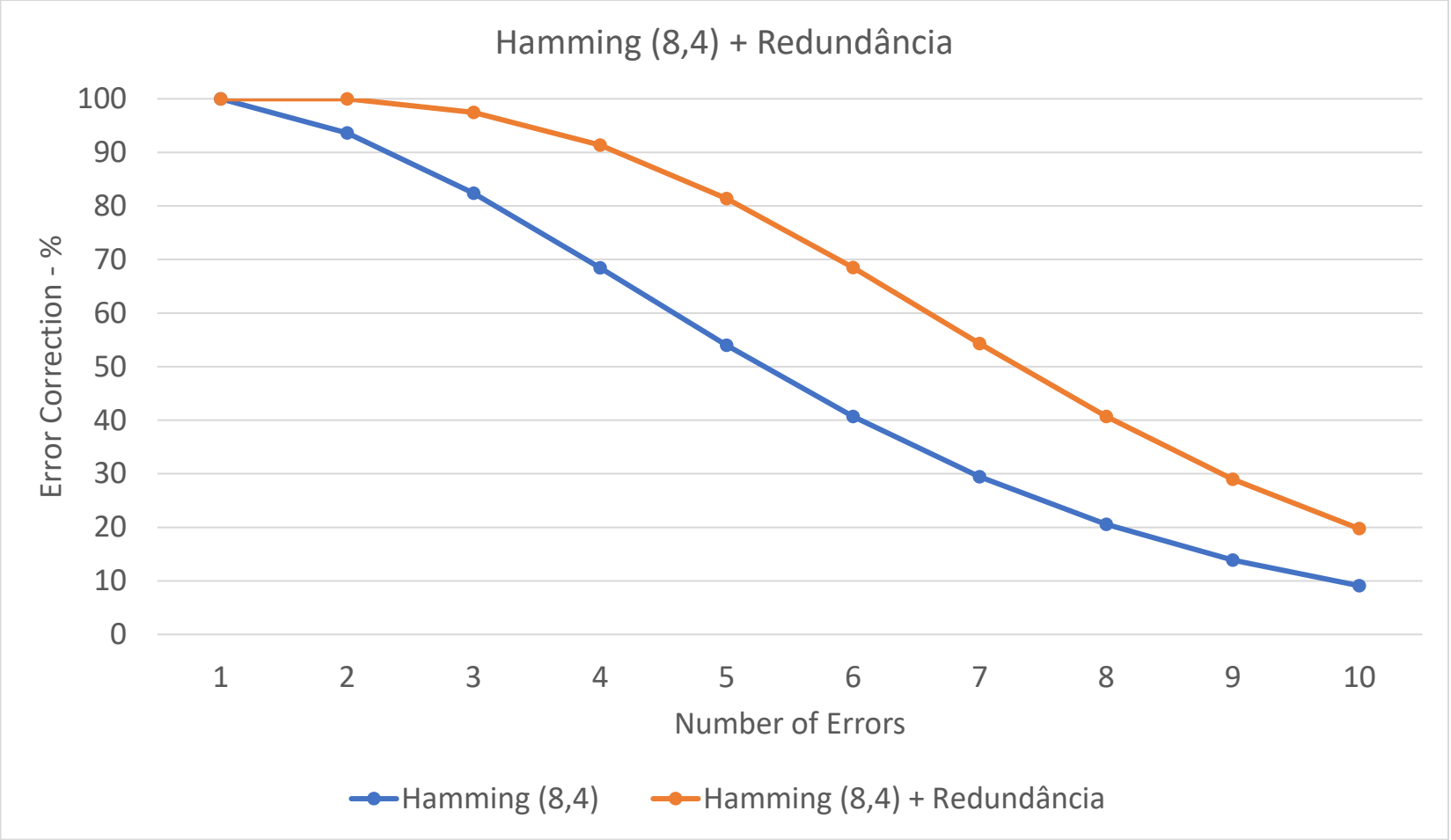


Fonte: elaborado pelo autor.

Assim, durante o processo de decodificação, quando é detectada a presença de erros duplos, caso em que a síndrome foi diferente de 0 (zero) e a paridade foi igual a 0 (zero). O algoritmo realizou a substituição dos dados pelos dados redundantes, o que causa uma suavização do decaimento do gráfico descendente de acordo com o Gráfico 4

É importante destacar que determinadas aplicações, como as espaciais críticas, necessitam saber se a informação contida na área de redundância de dados está correta, pois, caso apresentem erros, nada adiantaria realizar a substituição nos dados. Destaca-se ainda que, para a decodificação usada, tem-se uma limitação, pois os dados redundantes não possuem qualquer proteção de dados, são apenas dados redundantes. Desse modo, há a detecção de erros duplos, porém não se sabe informar sua localização. Se comparado com Gracia-Morán *et al* (2018), a correção de erros se compara com o CLC, por exemplo. No entanto, não se trata do mesmo processo de codificação. É possível visualizar a diferença das curvas conferindo o Gráfico 4.

Gráfico 4 – Gráfico {Dados} (8,4) + Redundância (c)

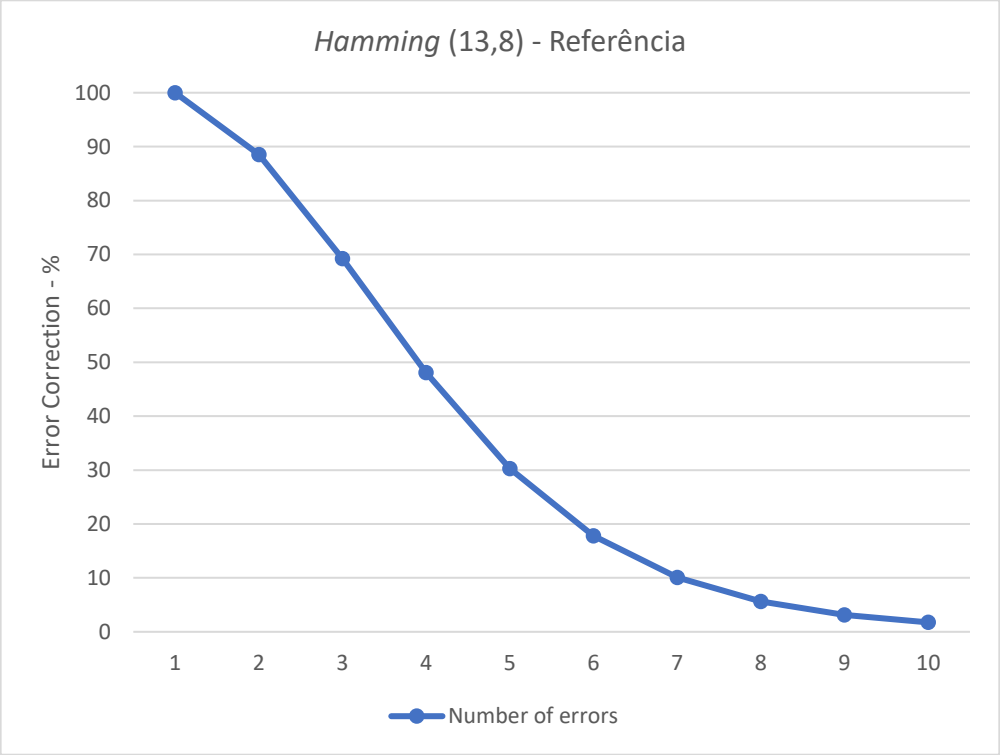


Fonte: elaborado pelo autor.

5.3 Código {Dados + Redundância} (13,8)

Para a terceira organização dos dados, de acordo com a Figura 11, os resultados apresentados foram variados, sendo essa variação positiva para a taxa de correção de erros. Assim, dependendo do tipo de organização dos dados redundantes na matriz, os resultados variaram em torno da referência. Desse modo, são apresentadas as diferenças para a discussão a seguir. O Gráfico 5 é a referência para o processo de decodificação dos dados usando esse tipo de organização. É possível perceber que a curva se mostra descendente, o que faz jus ao código de *Hamming* (13,8) proposto inicialmente, ou seja, ainda não foi aplicada nenhuma modificação no algoritmo de decodificação dos dados e redundância.

Gráfico 5 – {Dados + Redundância} (13,8) - Referência

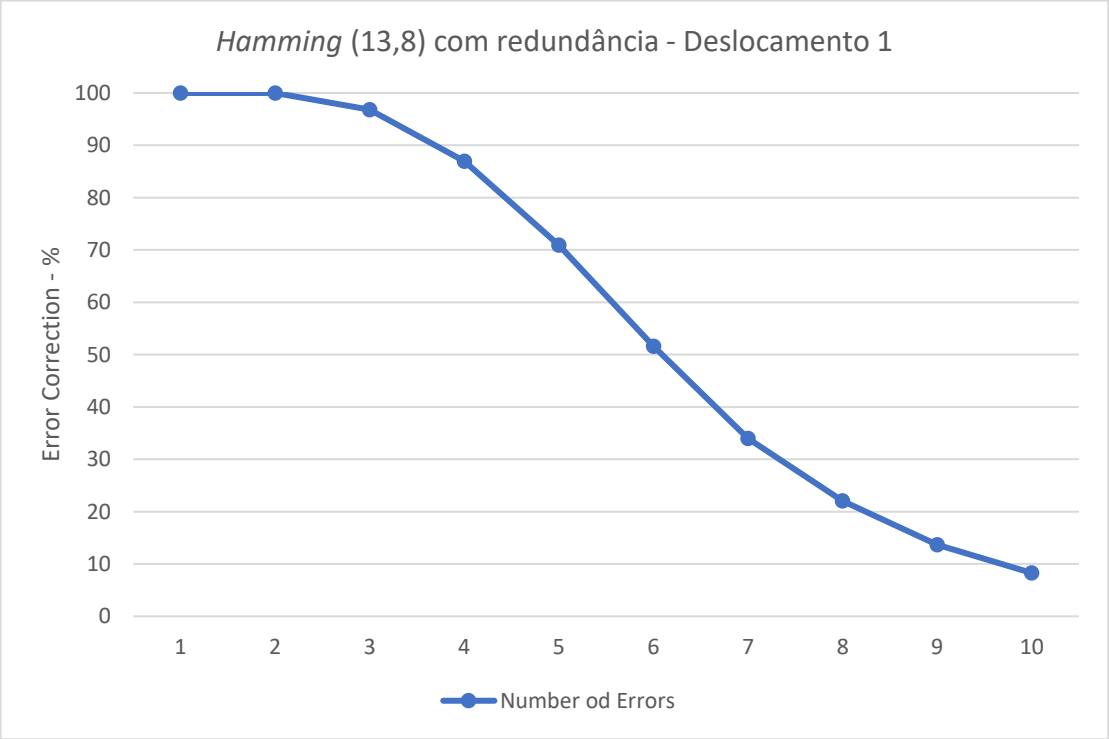


Fonte: elaborado pelo autor.

Para a organização da Figura 13, os dados redundantes, antes do processo de codificação, foram deslocados entre as linhas. Os resultados para os erros simples e duplos, para esse ordenamento de dados, são mostrados no Gráfico 6. Assim, comparando-se com o Gráfico 5, pode-se perceber a correção de 100% de erros simples e duplos. Essa situação se deu pelo fato de a detecção dos erros duplos ser permitida pelo código de *Hamming* estendido. Cabe salientar que a codificação, nessa organização, protege mais *bits*, visto que usa 4 (quatro) *bits*

de verificação. Percebe-se também a suavização da curva que se mostra descendente para até 10 (dez) erros.

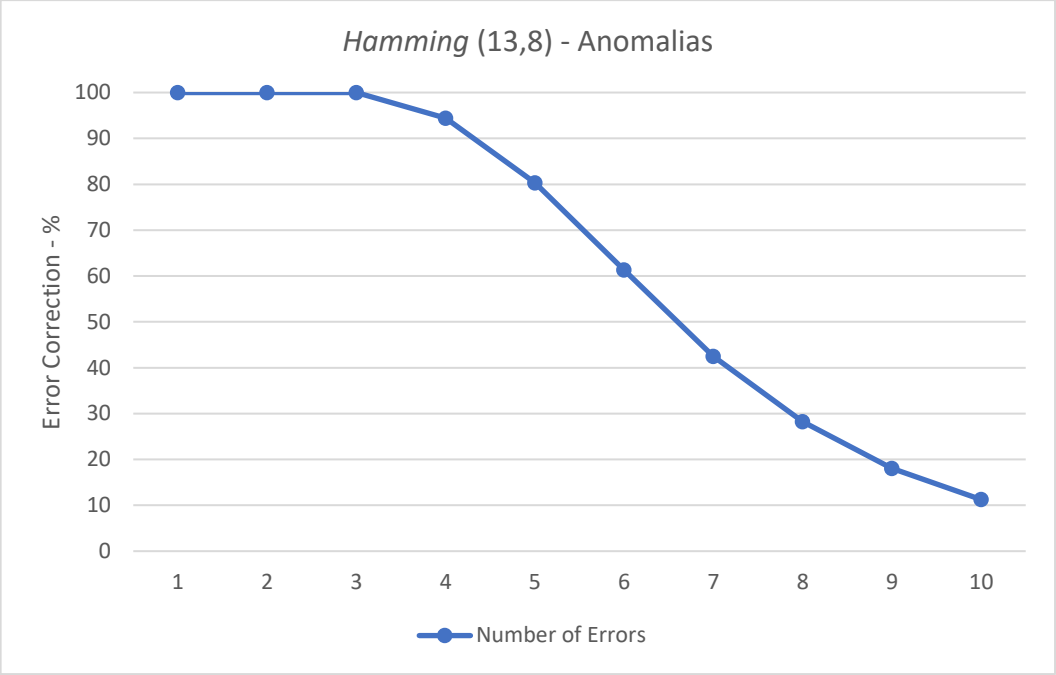
Gráfico 6 – Gráfico {Dados + Redundância} (13,8) – Erro Duplo Deslocamento 1



Fonte: elaborado pelo autor.

Continuando na terceira organização, observa-se, no Gráfico 7, uma decodificação para a situação de *anomalias*, circunstância que está contemplada na Tabela 3 como condição para o erro triplo, pois a síndrome é igual a 0 (zero) e a paridade é igual a 1 (um). Desse modo, durante a decodificação na FASE 2, de acordo com a Figura 21 e considerando que a linha deslocada de redundância para os dados **D_{00}, D_{01}, D_{02}, D_{03}** não apresenta erros em seus dados, há uma substituição dos dados pelos dados redundantes que foram deslocados entres as linhas. Consequentemente, para essa situação, é possível corrigir 100% dos casos de 3 (três) erros nos dados de acordo com o Gráfico 7. Assim, para essa organização, existem várias alternativas a serem exploradas, as quais serão apresentadas em seguida.

Gráfico 7 – Gráfico {Dados + Redundância} (13,8) – Erro Triplo Deslocamento 1

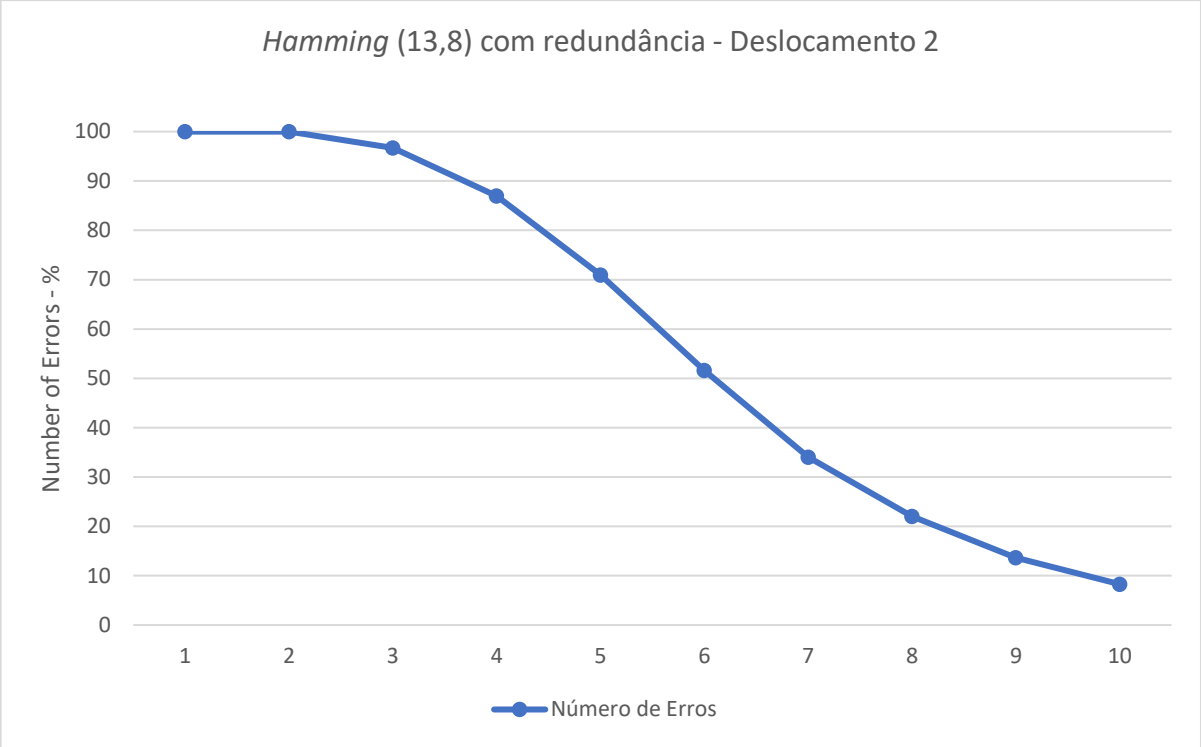


Fonte: elaborado pelo autor.

Ainda para a organização três, no Gráfico 11, observam-se as três variações de decodificação usadas para o ordenamento em questão, assim, é evidenciada uma evolução na taxa de correção de erros e na suavização das linhas para erros duplos e triplos se comparado com a linha de referência.

Quanto aos resultados para a organização da Figura 14, em que a distância para o dado correspondente está maior, os resultados foram melhores quando comparados com o primeiro deslocamento. Esse resultado pode ser observado por meio do Gráfico 8, que mostra uma discreta melhora na curva de decaimento indicando uma melhor taxa de correção nos dados em geral. É possível observar a evolução na exploração, conforme pode ser visto no Gráfico 12, em que se apresentam as curvas para os dois deslocamentos, assim, o deslocamento 2 apresentou também a correção de 100% de 3 (três) erros como também uma melhor taxa de correção de erros se comparada com o deslocamento 1.

Gráfico 8 – Gráfico {Dados + Redundância} (13,8) – Erro Duplo Deslocamento 2



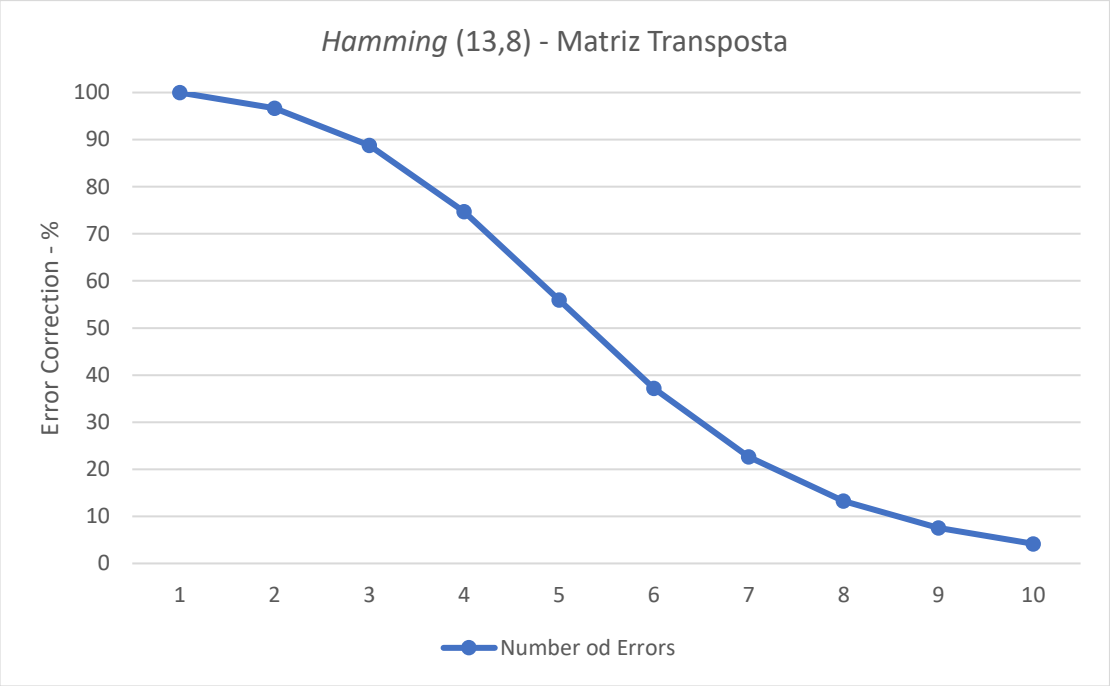
Fonte: elaborado pelo autor.

Para essa organização, ainda foram realizadas simulações de acordo com a Figura 15, a qual evidencia a transformação dos dados redundantes na matriz transposta; e com a Figura 16, a qual mostra que, além da transformação em matriz transposta, houve a realização da rotação da diagonal secundária. Assim, conforme visualizado no Gráfico 9, não houve a correção de 100% dos erros de 2 *bits* para a organização da matriz transposta, apresentando uma menor eficácia, quando comparado com as organizações de deslocamento entre as linhas. Porém, é possível perceber que ainda se pode avançar no processo exploratório, pois, para essa organização, é realizável implementar um algoritmo mais robusto, como o proposto pelo CLC, que utiliza linhas e colunas em sua codificação (GRACIA-MORÁN, 2018).

O possível processo exploratório a mais pode ser observado se houver a rotação da diagonal secundária dos dados redundantes, como mostrado no Gráfico 10, o qual está de acordo com a Figura 16. Com a rotação da diagonal secundária, foi possível corrigir, novamente, 100% dos erros duplos.

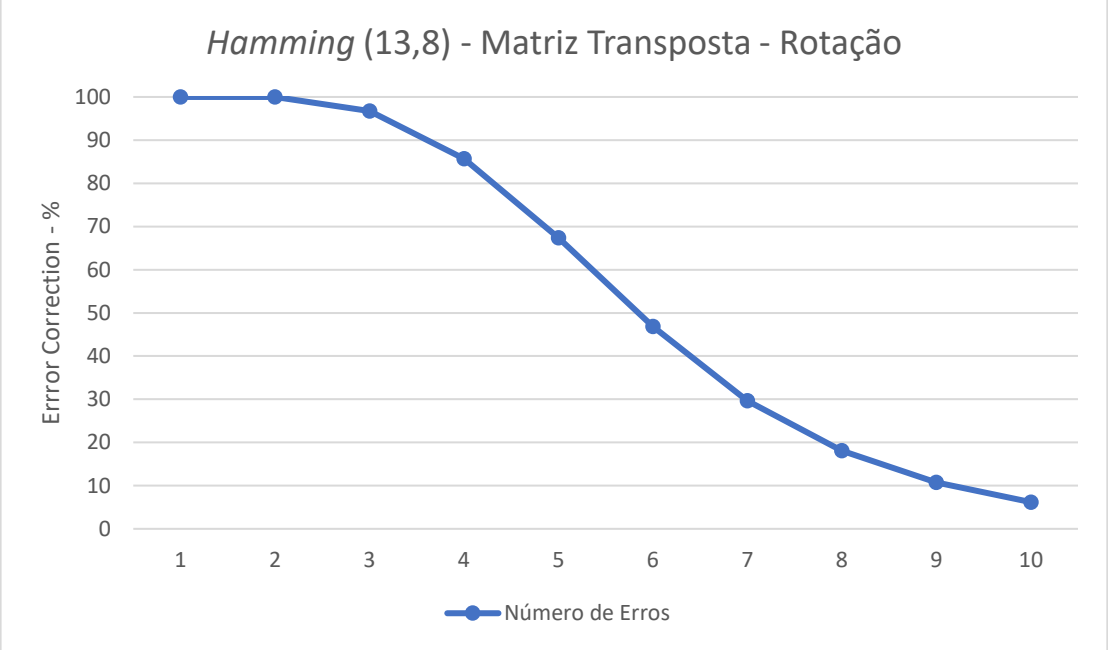
Por último, no Gráfico 13, observam-se todas as linhas de resultados para a exploração do *Hamming* (13,8) com redundância e suas diferentes organizações nos dados redundantes. Percebe-se que existe algo a ser explorado para a correção de erros contando com dados redundantes, o que pode ser pesquisado implementando um código matricial.

Gráfico 9 – Gráfico {Dados + Redundância} (13,8) – Matriz Transposta



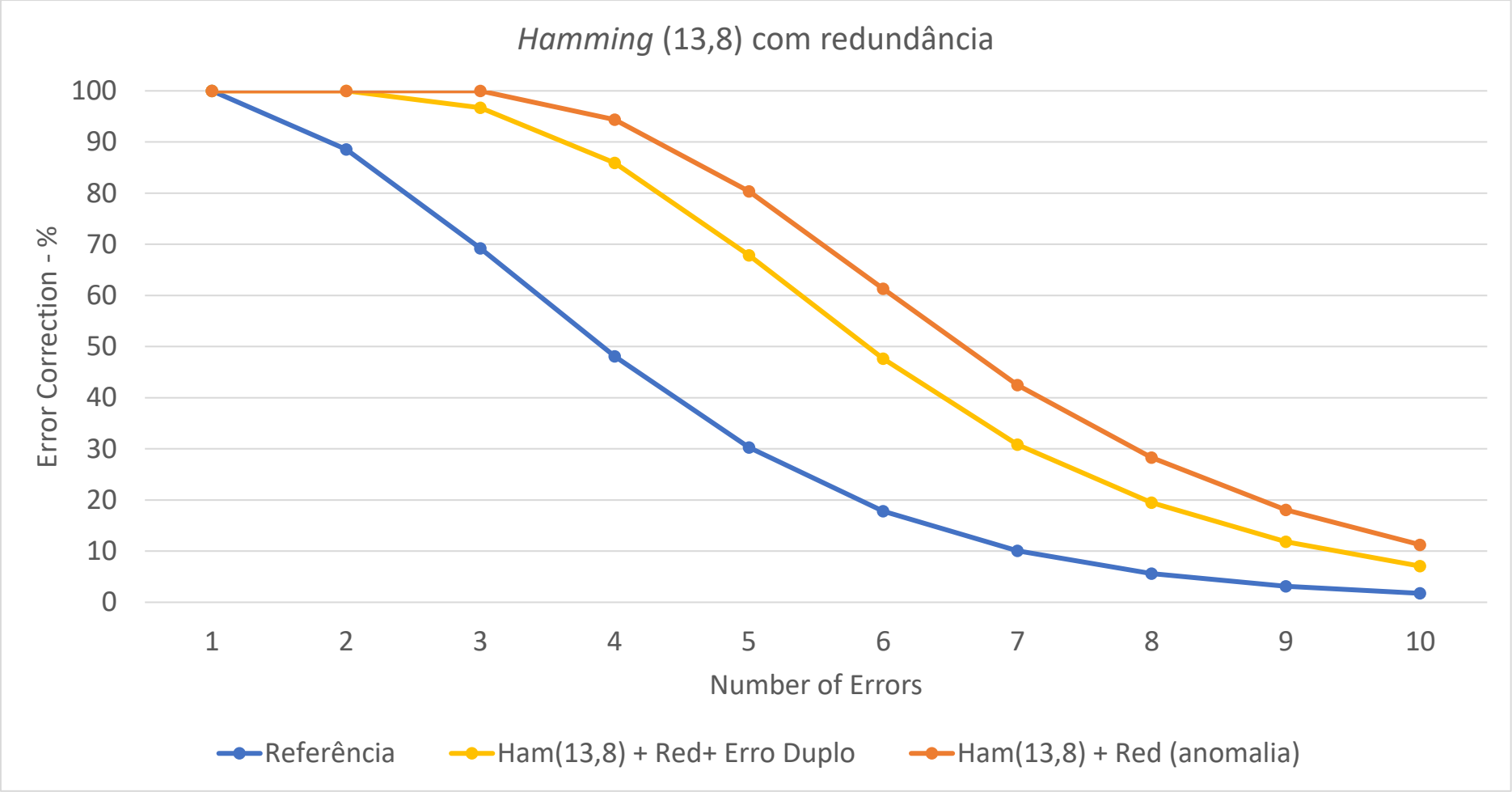
Fonte: elaborado pelo autor.

Gráfico 10 – Gráfico {Dados + Redundância} (13,8) – Matriz Transposta Rotacionada



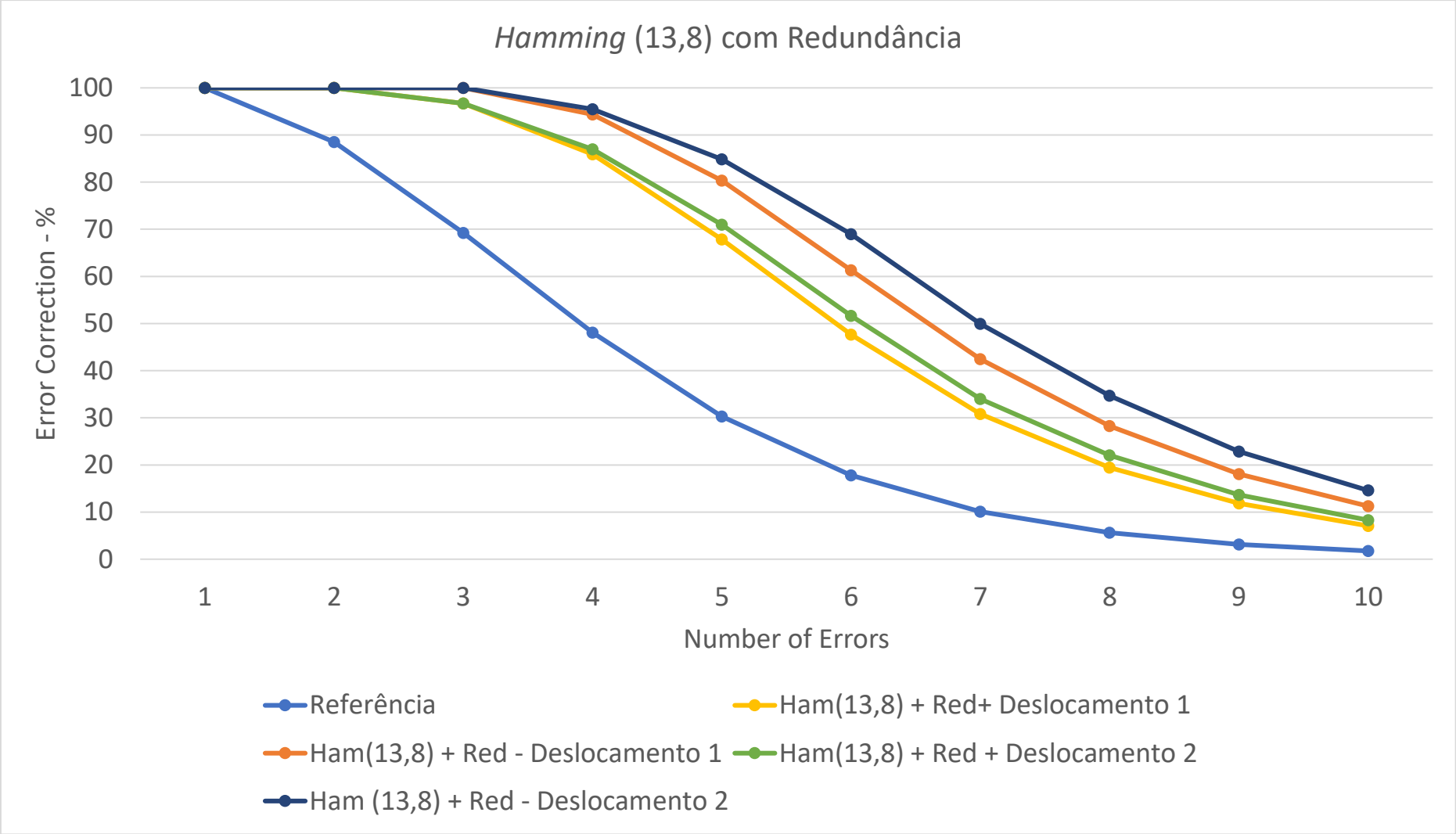
Fonte: elaborado pelo autor.

Gráfico 11 – Gráfico {Dados + Redundância} (13,8) Deslocamento 1



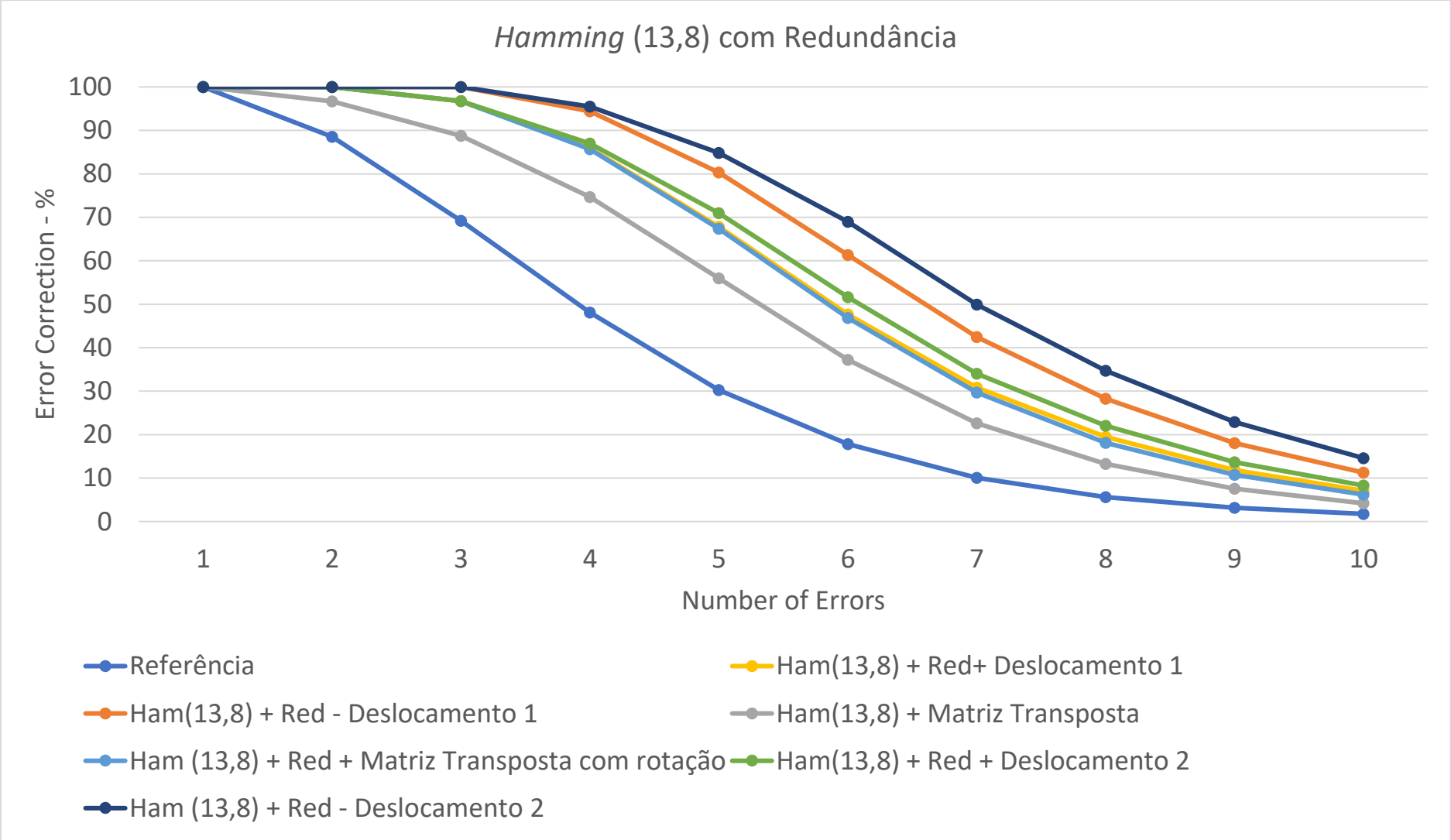
Fonte: elaborado pelo autor.

Gráfico 12 – Gráfico {Dados + Redundância} (13,8) Deslocamentos 1 e 2



Fonte: elaborado pelo autor.

Gráfico 13 – Gráfico {Dados + Redundância} (13,8) – Todas as organizações (a)

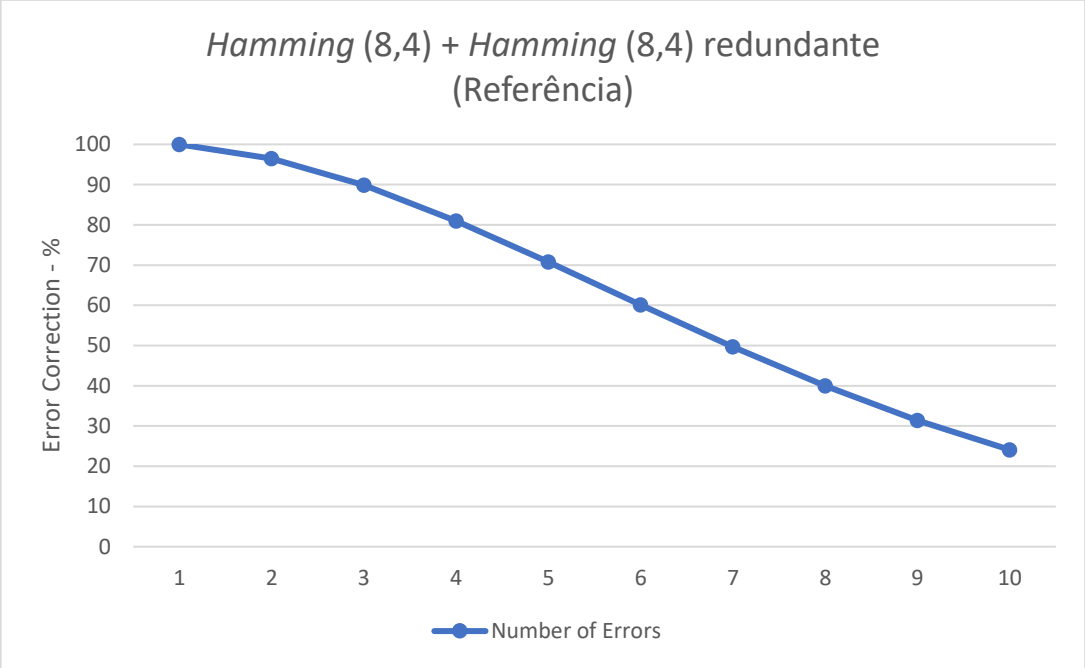


Fonte: elaborado pelo autor.

5.4 Código {Dados} (8,4) + {Redundância} (8,4)

Os resultados para a quarta organização dos dados, de acordo com a Figura 17, evidenciam-se por meio dos resultados dos gráficos a seguir. Assim, no Gráfico 14, que é usado como referência, é possível perceber que a linha segue o mesmo padrão das organizações anteriores, o que muda é a quantidade de combinações nessa organização, que precisará de um número maior de erros para a curva chegar a zero, devido à matriz ser maior. Nesse caso, ainda não foi realizada nenhuma codificação a mais, apenas o *Hamming* (8,4) estendido.

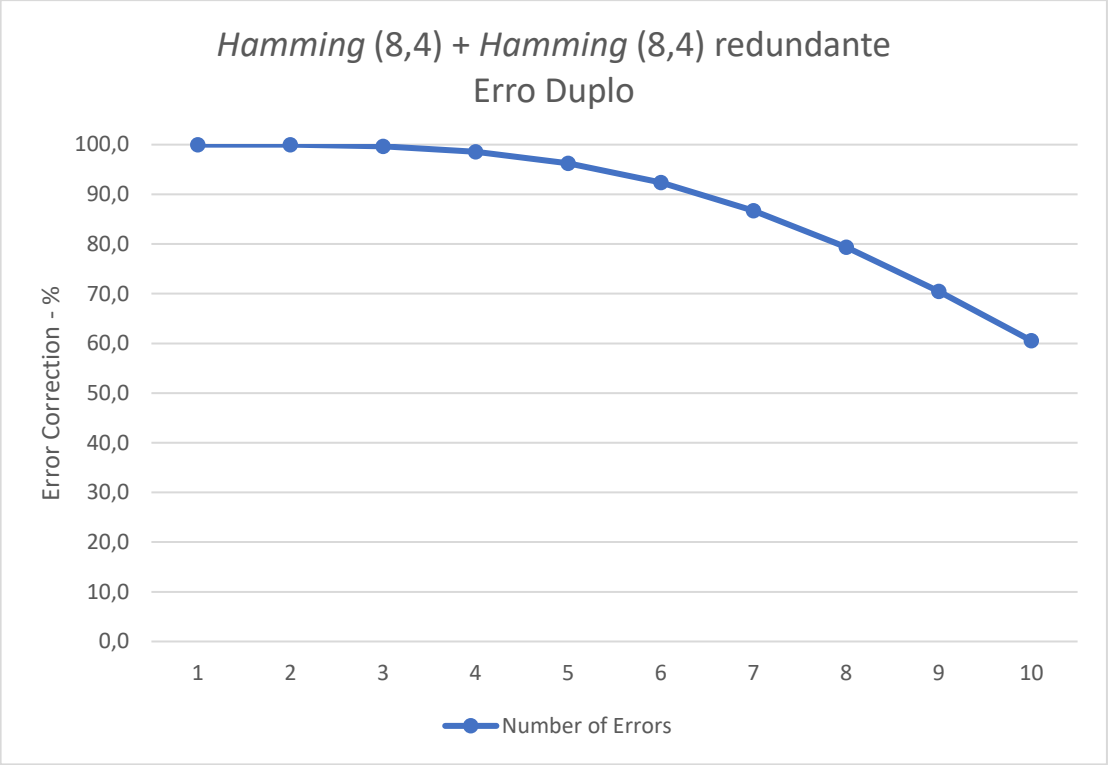
Gráfico 14 – Gráfico {Dados} (8,4) + {Redundância} (8,4) Referência



Fonte: elaborado pelo autor.

O próximo resultado é visualizado no Gráfico 15, em que o processo de decodificação é realizado de acordo com a Figura 22 (Fase 1). Nesse caso, a decodificação consegue corrigir 100% de erros duplos, situação em que apenas mudam os dados pelos dados redundantes, caso seja detectado o erro duplo. E pelo número de combinações dessa organização, chega-se à correção de 99,67% para os casos de 3 (três) erros, deixando de corrigir em torno de 0,33% dos casos de erros na matriz de dados e sendo bastante eficiente e com uma decodificação que aumenta muito a capacidade de detecção de erros.

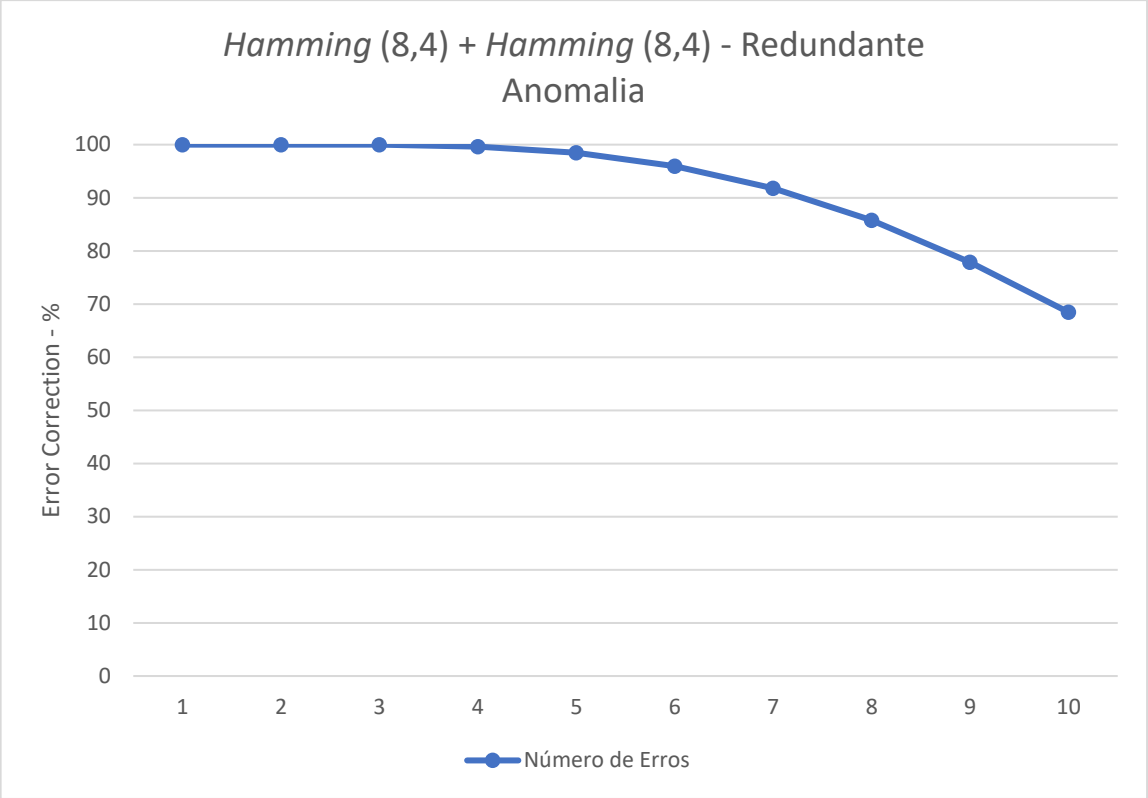
Gráfico 15 – Gráfico {Dados} (8,4) + {Redundância} (8,4) – Erro Duplo



Fonte: elaborado pelo autor.

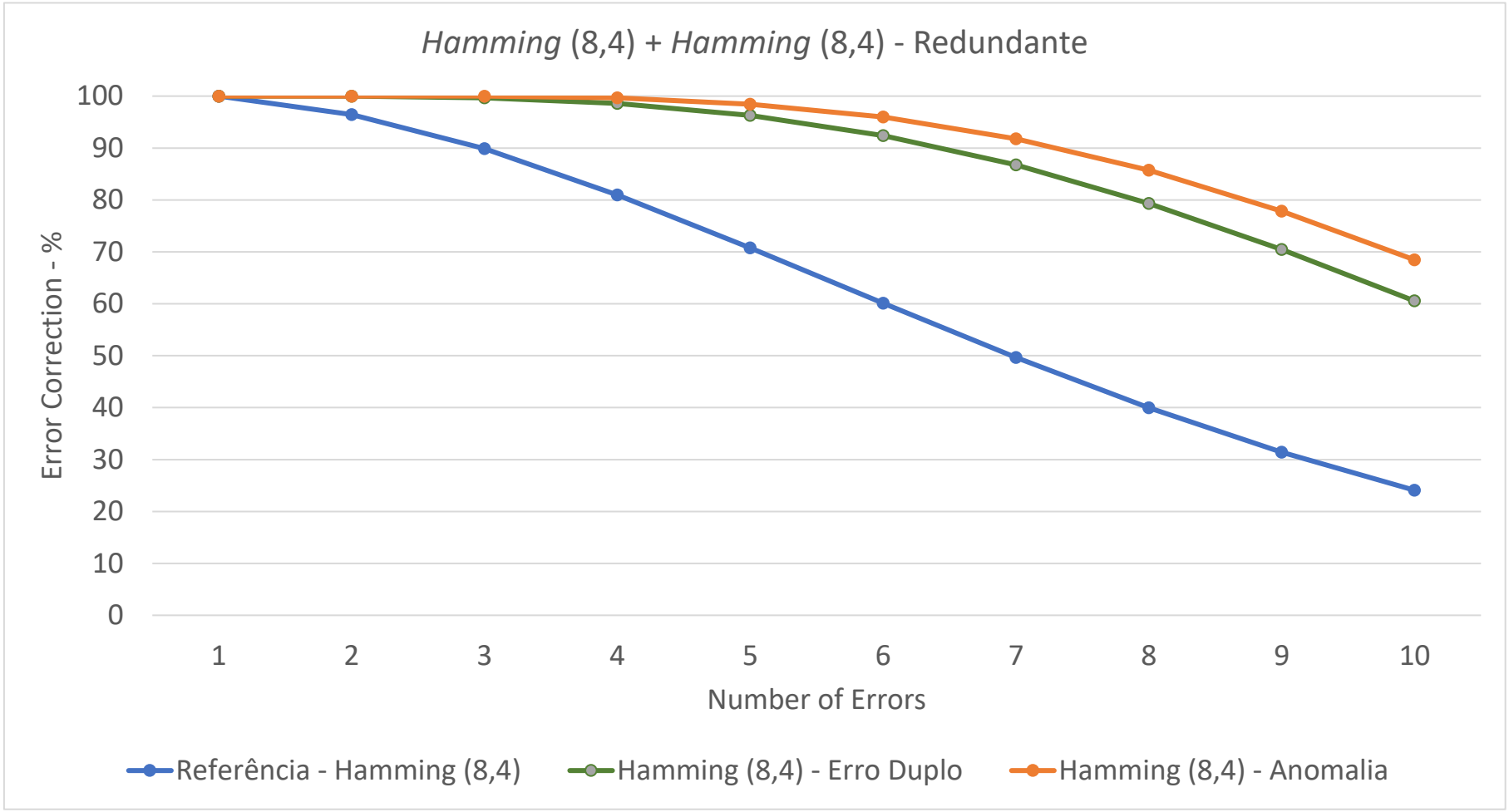
Seguindo nos resultados da quarta organização, tem-se, no Gráfico 16, o resultado desse arranjo, de acordo com a Figura 17, para a Fase 2 (Figura 23). Nesse caso, há a correção de 100% dos casos de 2 (dois) e de 3 (três) erros. Além disso, considerando também o número total de combinações para 4 (quatro) erros, o processo de decodificação deixa de corrigir 0,36% dos casos, ou seja, 98,57% para 4 (quatro) *bits* de erros. Por fim, para essa organização, tem-se o Gráfico 17 que exibe todas as linhas para esse resultado. Assim, para as fases 1 e 2 do processo de decodificação, as curvas têm o comportamento bem parecido, porém, é possível perceber uma maior correção para a condição de anomalia, quando acontecem 3 (três) *bits* de erros. Um fator relevante a ser considerado também é que, para a fase 2 (dois), só há a correção dos dados pelos dados redundantes, se, e somente se, a síndrome dos dados redundantes para a linha correspondente dos dados for igual a 0 (zero), ou seja, isso garante que não há a presença de erros. Tal situação pode ser bem utilizada em situações que exijam um alto grau de confiabilidade nos sistemas, como nos sistemas para aplicações aeroespaciais.

Gráfico 16 – Gráfico {Dados} (8,4) + {Redundância} (8,4) – Erro Triplo



Fonte: elaborado pelo autor.

Gráfico 17 – Gráfico {Dados} (8,4) + {Redundância} (8,4) – Todas as organizações (b)



Fonte: elaborado pelo autor.

6 CONCLUSÃO

Este trabalho propôs a exploração dos códigos corretores de erros com redundância a partir de várias formas de organizações para os dados redundantes unidimensionais e de diversas formas de decodificações. Nessa esteira, foi possível obter conhecimentos aprofundados sobre os códigos que utilizam *Hamming* em sua codificação e decodificação, como também organizações dos dados redundantes nas matrizes que podem acarretar variações nas taxas de correção de erros em *bits* de dados de acordo com determinada forma de organizá-los na matriz. A exploração evidenciou o estudo aprofundado usando determinadas organizações de dados e redundância (limitando o escopo deste trabalho) como também o conhecimento de como se pode avançar no desenvolvimento de códigos capazes de corrigir mais *bits* de erros durante o processo de decodificação de dados.

Além disso, foi possível desenvolver alguns processos de decodificações para as diversas organizações de dados que poderão corrigir entre 2 (dois) a 3 (três) *bits* de dados em 100% dos casos. Cumpre destacar que, para a última organização de dados, a taxa de correção para erros de 4 (*bits*) chegou a quase 100% dos casos.

Os objetivos deste trabalho foram alcançados, sendo o processo de exploração não só importante como também necessário para se avançar nos estudos futuros, como no desenvolvimento de códigos matriciais que podem trazer grandes benefícios para as organizações que implementaram a matriz transposta e a matriz transposta rotacionada.

Percebe-se que se pode explorar organizações espaciais de todas as formas dependendo apenas do tamanho da matriz e da forma de organização para a formação de códigos corretores de erros.

6.1 Trabalhos futuros

Este trabalho foi um estudo inicial para o desenvolvimento de códigos mais robustos e para a verificação de até onde se pode ir quando se tem redundância de dados no processo de codificação e decodificação, usando o *Hamming* organizado espacialmente. Assim, o avanço do conhecimento se dará com o estudo e o desenvolvimento de um código matricial, o qual consiga ter uma melhor taxa de correção para todas as organizações apresentadas. Além disso, a forma de organizar os dados se mostra importante, mas não se sabe até onde se pode avançar ou se tem um limite na taxa de correção de dados, assim, é preciso se aprofundar ainda

mais nos estudos. Outra avaliação futura será a relacionada à área e à potência, pois, devido ao fator tempo e à pandemia, a qual está ocorrendo, não houve tempo hábil para mais avaliações.

REFERÊNCIAS

AVIZIENIS, A.; LAPRIE, J. C.; RANDELL, B. **Fundamental concepts of denpendability**. [*S.l.*]: Research Report NO1145, 2001. Disponível em: https://www.cs.rutgers.edu/~rmartin/teaching/spring03/cs553/readings/avizienis00.pdf. Acesso em: 20 mar. 2021.

BAUMANN, R. C. Radiation-induced soft errors in advanced semiconductor technologies. **IEEE Transactions on Device and Materials Reliability**, [*S.l.*], v. 5, p. 305-316, 2005. Disponível em: https://ieeexplore.ieee.org/ abstract/document/1545891. Acesso em: 1 mar. 2021.

BROSSER, F.; MILH, E. **SEU mitigation techniques for advanced reprogrammable fpga in space**. [*S.l.*]: [s.n.], 2014.

CASTRO, H. D. S. *et al*. A correction code for multiple cells upsets in memory devices for space applications. *In.*: 14th IEEE INTERNATIONAL NEW CIRCUITS AND SYSTEMS CONFERENCE (NEWCAS), 2016, Vancouver. **Proceedings** [...], Vancouver, p. 1-4, 2016.

CHEN, C. L.; HSIAO, M. Y. Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review. **IBM Journal of Research and Development**, [*S.l.*], v. 28, n. 2, p. 124-134, 1984.

DAS, A.; TOUBA, N. A. A New Class of Single Burst Error Correcting Codes with Parallel Decoding. **IEEE Transactions on Computers**, [*S.l.*], v. 69, p. 253-259, 2020.

FOROUZAN, B. A. **Comunicação de dados e redes de computadores**. Tradução de Ariovaldo G., Sophia C. F. e Oliveria J. S. 4. ed. São Paulo: Mc Granw Hill, v. II, 2008.

FOUNDATION, E. **Software IDE ECLIPSE**. [*S.l.*]: [s.n.], 2020. Disponivel em: https:// www.eclipse.org/. Acesso em: 20 mar. 2021.

FREITAS, C. *et al*. LPC: An error correction code for mitigating faults in 3D memories. **IEEE Transactions on Computers**, [*S.l.*], 2020.

GHERMAN, V. *et al*. Programmable extended SEC-DED codes for memory errors. *In.*: 29º SIMPÓSIO DE TESTES IEEE VLSI. 29., 2011, Dana Point. **Proceedings** [...]. Dana Point, p. 140-145, 2011. Disponível em: https://www.researchgate.net/publication/221202591_ Programmable_extended_SEC-DED_codes_for_memory_errors. Acesso em: 20 mar. 2021.

GOMES, G. G. R. **Cesarkallas.net**. [S.l.]: [s.n.], 2011. Disponivel em: http://www.cesarkallas.net/arquivos/faculdade-pos/TP301-codificacao-fonte/2_Bloco_V2011_Rev4.pdf. Acesso em: 27 fev. 2021.

GRACIA-MORÁN, J. *et al*. Improving Error Correction Codes for Multiple-Cell Upsets in Space Applications. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, [*S.l.*], v. 26, n. 10, p. 2132-2142, out., 2018. Disponível em: https://ieeexplore.ieee.org/document/8370138. Acesso em: 27 fev. 2021.

GRANHAUG, K.; AUNET, S. Improving Yield e Defect Tolerance in Subthreshold CMOS Through Output-Wired Redundancy. **J Electron Test**, [*S.l.*], v. 24, p. 157-163, 2008. Disponivel em: https://doi-org.ez11.periodicos.capes.gov.br/10.1007/s10836-007-5027-1. Acesso em: 27 fev. 2021.

HAMMING, R. W. Error detecting and error correcting codes. **The Bell System Technical Journal**, [*S.l.*], v. 29, n. 2, p. 147-160, abr., 1950.

HAYKIN, S.; MOHER, M. **Sistemas de comunicação**. Tradução de Gustavo Guimarães Parma. 5. ed. Porto Alegre: Bookman, 2011.

KLOCKMANN, A.; GEORGAKOS, G. ; GOESSEL, M. **A new 3-bit burst-error correcting code**. [*S.l.*]: IEEE, 2017. Disponível em: https://ieeexplore.ieee.org/document/8046167. Acesso em: 26 fev. 2021.

KOREN, I. Hamming Code Simulator. **University of Massachusetts Amherst**. Massachusetts: ECS, 2021. Disponivel em: http://www.ecs.umass.edu/ece/koren/Fault TolerantSystems/simulator/ Hamming/HammingCodes.html. Acesso em: 26 fev. 2021.

LI, J. *et al*. Extending 3-bit Burst Error-Correction Codes With Quadruple Adjacent Error Correction. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, [*S.l.*], v. 26, n. 2, p. 221-229, fev., 2018.

MOON, T. K. **Error Correction Coding:** Mathematical Methods and Algorithms. 1. ed. Nova York, EUA: Wiley-Interscience, 2005.

MOREIRA, J.; FARRELL, P. G. **Essentials of error control coding. Wilwy.** [S.l.]: Wes Sussex, John Wiley & Sons, Ltd., p. 64-65, 2006.

NEUBAUER, A.; FREUDENBERGER, J.; KÜHN, V. **Conding Theory. Algorithms, Architectures, and Applications**. [*S.l.*]: Wiley, John Wiley & Sons, Ltd., p. 27-31, 2007.

PFEIFER, P.; VIERHAUS, H. T. Iterative error correction with double/triple error detection. **Signal Processing – Algorithms, Architectures, Arrangements, and Applications (SPA)**, Poland, p. 14-19, 2016. Disponível em: https://ieeexplore.ieee.org/document/77 63579. Acesso em: 26 fev. 2021.

RADAELLI, D. *et al*. Investigation of multi-bit upsets in a 150 nm technology SRAM device. **IEEE Transactions on Nuclear Science**, [*S.l.*], v. 52, p. 2433-2437, 2005. Disponível em: https://ieeexplore.ieee.org/document/1589220. Acesso em: 27 fev. 2021.

RICHTER, M.; OBERLAENDER, K.; GOESSEL, M. **New Linear SEC-DED Codes with Reduced Triple Bit Error Miscorrection Probability**. Greece: [s.n.], 2008. Disponível em: https://ieeexplore.ieee.org/document/4567057. Acesso em: 27 fev. 2021.

SHU, L. **Error control coding:** fundamentals and applications. 1. ed. Londres: Pearson, 1983.

SILVA, F. *et al*. An extensible code for correcting multiple cell upset in memory arrays. **J Electron Test**, [*S.l.*], n. 34, p. 417-433, 2018. Disponível em: https://link.springer.com/article/10.1007/s10836-018-5738-5 Acesso em: 27 fev. 2021.