UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE SÃO JOÃO DA BOA VISTA

LEONARDO TERÇAS

Codificação de Canal para Comunicações Ultra Confiáveis em Sistemas 5G

São João da Boa Vista

2019

LEONARDO TERÇAS

Codificação de Canal para Comunicações Ultra Confiáveis em Sistemas 5G

Trabalho de Graduação apresentado ao Conselho de Curso de Graduação em Engenharia Eletrônica e de Telecomunicações do Campus de São João da Boa Vista, Universidade Estatual Paulista "Júlio de Mesquita Filho", como parte dos requisitos para obtenção do diploma de Graduação em Engenharia Eletrônica e de Telecomunicações .

Orientador: Profa. Dra. Cintya Wink de Oliveira Benedito

São João da Boa Vista
2019

Terças, Leonardo

Codificação de canal para comunicações ultra confiáveis em sistemas 5G / Leonardo Terças -- São João da Boa Vista, 2019.
79 p. : il. color.

Trabalho de Conclusão de Curso – Câmpus Experimental de São João da Boa Vista – Universidade Estadual Paulista "Júlio de Mesquita Filho".
Orientadora: Profa. Dra. Cintya Wink de Oliveira Benedito
Bibliografia
1. Codificação 2. Códigos corretores de erros (Teoria da informação) 3. Sistemas de comunicação sem fio 4. Teoria da informação 5. Telecomunicações

CDD 23. ed. – 621.382

Ficha catalográfica elaborada pela Biblioteca-BJB
Bibliotecário responsável: João Pedro Alves Cardoso – CRB-8/9717

UNIVERSIDADE ESTADUAL PAULISTA
“JÚLIO DE MESQUITA FILHO”
CÂMPUS EXPERIMENTAL DE SÃO JOÃO DA BOA VISTA
GRADUAÇÃO EM ENGENHARIA ELETRÔNICA E DE TELECOMUNICAÇÕES

TRABALHO DE CONCLUSÃO DE CURSO

CODIFICAÇÃO DE CANAL PARA COMUNICAÇÕES ULTRA CONFIÁVEIS EM SISTEMAS 5G

Aluno: Leonardo Terças

Orientador: Prof.ª Dr.ª Cintya Wink de Oliveira Benedito

Banca Examinadora:
- Cintya Wink de Oliveira Benedito (Orientadora)
- Edgar Eduardo Benitez Olivo (Examinador)
- Ivan Aritz Aldaya Garde (Examinador)

A ata da defesa com as respectivas assinaturas dos membros encontra-se no prontuário do aluno (Expediente nº 22/2019)

São João da Boa Vista, 05 de dezembro de 2019

Dedico este trabalho ao meu pai Paulo Rogério Terças, à minha mãe Sandra Regina Mirabelli Terças e ao meu irmão Rafael Terças, por todo incentivo, esforço e confiança depositados.

AGRADECIMENTOS

À minha família por toda confiança, carinho e esforço, que mesmo longe se mostraram presentes, me incentivando e dando forças em todos os momentos desta fase da minha vida.

À minha orientadora, Profª. Dra. Cintya Wink de Oliveira Benedito, por toda disposição, atenção, paciência e orientação durante o desenvolvimento deste trabalho.

Ao meu co-orientador de iniciação científica, Profº. Dr. Carlos Héracles Morais de Lima, por despertar em mim o interesse pelo estudo, pesquisa e ramo acadêmico.

Aos membros da banca examinadora pela disponibilidade e atenção dedicada ao trabalho, assim como, por suas sugestões.

À UNESP SJBV e todos os professores e funcionários que compartilharam comigo suas experiências.

À FAPESP pelo financiamento da pesquisa que auxiliou no desenvolvimente deste trabalho.

Aos meus amigos da turma 015, por todo o apoio e amizade durante os 5 anos de graduação, principalmente aos meus amigos de grupo Carolina Ronchini, Carlos Zan e Juliana Menzinger pela paciência, companheirismo e noites mal dormidas.

Aos meus veteranos que sempre me incentivaram e não me fizeram desistir da caminhada, mostrando a garra e o caminho para vencer, principalmente à Letícia Fidanza e Melissa Santos pelos sábios conselhos.

Aos meus amigos com quem dividi moradia na graduação, por toda alegria transpassada e histórias inesquecíveis vividas, muito obrigado Thiago Ogawa, Sidney Borges e Bruno Fernandes.

Aos amigos conquistados na UNESP, por trazerem grandes reflexões e tornar a caminhada da graduação menos cansativa, principalmente a Allan Branco, Laura Almeida, Lanna Jeacomini e Marcelo Pereira por sempre estarem ao meu lado.

Aos meus colegas de trabalho na Atlética, que tornaram-se amigos para a vida, muito obrigado pelas experiências compartilhadas e discussões construtivas, principalmente à Felipe Stringa, Guilherme Nascimento, Lucas Viotto e Renato Ferreira por sempre estarem comigo quando precisei.

E por fim à Deus pelo seu carinho e grande amor, que me dá força para seguir em frente em meio as dificuldades encontradas.

Eu só tenho a agradecer!

"Sucesso significa realizar seus próprios sonhos, cantar sua própria canção, dançar sua própria dança, criar do seu coração e apreciar a jornada, confiando que não importa o que aconteceça, tudo ficará bem.

Criar sua própria aventura!"

(Elana Lindquist)

RESUMO

Codificação de canal nada mais é que uma técnica onde bits de redundância são inseridos na mensagem de interesse, de modo que, durante a recepção da mesma, estes possam ser utilizados para detectar e corrigir erros. Esta técnica apresentada por Shannon em 1948, vem sendo alvo de diversas pesquisas, que buscam desenvolver e aprimorar bons códigos corretores de erros, para atingir a capacidade de canal prevista pelo matemático. Visando sistemas de comunicação sem fio de quinta geração (5G), algumas técnicas de codificação de canal se tornaram candidatas para serem utilizadas em suas transmissões. O objetivo deste trabalho, é realizar o estudo e a comparação de duas destas técnicas, os códigos convolucionais e os códigos polares, visando a aplicação em comunicações ultraconfiáveis e de pacotes curtos em sistemas 5G. Assim, foi realizado o estudo dos codificadores e decodificadores destes códigos e uma implementação computacional para avaliar o desempenho dos mesmos. Os resultados mostram que os códigos polares possuem um bom desempenho em correção e detecção de erro, entretanto possuem alta latência, utilizando o decodificador por cancelamento sucessivo. Já os códigos convolucionais se sobressaem em latência, entretanto não atingem as métricas esperadas em correção e detecção de erro.

PALAVRAS-CHAVE: Codificação de Canal. Códigos Convolucionais. Códigos Polares. 5G.

ABSTRACT

Channel coding is nothing more than a technique where redundancy bits are inserted into the message of interest, so that while reception they can be used to detect and correct errors. This technique introduced by Shannon in 1948, has been the subject of several researches that seek to develop and improve good error-correcting codes to achieve the channel capacity predicted by the mathematician. Aiming at fifth generation wireless communication systems (5G), some channel coding techniques have become candidates for use in their transmissions. The objective of this work is to study and compare two codes, convolutional and polar codes, aiming at application in ultra-reliable and short packet communications in 5G systems. Thus, the study of the coders and decoders of these codes and a computational implementation to evaluate their performance was performed. The results show that polar codes perform better in error correction and detection, however they have high latency using the successive cancellation decoder. Convolutional codes, however, perform better in latency, but do not meet expected metrics in error correction and detection.

KEYWORDS: Channel Coding. Convolutional Codes. Polar Codes. 5G.

LISTA DE ILUSTRAÇÕES

Figura 1 Canal binário sem ruído. 19
Figura 2 Canal binário simétrico. 19
Figura 3 Capacidade canal binário simétrico. 20
Figura 4 Canal binário com apagamento. 20
Figura 5 Canal gaussiano. 21
Figura 6 Código de bloco. 22
Figura 7 Máquina de estados (1,1,2). 28
Figura 8 Esquema simplificado da codificação convolucional. 28
Figura 9 Máquina de estados código convolucional (2,1,7). 29
Figura 10 Representação diagrama de estados. 33
Figura 11 Máquina de estados código convolucional (2,1,3). 33
Figura 12 Diagrama de estados código convolucional (2,1,3). 34
Figura 13 Codificação da mensagem **u** = (10111). 34
Figura 14 Diagrama de treliça código convolucional (2,1,3). 35
Figura 15 Codificação da mensagem u = (10111). 35
Figura 16 Diagrama de treliça código convolucional (2,1,2). 38
Figura 17 Decodificação utilizando o algoritmo de Viterbi. 38
Figura 18 Caminho escolhido pela decodificação utilizando o algoritmo de Viterbi. 38
Figura 19 Canal sem memória. 41
Figura 20 Canal para N = 2. 41
Figura 21 Canal para N = 4. 42
Figura 22 Canal para N = 8. 44
Figura 23 Canal geral W_N. 45
Figura 24 $I(W_1)$. 46
Figura 25 $I(W_2)$. 47
Figura 26 $I(W_4)$. 47
Figura 27 $I(W_8)$. 48
Figura 28 Canais polarizados. 49
Figura 29 Exemplo circuito da transformação $F^{\otimes 3}$. 54
Figura 30 Codificação **u**. 55
Figura 31 Exemplo arquitetura de decodificação SC com N = 8. 57
Figura 32 Exemplo decodificação SC com N = 8. 59
Figura 33 Ciclo de *clock* do processador nos decodificador SC. 62
Figura 34 Arquiterura em árvore SC com N = 8 e R = 1/2. 62
Figura 35 Exemplo arquitetura de decodificação SSC com N = 8. 63
Figura 36 Exemplo decodificação SSC com N = 8. 64
Figura 37 Ciclo de *clock* do processador no decodificador SSC. 67

Figura 38 Decodificadores em árvore SSC para diferentes taxas. 67
Figura 39 Simulação da BER para códigos convolucionais com (N,k) = (64,32; 128,64; 256,128; 512,256; 1024,512; 2048,1024) 70
Figura 40 Simulação da BER para códigos polares com (N,k) = (64,32; 128,64; 256,128; 512,256; 1024,512; 2048,1024) 71
Figura 41 BER com taxa 1/2 para (N,k) = (64,32; 128,64; 256,128; 512,256) 72
Figura 42 Latência de processamento com taxa 1/2 73
Figura 43 *Throughput* com taxa 1/2 73
Figura 44 *Clocks* necessários para decodificar diferentes comprimentos de palavras 74
Figura 45 Bits decodificados por *clock* para diferentes comprimento de palavra 75

LISTA DE ABREVIATURAS E SIGLAS

$5G$	Quinta geração das comunicações móveis
$AWGN$	*Additive White Gaussian Noise*
BEC	*Binary Erasure Channel*
BER	*Bit Error Rate*
$BPSK$	*Binary Phase Shift Keying*
BSC	*Binary Symmetric Channel*
IoT	*Internet of Things*
$LDPC$	*Low Density Parity Check*
LR	*Likelihood Ratio*
MTC	*Machine Type Communication*
SC	*Successive Cancellation Decoder*
SSC	*Simplified Successive Cancellation Decoder*
URC	*Ultra Reliability Communications*
$V.A.$	Variável Aleatória

LISTA DE SÍMBOLOS

μ_{A^C}	Bits congelados
μ_A	Bits de informação
B	Banda do canal
$\mathbf{c}$	Palavra-código
$\hat{\mathbf{c}}$	Estimativa de uma palavra código gerada pelo decodificador
$\mathcal{C}$	Código
$B_{\mathcal{C}}$	Base de um subespaço
C	Capacidade do canal
d	Distância mínima de Hamming
D	Variável de atraso unitário
d_h	Distância de Hamming
e	Bit apagamento
E_b	Energia do bit
$F^{\otimes n}$	Produto de Kronecker
F_q	Subespaço de q elementos
$\mathbf{g}^i$	Sequência geradora
$g(D)$	Polinômio gerador de um codificador
G	Matriz geradora
H	Matriz controle de paridade
$H(X)$	Entropia
$I(X;Y)$	Informação mútua
k	Comprimento do código de saída
$L_N^{(i)}(\mathbf{y}_1^N, \mathbf{u}_1^{i-1})$	Razão de verossimilhança
$\mathbf{m}$	Mensagem de interesse
$\hat{\mathbf{m}}$	Estimativa da mensagem recebida pelo receptor

M	Quantidade de memórias do codificador
n	Comprimento do código de entrada
N	Quantidade de canais
N'	Quantidade de simulações realizadas
N_0	Densidade de potência do ruído
N'_A	Número de ocorrências de um evento A
$\aleph$	Variância
$\mathcal{N}_0$	Nós bits congelados
$\mathcal{N}_1$	Nós bits de informação
$\mathcal{N}_R$	Nós combinação de bits congelados e de informação
$\mathcal{P}$	Matriz de ordem $k \times (n-k)$
P	Potência de transmissão
p	Probabilidade de erro
$p(x, y)$	Probabilidade conjunta
$p(x_k)$	Probabilidade de um evento de interessse
P'^t	Matriz P' transposta
q	Quantidade de elementos
$\mathbf{r}$	Mensagem recebida
R	Taxa de codificação
R_n	Matriz permutação de canais
T	Throughput
$\hat{\mathbf{u}}$	Vetor de bits estimados
$u(D)$	Polinômio de entrada de um codificador
$\mathbf{u}$	Vetor de entrada em um codificador
$v(D)$	Polinômio de saída de um codificador
$\mathbf{v}$	Vetor de saída em um codificador
W	Canal sem memória

$w_H(u)$	Peso de Hamming
x	Evento de interesse
X	Variável aleatória discreta
X_i	Entrada canal de tempo discreto
Y	Variável aleatória
Y_i	Saída canal de tempo discreto
Z_i	Ruído canal de tempo discreto

SUMÁRIO

1 INTRODUÇÃO 15

2 REVISÃO DE CONCEITOS 17

2.1 Teoria da Informação 17

2.2 Codificação de Canal 22

2.2.1 Código de Blocos Lineares 22

3 CÓDIGOS CONVOLUCIONAIS 28

3.1 Codificação 28

3.2 Decodificação 36

3.2.1 Algoritmo de Viterbi 37

4 CÓDIGOS POLARES 40

4.1 Polarização de Canal 40

4.2 Codificação 50

4.3 Decodificação 56

4.3.1 Decodificador por Cancelamento Sucessivo 57

4.3.2 Decodificador por Cancelamento Sucessivo Simplificado 63

5 IMPLEMENTAÇÃO, ANÁLISES E RESULTADOS 69

6 CONCLUSÕES 76

REFERÊNCIAS 78

1 INTRODUÇÃO

O sistema de quinta geração de comunicações sem fio 5G não será apenas uma evolução das redes atuais, será uma tecnologia que irá revolucionar a maneira como vivemos, interagimos e fazemos negócios. O sistema 5G promoverá meios técnicos para, "ter a sociedade totalmente conectada" e demandará uma nova estrutura de planejamento, implantação, operação e otimização para interagir plenamente com as novas tecnologias digitais emergentes no mercado. A implementação do sistema 5G está prevista para ocorrer a partir de 2020 e tornou-se grande alvo de diversas pesquisas, que buscam desenvolver e aprimorar técnicas e tecnologias para que a proposta deste sistema seja alcançada. Este sistema irá disponibilizar novos recursos e possibilidades, como os serviços de alta capacidade, a implantação massiva de objetos inteligentes (*Internet of Things* - IoT) e serviços com alta confiabilidade e/ou baixa latência, [TAHIR, SCHWARZ e RUPP 2017, OLIVEIRA e LAMARE 2017].

As aplicações comerciais atuais visam principalmente as comunicações orientadas a humanos, que focam em assegurar uma boa conectividade na maior parte do tempo, entretanto em situações onde o sinal é degradado a taxa de transmissão é nula. Já os sistemas emergentes, cuja transmissões são orientadas a máquinas dependem criticamente da disponibilidade ininterrupta dos enlaces de rádio, mesmo que a qualidade da conexão fique baixa, [5GPPP 2015].

Shannon, em 1948, mostrou que uma comunicação livre de erros em um canal com ruído é possível, desde que uma estratégia de codificação de canal apropriada seja utilizada e a taxa de transmissão de informação seja menor ou igual a capacidade do canal, [SHANNON 1948]. Deste modo, diversas pesquisas vêm sendo desenvolvidas sobre esquemas de codificação de canal, com o objetivo de atingir a capacidade do canal prevista pela teoria de Shannon. A codificação de canal é uma técnica onde bits de redundância são inseridos de maneira estruturada no transmissor (codificação), de modo que no receptor durante a decodificação, esta redundância auxilie a detectar e corrigir possíveis erros adquiridos durante a transmissão. Assim, códigos com alto desempenho de codificação e decodificação são essenciais para os futuros sistemas de comunicação sem fio. Alguns códigos foram selecionados como candidatos para a comunicação sem fio de quinta geração, como os códigos convolucionais, os códigos turbo, os códigos LDPC (em inglês, *Low-density-parity-check*) e os códigos polares [TAHIR, SCHWARZ e RUPP 2017].

Neste trabalho iremos comparar o desempenho de técnicas de codificação de canal em cenários baseados na comunicação entre máquinas (MTC, em inglês *Machine Type Communication*), onde a quantidade de dados de informação é comparável aos dados de controle, ou seja, os sistemas MTC são compostos por pacotes curtos. Estes cenários estarão operando no modo previsto para as comunicações ultra confiáveis (URC, em inglês *Ultra-Reliable Communication*), que é um modo de operação ainda não existente nos sistemas atuais, que prevê a sustentação de uma qualidade de serviço mínima durante quase 100% do período de conexão [METIS 2012 - 2015, METIS 2015 - 2017, POPOVSKI 2015]. Serão avaliadas duas técnicas de codificação de canal, os códigos convolucionais e os códigos polares, pois tais códigos apresentam grandes chances de compor a nova geração, devido a baixa complexibilidade de implementação e o alto desempenho [ISCAN, LENTNER e XU 2016, SYBIS

et al. 2016]. As simulações serão realizadas através de implementação computacional no software MATLAB, onde palavras de comprimento k serão codificadas, transmitidas por um canal AWGN (*Additive White Gaussian Noise*) e decodificadas por um receptor, a fim de avaliar a taxa de erro de bit (BER - *Bit Error Rate*), a latência de processamento, que é o tempo utilizado para codificar e decodificar uma mensagem de interesse, e a taxa de transferência de informação (*Throughput*), que é a quantidade de bits que são decodificados por segundo levando em consideração um ciclo de *clock* do processador [SARKIS e GROSS 2013b, LEROUX et al. 2011].

O presente trabalho está organizado da seguinte maneira: no Capítulo 2, será realizada uma breve revisão sobre a teoria da informação e codificação de canal. No Capítulo 3, abordaremos os códigos convolucionais, exemplificando como é realizada sua codificação e decodificação. No Capítulo 4, descreveremos os códigos polares, abordando a técnica de polarização de canal que o código faz uso, sua codificação e dois tipos de decodificação que podem ser utilizados por este código. No Capítulo 5, serão abordadas as métricas que serão analisadas nas simulações, as implementações computacionais realizadas e os resultados obtidos. Por fim, no Capítulo 6, serão apresentadas as conclusões obtidas pelo presente trabalho.

2 REVISÃO DE CONCEITOS

Neste capítulo iremos realizar uma breve revisão de conceitos importantes que serão necessários para o desenvolvimento do trabalho. Na Seção 2.1 serão apresentados os conceitos de entropia e informação mútua, para enfim definir a capacidade de um canal. E na Seção 2.2 será abordada a codificação de canal, juntamente com o estudo dos códigos de bloco lineares, apresentando conceitos básicos destes códigos e exemplos de aplicação. As definições apresentadas neste capítulo podem ser encontradas em [COVER e THOMAS 2006, RYAN e LIN 2009, HAYKIN 2001, RAPPAPORT 2008].

2.1 TEORIA DA INFORMAÇÃO

A teoria da informação proposta por Shannon [SHANNON 1948], desenvolve estudos de conceitos relacionados a quantificação, armazenamento e transmissão de informação, os quais são de extrema importância para o ramo das comunicações. Uma das medidas mais importantes da teoria da informação é a entropia, este parâmetro pode ser definido como a quantidade média de informação de um evento de interesse, associado a uma variável aleatória. Já informação pode ser definida como a quantidade de incerteza de um processo em termos probabilísticos, onde incerteza, é uma "surpresa" na ocorrência de um evento.

Partindo de um evento de interesse, $x = x_k$ com probabilidade $p(x_k)$, a quantidade de informação pode ser relacionada com o inverso da probabilidade de ocorrência.

Definição 2.1.1 *Define-se a quantidade de informação de um evento de uma variável aleatória discreta $x = x_k$ com probabilidade $p(x_k)$, como*

$$I[x_k] = \log_2 \left[\frac{1}{p(x_k)}\right] = -\log_2 p(x_k). \tag{2.1}$$

Deste modo, a média da quantidade de informação, definida como a medida de informação média por símbolo, pode ser dada da seguinte forma

$$E[I(x_k)] = -\sum_{i=1}^{k} p(x_i) \log_2[p(x_i)]. \tag{2.2}$$

Assim, definimos entropia da seguinte forma.

Definição 2.1.2 *Assumindo X uma fonte discreta e sem memória, com função massa de probabilidade $p(x) = P[X = x]$, com $x = (x_0 \dots x_n)$. A entropia de X é definida da seguinte forma*

$$H(X) = -\sum p(x) \log_2[p(x)]. \tag{2.3}$$

Exemplo 2.1.1 *Considere uma função massa de probabilidade $p(x)$ assumindo os seguintes valores*

$$X = \begin{cases} \frac{1}{2}, & x = a; \\ \frac{1}{4}, & x = b; \\ \frac{1}{4}, & x = c. \end{cases}$$

Utilizando (2.3), temos que o valor de entropia será

$$H(X) = -\tfrac{1}{2}\log_2(\tfrac{1}{2}) - (\tfrac{1}{4})\log_2(\tfrac{1}{4}) - \tfrac{1}{4}\log_2(\tfrac{1}{4}) = \tfrac{3}{2} \textit{ bits.}$$

Outra medida importante da teoria da informação é a informação mútua, que é a quantidade de informação que uma variável aleatória contém em relação a outra variável aleatória, ou seja, é a redução de incerteza de uma variável aleatória por estar relacionada com outra.

Definição 2.1.3 *Sejam X e Y duas variáveis aleatórias com função densidade de probabilidade conjunta dada por $p(x, y)$ e, com funções densidade de probabilidade marginais $p(x)$ e $p(y)$. A informação mútua associada a este par de variáveis aleatórias é dada por*

$$I(X;Y) = \sum_{x,y} p(x,y)\log_2\left[\frac{p(x,y)}{p(x)p(y)}\right]. \tag{2.4}$$

Pode-se simplificar a Equação 2.4 de modo que

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X). \tag{2.5}$$

Esta medida é importante para transmissões de dados pois com ela é possível mensurar a eficiência de um canal. Se as informações enviadas são dependentes, a informação mútua será maior, estabelecendo assim uma comunicação confiável. Caso estas variáveis sejam independentes, haverá redução de incerteza e a informação mútua será nula, caracterizando uma comunicação não confiável.

Em situações onde variáveis aleatórias são contínuas, a entropia diferencial não será uma soma e sim uma integral de valores.

Definição 2.1.4 *Seja X uma variável aleatória contínua, com função densidade de probabilidade $p(x)$. A entropia é obtida por*

$$H(X) = -\int_x p(x)\log_2[p(x)]dx. \tag{2.6}$$

De modo análogo, temos a definição de informação mútua para o caso contínuo.

Definição 2.1.5 *Sejam X e Y variáveis aleatórias contínuas, com função massa de probabilidade conjunta $p(x, y)$ e funções massa de probabilidade marginais $p(x)$ e $p(y)$, temos a seguinte equação*

$$I(X;Y) = \iint_{x,y} p(x,y)\cdot\log_2\left[\frac{p(x,y)}{p(x)p(y)}\right]dxdy. \tag{2.7}$$

Com o conceito de informação mútua podemos definir a capacidade de um canal, que tanto no caso discreto como no caso contínuo, nada mais é do que o limite da taxa em que uma informação pode ser transmitida de forma confiável através de um canal de comunicação.

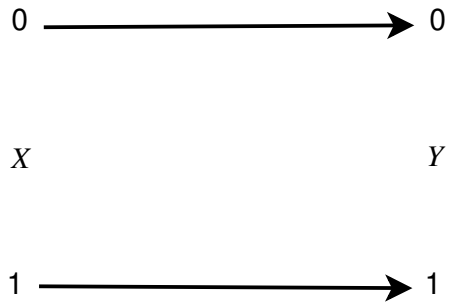
Definição 2.1.6 *Define-se capacidade de canal como sendo a taxa máxima de informação que este canal suporta, levando em consideração todas as possíveis distribuições das prováveis entradas, ou seja, a capacidade C de um canal é*

$$C = \max_{p(x)} I(X;Y). \tag{2.8}$$

Vale salientar, que a capacidade do canal é medida em bits por uso do canal. A seguir, veremos alguns exemplos de canais importantes e o cálculo de suas respectivas capacidades.

Exemplo 2.1.2 *Na Figura 1 podemos observar um canal binário sem ruído, ou seja, um canal ideal.*

Figura 1 – Canal binário sem ruído.

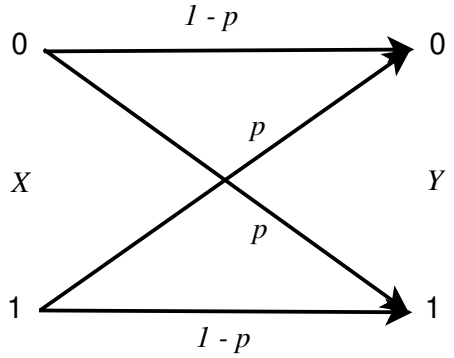


Fonte: Próprio autor.

Neste canal, a mensagem que é transmitida é recebida, ou seja, $P[Y = 1|X = 1] = P[Y = 0|X = 0] = 1$ *e* $P[Y = 0|X = 1] = P[Y = 1|X = 0] = 0$. *Para casos onde* $P[X = 1]$ *e* $P[X = 0]$ *aproximam-se de* 0.5, *a capacidade do canal será* $1\ bit$ *por uso do canal, que é a informação máxima que pode ser enviada.*

Exemplo 2.1.3 *Na Figura 2 temos o chamado Canal Binário Simétrico (Binary Symmetric Channel - BSC). Neste canal a probabilidade de se transmitir um bit 0 e receber um bit 1 é a mesma que se transmitir um bit 1 e receber um bit 0, ou seja,* $P[Y = 0|X = 1] = P[Y = 1|X = 0] = p$.

Figura 2 – Canal binário simétrico.

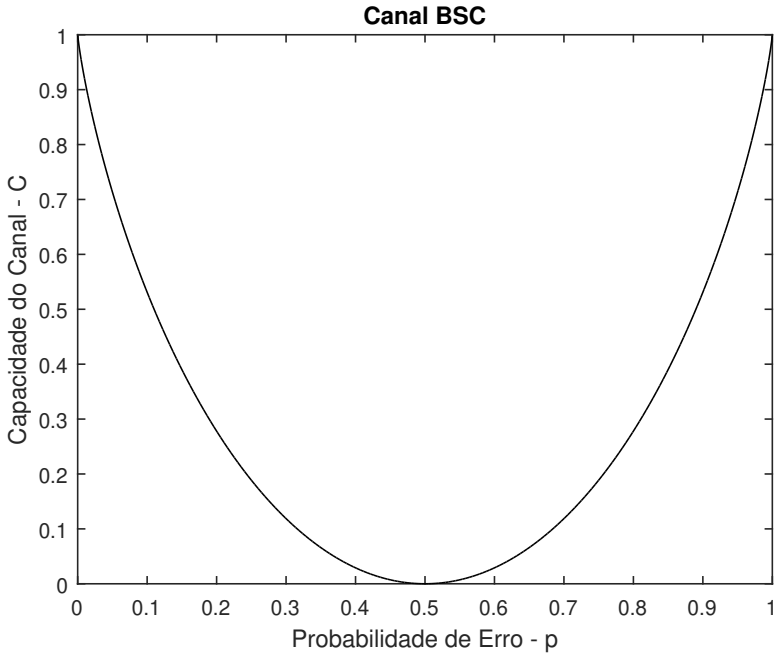


Fonte: Próprio autor.

Sabendo que $P[Y = 1|X = 1] = P[Y = 0|X = 0] = 1-p$, *a capacidade do canal será* $C = 1-H(p)$, *onde* $H(p) = -p\log_2(p) - (1-p)\log_2(1-p)$, *que é a entropia de uma V.A. binária, por exemplo, se*

a probabilidade do canal for $p = 0.5$ então a capacidade do canal será nula, pois haverá incerteza do bit transmitido e qual será recebido, ou seja, toda a informação do canal será perdida. Entretanto, conforme $p \longrightarrow 0$ ou $p \longrightarrow 1$, melhor será o desempenho do canal aproximando-se de um canal ideal, como pode ser observado na Figura 3.

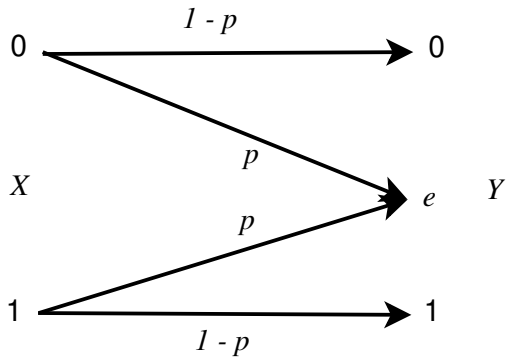
Figura 3 – Capacidade canal binário simétrico.



Fonte: Próprio autor.

Exemplo 2.1.4 *Na Figura 4 temos o Canal Binário com Apagamento (Binary Erasure Channel - BEC). Neste canal têm-se duas possíveis entradas, entretanto, na saída há uma terceira possibilidade, conhecida como apagamento, que representa um bit que foi corrompido na propagação pelo canal. Esta terceira saída é representada por e e a probabilidade de ocorrência é dada por $P[Y = e|X = 1] = P[Y = e|X = 0] = p$.*

Figura 4 – Canal binário com apagamento.



Fonte: Próprio autor.

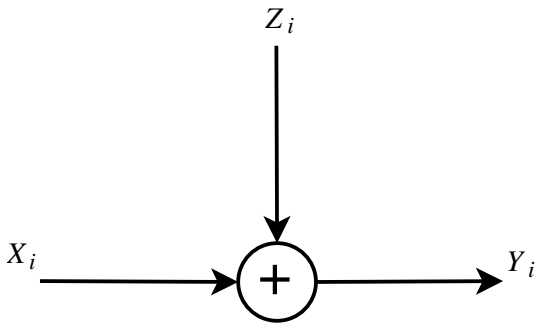
A capacidade desse canal é $C = 1 - p$. Neste canal, ao receber um bit 0 ou 1, tem-se a certeza que os mesmos foram enviados e, caso seja recebido o apagamento, o mesmo pode ser reenviado.

Exemplo 2.1.5 *Na Figura 5 temos o Canal Gaussiano que é um canal de tempo discreto representado por*

$$Y_i = X_i + Z_i, \tag{2.9}$$

onde, a saída do canal Y_i é dada pela soma da entrada X_i com um ruído aditivo gaussiano e branco (Additive White Gaussian Noise - AWGN) Z_i de média 0 e variância $\aleph$, no tempo i. O ruído Z_i é independente da entrada X_i.

Figura 5 – Canal gaussiano.



Fonte: Próprio autor.

Para o cálculo da capacidade, temos alguns casos a considerar:

- *Se a variância do ruído é igual a zero, então a informação de saída será igual a informação da entrada, deste modo, o canal pode transmitir um número real arbitrário sem erros com a capacidade deste canal sendo infinita.*

- *Se a variância do ruído é maior que zero sem restrições de entrada, então a saída pode distinguir os dados recebidos com uma pequena probabilidade de erro, este cenário também possui capacidade infinita.*

- *Se a entrada possui alguma restrição então a capacidade do canal não será infinita. Uma limitação comum em uma transmissão é a potência (P). Podemos assumir que*

$$E[X^2] \leqslant P \quad e \quad \sigma^2 = \aleph, \tag{2.10}$$

onde σ^2 é a variância. E, considerando que o ruído é independente da entrada do canal, temos

$$E[Y^2] = E[X^2] + E[Z^2] \leqslant P + \aleph. \tag{2.11}$$

Assim, através da Equação 2.8 e da Equação 2.11, a capacidade de informação de um canal gaussiano com restrição de potência P, será dada da seguinte forma

$$C = \max_{p(x)] \leqslant P} \frac{1}{2} \log_2 \left(1 + \frac{P}{\aleph}\right), \tag{2.12}$$

onde C é dada em bits por transmissão. Além disso, se o canal for limitado em banda B, a capacidade do canal será da seguinte forma

$$C = B \log_2 \left(1 + \frac{P}{\aleph}\right), \tag{2.13}$$

onde C é dada em bits/s, a banda B do canal em Hz e $P/\aleph$ é a razão sinal-ruído, dada em dB.

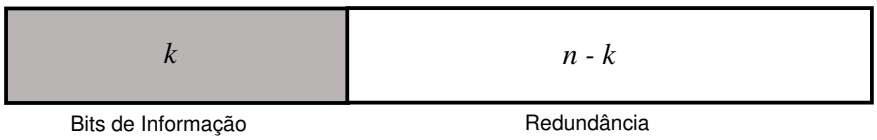
2.2 CODIFICAÇÃO DE CANAL

A codificação de canal é frequentemente usada em sistemas de comunicação digital para tentar proteger a informação do ruído e da interferência e, desse modo, reduzir o número de erros de bits. Esta codificação consiste na inserção de bits de redundância no sinal de informação, e estes bits adicionais visam ajudar na detecção e correção de erros proporcionando uma maior confiabilidade na informação recebida. A codificação do canal é, então, o processo em que uma redundância controlada é adicionada à informação objetivando a detecção e a correção de erros introduzidos pelas características do canal. Existem dois tipos principais de códigos corretores de erros: os códigos de bloco e os códigos convolucionais. Os códigos convolucionais serão apresentados no Capítulo 3. Na Subseção 2.2.1 serão apresentados os códigos de bloco, para que no Capítulo 4 os códigos polares, que formam uma classe especial dos códigos de bloco, sejam melhor desenvolvidos.

2.2.1 Código de Blocos Lineares

Um código de bloco linear é caracterizado pelos parâmetros (n, k), onde n é o comprimento do código e k é sua dimensão. Ele é denominado código de bloco pois durante a codificação, uma sequência de k bits terá adicionada $n - k$ bits de redundância gerando assim, uma nova palavra de n bits, sendo $n > k$, como ilustrado na Figura 6.

Figura 6 – Código de bloco.



Fonte: Próprio autor.

Definição 2.2.1 *Seja F_q um corpo finito de q elementos. Dizemos que $\mathcal{C}$ é um código de bloco linear, de comprimento n e dimensão k sobre F_q se $\mathcal{C}$ for um subespaço vetorial de dimensão k de F_q^n.*

Para valores de $q = 2$ o código é denominado binário e de $q = 3$ esse código é denominado ternário. Para $q = n$, dizemos que o código é q-ário. Além disso, a quantidade de palavras-código de um código $\mathcal{C}(n, k)$ sobre F_q é q^k e a taxa do código é dada pela razão $R = k/n$.

Exemplo 2.2.1 *A partir de F_2^4 considere um código $\mathcal{C}$ de parâmetros (n, k) iguais a $(4, 2)$. Este código possui 16 palavras-código pois $q^k = 2^4 = 16$, e taxa de codificação $R = \frac{2}{4} = \frac{1}{2}$. As palavras-código*

deste código de bloco linear são apresentadas a seguir

$$\mathcal{C} = (0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111).$$

Definição 2.2.2 *Sejam dois vetores,* $\boldsymbol{u} = (u_1, \ldots, u_n)$ *e* $\boldsymbol{v} = (v_1, \ldots, v_n) \in F_q^n$*, a distância de Hamming* $d_h(\boldsymbol{u}, \boldsymbol{v})$ *destes vetores, será a quantidade de coordenadas que estes vetores diferem entre si.*

Exemplo 2.2.2 *Considerando os vetores* $\boldsymbol{u}$ *= (0010),* $\boldsymbol{v}$ *= (0111) e* $\boldsymbol{w}$ *= (1010) em* F_2^4*, a distância de Hamming entre eles será:*

$$\begin{aligned} d_h(0010, 0111) &= 2, \\ d_h(0010, 1010) &= 1, \\ d_h(1010, 0111) &= 3. \end{aligned}$$

Propriedade 2.2.1 *Sejam* $\boldsymbol{u}$ *e* $\boldsymbol{v}$ *dois vetores* $\in F_q^n$*, as seguintes propriedades são válidas:*

1. $d_h(\boldsymbol{u}, \boldsymbol{v}) \geq 0$*;*
2. $d_h(\boldsymbol{u}, \boldsymbol{v}) = d_h(\boldsymbol{v}, \boldsymbol{u})$*;*
3. $d_h(\boldsymbol{u}, \boldsymbol{v}) \leq d_h(\boldsymbol{u}, \boldsymbol{w}) + d_h(\boldsymbol{w}, \boldsymbol{v})$.

Definição 2.2.3 *Seja* $\mathcal{C}$ *um código de bloco linear. A distância mínima de Hamming deste código é definida por*

$$d(C) = \min[d_h(\boldsymbol{u}, \boldsymbol{v})], \tag{2.14}$$

para todo $\boldsymbol{u}, \boldsymbol{v} \in C$ *e* $\boldsymbol{u} \neq \boldsymbol{v}$.

Exemplo 2.2.3 *Considere o código* $\mathcal{C} = \{0000, 1110, 1001, 1011\} \subset F_2^4$*. Calculando-se a distância de Hamming, temos que*

$$\begin{aligned} d_h(0000, 1110) &= 3; \quad d_h(1110, 1001) = 3; \\ d_h(0000, 1001) &= 2; \quad d_h(1110, 1011) = 2; \\ d_h(0000, 1011) &= 3; \quad d_h(1001, 1011) = 1. \end{aligned}$$

Deste modo, a distância mínima de Hamming deste código é $d(C) = 1$.

Definição 2.2.4 *Define-se peso de Hammming* $w_h(\boldsymbol{u})$ *de um vetor* $\boldsymbol{u}$*, como sendo o número de componentes não nulos contidos no vetor. Deste modo, o peso mínimo de Hamming* $w(\mathcal{C})$ *de um código* $\mathcal{C}$ *é aquele que possui o menor peso das palavras-código.*

Exemplo 2.2.4 *Considerando o código do exemplo anterior, o peso das palavras-código de* $\mathcal{C}$ *são:*

$$\begin{aligned} w_h(1110) &= 3, \\ w_h(1001) &= 2, \\ w_h(1011) &= 3. \end{aligned}$$

Portanto, o peso mínimo de $\mathcal{C}$ *é igual a* $w(\mathcal{C}) = 2$.

Teorema 2.2.1 *Seja $\mathcal{C}$ um código e $\boldsymbol{u}$ e $\boldsymbol{v}$ dois vetores pertencentes à $\mathcal{C}$. Tem-se que:*

1. $d_h(\boldsymbol{u}, \boldsymbol{v}) = w_h(\boldsymbol{u} - \boldsymbol{v})$.
2. $d(\mathcal{C}) = w(\mathcal{C})$.

Neste trabalho será utilizada somente a distância mínima de Hamming, assim, para facilitar a notação, iremos omitir o índice h. Em casos com número binários, a Condição 1 do Teorema 2.2.1 equivale a operação mod-2 entre os vetores, para avaliar o peso do vetor resultante, $d(\mathbf{u}, \mathbf{v}) = w(\mathbf{u} \bigoplus \mathbf{v})$.

Exemplo 2.2.5 *Considere dois vetores $\boldsymbol{u} = (10101)$ e $\boldsymbol{v} = (11101)$. A distância de Hamming será*

$$d(\boldsymbol{u}, \boldsymbol{v}) = w(\boldsymbol{u} \bigoplus \boldsymbol{v}) = w(10101 \bigoplus 11101) = w(01000).$$

Deste modo, temos que $d(\boldsymbol{u}, \boldsymbol{v}) = 1$.

Definição 2.2.5 *Um código de bloco $\mathcal{C}$ com parâmentros (n, k) e distância mínima de Hamming d, é representado com parâmetros (n, k, d).*

Teorema 2.2.2 *Seja um código de bloco $\mathcal{C}$ com parâmentros (n, k, d), se a distância mínima de Hamming for correspondente a d, esse código poderá corrigir até $\left\lfloor \frac{d-1}{2} \right\rfloor$ erros e detectar até $(d-1)$ erros.*

Definição 2.2.6 *Seja F_q um corpo finito e $\mathcal{C} \subset F_q^n$ o código de bloco linear com parâmetros (n, k, d) e q^k palavras-código. Se $B_{\mathcal{C}} = \{G_0, G_1, \ldots, G_{k-1}\}$ é uma base de $\mathcal{C}$, então a matriz cujas linhas são os vetores $G_i = (g_{i,0}, \ldots, g_{i,n-1})$, com $i = 1, \ldots, k$, ou seja,*

$$G = \begin{pmatrix} G_0 \\ G_1 \\ \vdots \\ G_{k-1} \end{pmatrix} = \begin{pmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,n-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{0,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1}. \end{pmatrix},$$

é chamada de matriz geradora do código $\mathcal{C}$.

Vale salientar que existe mais de uma base de $\mathcal{C}$, ou seja, a matriz geradora não é única. Deste modo, independente da escolha da base tem-se uma matriz G.

Exemplo 2.2.6 *Considere o código $\mathcal{C} = \{0000, 1110, 1000, 1011\}$. Uma das possibilidades da matriz geradora para $\mathcal{C}$ é dada por*

$$G = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}.$$

Seja $\mathcal{C}$ um código com parâmetros (n, k, d) e matriz geradora G, e seja $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1}) \in F_q^k$ a mensagem a ser codificada. As palavras-código de $\mathcal{C}$ podem ser dadas pela combinação linear das linhas $G_0, G_1, \ldots, G_{k-1}$ de G, com os k *bits* de $\mathbf{u}$, ou seja

$$\begin{aligned} \mathbf{v} &= \mathbf{u} \cdot G && (2.15) \\ &= u_0 \cdot G_0 + u_1 \cdot G_1 + \cdots + u_{k-1} \cdot G_{k-1}. && (2.16) \end{aligned}$$

Exemplo 2.2.7 *Considere $\mathcal{C}$ um código com parâmetros $(5, 3)$ em F_2^5. Inicialmente para construirmos G, é necessário que uma base para $\mathcal{C}$ seja definida. Baseado em k e n, sabe-se que a base $B_{\mathcal{C}}$ deverá possuir 3 vetores com 5 bits. Dentre as 2^5 palavras possíveis, escolhemos 3 vetores que sejam linearmente independentes, portanto,*

$$B_{\mathcal{C}} = \{10111, 01011, 00101\},$$

é uma base para $\mathcal{C}$. Assim, G será dada da seguinte forma

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}. \tag{2.17}$$

As possíveis entradas considerando F_2^3, são

$$\{000, 100, 010, 001, 110, 011, 101, 111\}.$$

Dessa forma, a codificação será dada por (2.15).

$$\begin{array}{ll} (000) \cdot G = (00000); & (100) \cdot G = (10111); \\ (001) \cdot G = (00101); & (101) \cdot G = (10010); \\ (010) \cdot G = (01011); & (110) \cdot G = (11100); \\ (011) \cdot G = (01110); & (111) \cdot G = (11001). \end{array}$$

Logo,

$$\mathcal{C} = \{00000, 00101, 01011, 01110, 10111, 10010, 11100, 11001\}.$$

Para identificar uma mensagem enviada $\mathbf{u}$ a partir de uma mensagem recebida $\mathbf{v}$ é necessário mostrar que existe uma mensagem $\mathbf{u} \in \mathbb{Z}_k^n$, tal que a Equação 2.15 é satisfeita.

Exemplo 2.2.8 *Considere um código $\mathcal{C}$ com parâmetros $(5, 3)$ e matriz geradora dada por (2.17). Suponha que $\boldsymbol{v} = (00101)$ seja a mensagem recebida e queremos identificar a mensagem enviada $\boldsymbol{u} = (u_1 u_2 u_3)$ tal que*

$$(u_1 u_2 u_3) \cdot \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} = (00101).$$

Pela multiplicação, temos um sistema linear com 3 *incógnitas e* 5 *equações*

$$\begin{cases} u_1 = 0; \\ u_2 = 0; \\ u_1 + u_3 = 1; \\ u_1 + u_2 = 0; \\ u_1 + u_2 + u_3 = 1. \end{cases}.$$

Podemos observar que $u_1 = 0$ *e* $u_2 = 0$*. Então*

$$\begin{matrix} u_1 + u_3 = 1; \\ 0 + u_3 = 1. \end{matrix}.$$

Portanto, $u_3 = 1$*, deste modo, a mensagem enviada foi* $\boldsymbol{u} = (001)$.

Este método, não é a melhor maneira de decodificar um canal, uma alternativa mais eficiente é utilizar G na sua forma padrão, de modo que apenas observando o início das palavras é possível realizar a decodificação da mensagem.

Definição 2.2.7 *Uma matriz geradora* G *de um código* $\mathcal{C}$ *está na forma padrão (ou sistemática) se*

$$G = (I_k|\mathcal{P}), \tag{2.18}$$

em que I_k *é a matriz identidade de ordem* k *e* $\mathcal{P}$ *é uma matriz de ordem* $k \times (n-k)$.

Exemplo 2.2.9 *Considere um código* $\mathcal{C}$ *com parâmetros* $(5, 3)$ *e matriz geradora* G *dada por (2.17). Para definir a matriz geradora em sua forma padrão é necessário realizar algumas operações elementares e permutações em suas colunas, de modo a resultar na matriz em sua forma sistemática. Realizando tais operações, a matriz resultante* G' *na forma sistemática será*

$$G' = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

Desta forma, as três primeiras colunas são correspondentes à I_3 *e as duas últimas à* $\mathcal{P}$.

Definição 2.2.8 *Considere* $\mathcal{C}$ *um código de parâmetros* (n, k)*. Se* $G = (I_k|\mathcal{P})$ *é uma matriz geradora de* $\mathcal{C}$ *na forma padrão, então* $H = (-\mathcal{P}^t|I_{n-k})$ *é chamada de matriz controle de paridade de* $\mathcal{C}$*, onde* $\mathcal{P}^t$ *é a matriz transposta de* $\mathcal{P}$ *de ordem* $(n-k) \times k$ *e,* I_{n-k} *a matriz identidade de ordem* $n-k$.

Esta matriz H é utilizada para verificar se uma mensagem pertence ou não ao código. Isto é realizado pela multiplicação da matriz H pela mensagem recebida, se a mesma for igual a 0, então é palavra código, caso contrário não é palavra código. Como podemos observar a seguir

$$G \cdot H^t = (I_k|\mathcal{P}) \cdot (-\mathcal{P}^t|I_{n-k}) = -\mathcal{P} + \mathcal{P} = 0,$$

se $\mathbf{v} \in \mathcal{C}$, então,

$$\mathbf{v} = \mathbf{u} \cdot G \longrightarrow \mathbf{v} \cdot H^t = \mathbf{u} \cdot (G \cdot H^t) = 0,$$

deste modo, **v** é uma palavra código.

Exemplo 2.2.10 *Seja $\mathcal{C}$ um código com parâmetros $(5, 3)$ e matriz geradora G dada por*

$$G = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

Queremos verificar se as mensagens recebidas, $\boldsymbol{v}_1 = (10111)$ e $\boldsymbol{v}_2 = (00010)$ são palavras código. Observa-se que a matriz G encontra-se na sua forma padrão, então podemos determinar a matriz H, de modo que

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

Vamos verificar se $\boldsymbol{v}_1 = (10111)$ é uma palavra código de $\mathcal{C}$. Multiplicando v_1 por H^t

$$\boldsymbol{v}_1 \cdot H^t = (10111) \cdot \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = (00).$$

Como a multiplicação resultou em um vetor nulo, então $\boldsymbol{v}_1 \in \mathcal{C}$. Agora, vamos verificar se $\boldsymbol{v}_2 = (00010)$ é uma palavra código de $\mathcal{C}$. Multiplicando $\boldsymbol{v}_2$ por H^t

$$\boldsymbol{v}_2 \cdot H^t = (00010) \cdot \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = (10).$$

Como a multiplicação não resultou em um vetor nulo, então $\boldsymbol{v}_2 \notin \mathcal{C}$.

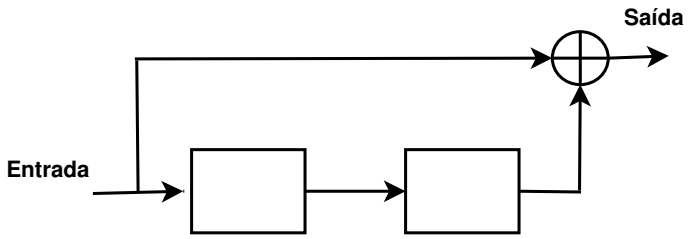
3 CÓDIGOS CONVOLUCIONAIS

Neste capítulo, abordaremos os códigos convolucionais, que são códigos corretores de erros amplamente utilizados pela sua baixa complexidade de implementação tanto do codificador quanto decodificador e, pela fácil adaptação da taxa de codificação. Na Seção 3.1 serão apresentados os métodos de codificação deste código e na Seção 3.2 será apresentada a decodificação baseada no algoritmo de Viterbi. As definições apresentadas neste capítulo podem ser encontradas em [FORNEY 1970, OMURA 1969, ISCAN, LENTNER e XU 2016, TAHIR, SCHWARZ e RUPP 2017, SYBIS et al. 2016].

3.1 CODIFICAÇÃO

Um código convolucional utiliza de memórias finitas no codificador para gerar uma redundância, [FORNEY 1970]. Este código é definido através dos parâmetros (n, k, M), onde n é o número de bits produzidos na saída, k o número de bits inseridos na entrada e M a quantidade de memórias no codificador. Um código com estes parâmetros pode ser caracterizado por uma máquina de estados. Na Figura 7 podemos observar a máquina de estados de um código convolucional com parâmtros $(1, 1, 2)$, onde cada retângulo representa uma memória do codificador e este código possui apenas uma entrada e uma saída.

Figura 7 – Máquina de estados (1,1,2).



Fonte: Próprio autor.

Um esquema simplificado para codificação convolucional pode ser visto na Figura 8.

Figura 8 – Esquema simplificado da codificação convolucional.

Fonte: Próprio autor.

Se a sequência de entrada k de informação no codificador a cada iteração t_n é dada por

$$\mathbf{u} = (u_0^1 u_0^2 \dots u_0^k u_1^1 u_1^2 \dots u_1^k \dots u_{t_n}^1 u_{t_n}^2 \dots u_{t_n}^k), \tag{3.1}$$

e a sequência codificada n na saída a cada iteração t_n é

$$\mathbf{v} = (v_0^1 v_0^2 \dots v_0^n v_1^1 v_1^2 \dots v_1^n \dots v_{t_n}^1 v_{t_n}^2 \dots v_{t_n}^k), \tag{3.2}$$

onde a entrada **u** é demultiplexada em *k* entradas e as *n* saídas são multiplexadas em **v**, então existe uma sequência geradadora para as memórias do codificador dada por

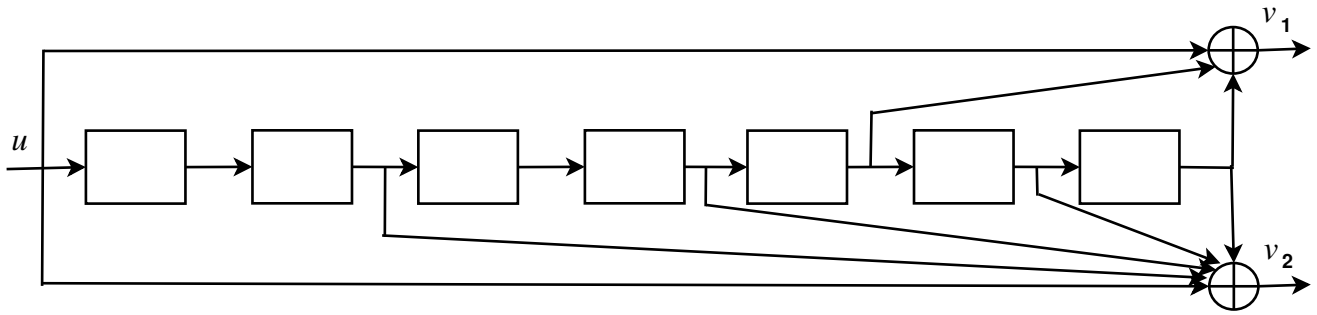
$$\mathbf{g}^i = (g_0^i g_1^i \dots g_M^i), \tag{3.3}$$

onde $i = 1, \dots, n$.

Estas sequências geradoras são também chamadas de resposta ao impulso e podem ser determinadas aplicando 1's e 0's para cada caminho em cada memória do codificador, onde 1 significa conexão e 0 não conexão.

Exemplo 3.1.1 *Considere o código convolucional (2,1,7) com máquina de estados como da Figura 9.*

Figura 9 – Máquina de estados código convolucional (2,1,7).



Fonte: Próprio autor.

Neste caso, os geradores ou resposta ao impulso deste código serão dados por

$$\boldsymbol{g}_1(10000101),$$
$$\boldsymbol{g}_2(10101011).$$

A codificação de um código convolucional pode ser realizada de várias formas, dentre elas: polinômio gerador, matriz geradora, convolução, diagrama de estados ou diagrama de treliça. A seguir, apresentaremos tais formas.

- **Polinômio Gerador**

  Seja

  $$\mathbf{g}^j = (g_0^j g_1^j ... g_m^j), \tag{3.4}$$

  com $j = 1, \dots, n$, uma sequencia geradora de um código convolucional (n, k, M), onde o correspondente polinômio gerador para cada *j* é

  $$g^j(D) = g_0^j + g_1^j D + g_2^j D^2 + \dots + g_M^j D^M, \tag{3.5}$$

  onde *D* é a variável de atraso unitário. Se **u** é uma mensagem a ser codificada e *u(D)* o polinômio que representa **u**, então as sequências codificadas serão dadas por

$$
\begin{aligned}
v^1(D) &= u(D)g^1(D), \\
v^2(D) &= u(D)g^2(D), \\
&\vdots \\
v^n(D) &= u(D)g^n(D),
\end{aligned}
$$

que serão multiplexadas na palavra-código $\mathbf{v} = (v^1 v^2 \ldots v^n)$.

Exemplo 3.1.2 *Considerando o código convolucional (2,1,7) com máquina de estados como da Figura 9, vamos codificar a mensagem* $\boldsymbol{u} = (1010011)$. *Representando as sequências geradoras e a mensagem em polinômios geradores temos que*

$$
\begin{aligned}
g^1(D) &= 1 + D^5 + D^7, \\
g^2(D) &= 1 + D^2 + D^4 + D^6 + D^7, \\
u(D) &= 1 + D^2 + D^5 + D^6.
\end{aligned}
$$

Deste modo para $v^1(D)$

$$
\begin{aligned}
v^1(D) &= g^1(D)u(D) \\
&= (1 + D^5 + D^7)(1 + D^2 + D^5 + D^6) \\
&= 1 + D^2 + D^6 + D^9 + D^{10} + D^{11} + D^{12} + D^{13}.
\end{aligned}
$$

Portanto, $\boldsymbol{v}_1 = (10100010011111)$. *Do mesmo modo para* $v^2(D)$

$$
\begin{aligned}
v^2(D) &= g^2(D)u(D) \\
&= (1 + D^2 + D^4 + D^6 + D^7)(1 + D^2 + D^5 + D^6) \\
&= 1 + D^5 + D^6 + D^{10} + D^{11} + D^{13}.
\end{aligned}
$$

E, assim, $\boldsymbol{v}_2 = (10000110001101)$. *Logo, multiplexando,*

$$
\boldsymbol{v} = (\boldsymbol{v}_1 \boldsymbol{v}_2) = (11001000000111000010111111011).
$$

- **Matriz Geradora**

Para este método de codificação, uma matriz geradora é utilizada, esta matriz é construída a partir das sequências geradoras $\mathbf{g}^j = (g_0^j g_1^j \ldots g_M^j)$, com $j = 1, \ldots, n$, de um código convolucional (n, k, M). Cada linha da matriz geradora é formada por uma réplica da sequência geradora e cada nova linha é deslocada uma posição para a direita e bits zero são adicionados nas lacunas.

Assim, a matriz geradora tem a seguinte forma

$$G = \begin{pmatrix} G_0 & G_1 & \dots & G_M & 0 & 0 & 0 \\ 0 & G_0 & G_1 & \dots & G_M & 0 & 0 \\ 0 & 0 & G_0 & G_1 & \dots & G_M & 0 \\ 0 & 0 & 0 & G_0 & G_1 & \ddots & \ddots \end{pmatrix}, \tag{3.6}$$

onde a dimensão da matriz depende do comprimento da mensagem. Por exemplo, para um código convolucional $(2, 1, 2)$, temos que os polinômios geradores tem a seguinte forma

$$\mathbf{g}^1 = (g_0^1 g_1^1 g_2^1), \tag{3.7}$$

e,

$$\mathbf{g}^2 = (g_0^2 g_1^2 g_2^2). \tag{3.8}$$

Assim, a matriz geradora será dada por

$$G = \begin{pmatrix} g_0^1 g_0^2 & g_1^1 g_1^2 & g_2^1 g_2^2 & 0 & 0 & 0 \\ 0 & g_0^1 g_0^2 & g_1^1 g_1^2 & g_2^1 g_2^2 & 0 & 0 \\ 0 & 0 & g_0^1 g_0^2 & g_1^1 g_1^2 & g_2^1 g_2^2 & 0 \\ 0 & 0 & 0 & g_0^1 g_0^2 & g_1^1 g_1^2 & \ddots \end{pmatrix}. \tag{3.9}$$

Casos onde existe mais de uma saída, G_i será a concatenação de g_i^j, para cada $i = 1, \dots, M$ e $j = 1, \dots, n$. A codificação, também pode ser realizada utilizando a Equação 2.15.

Exemplo 3.1.3 *Considerando o código convolucional (2,1,7) com máquina de estados como da Figura 9, vamos codificar a mensagem* $\boldsymbol{u} = (1010011)$. *Sabendo que as sequências geradoras são dados por*

$$\boldsymbol{g}_1 = (10000101) \quad e \quad \boldsymbol{g}_2 = (10101011).$$

A matriz G terá a forma

$$G = \begin{pmatrix} 11 & 00 & 01 & 00 & 01 & 10 & 01 & 11 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 11 & 00 & 01 & 00 & 01 & 10 & 01 & 11 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 11 & 00 & 01 & 00 & 01 & 10 & 01 & 11 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 11 & 00 & 01 & 00 & 01 & 10 & 01 & 11 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 11 & 00 & 01 & 00 & 01 & 10 & 01 & 11 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 11 & 00 & 01 & 00 & 01 & 10 & 01 & 11 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 11 & 00 & 01 & 00 & 01 & 10 & 01 & 11 \end{pmatrix}.$$

Para codificar pela matriz geradora basta efetuar a multiplicação da mensagem pela matriz

geradora, como em (2.15), da seguinte forma

$$\boldsymbol{v} = \boldsymbol{u}G = (1010011)\begin{pmatrix} 11 & 00 & 01 & 00 & 01 & 10 & 01 & 11 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 11 & 00 & 01 & 00 & 01 & 10 & 01 & 11 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 11 & 00 & 01 & 00 & 01 & 10 & 01 & 11 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 11 & 00 & 01 & 00 & 01 & 10 & 01 & 11 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 11 & 00 & 01 & 00 & 01 & 10 & 01 & 11 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 11 & 00 & 01 & 00 & 01 & 10 & 01 & 11 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 11 & 00 & 01 & 00 & 01 & 10 & 01 & 11 \end{pmatrix}.$$

Logo, $\boldsymbol{v} = (11001000000111000010111111011)$

- **Convolução**

A codificação por convolução de dois sinais **u** e **g** é dada por

$$\mathbf{v}(i) = (\mathbf{u} * \mathbf{g})(i) = \sum_{j=0}^{i} u(j)g(j-i). \tag{3.10}$$

Exemplo 3.1.4 *Considerando o código convolucional (2,1,7) com máquina de estados como da Figura 9, vamos codificar a mensagem* $\boldsymbol{u} = (1010011)$*, sabendo que as geradoras são dados por*

$$\boldsymbol{g}_1 = (10000101) \quad e \quad \boldsymbol{g}_2 = (10101011).$$

Para $\boldsymbol{v}^1 = \boldsymbol{u} * \boldsymbol{g}^1$*, temos*

$$\boldsymbol{v}^1 = (1010011) * (10000101),$$

que pode ser calculada fazendo

$$\left|\begin{array}{cccccccccccccccccccccccc}
 & & & & & & & 1 & 0 & 1 & 0 & 0 & 1 & 1 & & & & & & & & & & \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & & & & & & & & & & & & & & & & \\
 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & & & & & & & & & & & & & & & \\
 & & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & & & & & & & & & & & & & & \\
 & & & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & & & & & & & & & & & & & \\
 & & & & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & & & & & & & & & & & & \\
 & & & & & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & & & & & & & & & & & \\
 & & & & & & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & & & & & & & & & & \\
 & & & & & & & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & & & & & & & & & \\
 & & & & & & & & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & & & & & & & & \\
 & & & & & & & & & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & & & & & & & \\
 & & & & & & & & & & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & & & & & & \\
 & & & & & & & & & & & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & & & & & \\
 & & & & & & & & & & & & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & & & & \\
 & & & & & & & & & & & & & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & & & \\
 & & & & & & & & & & & & & & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & & \\
 & & & & & & & & & & & & & & & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & \\
 & & & & & & & & & & & & & & & & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{array}\right|.$$

Portanto, $\boldsymbol{v}_1 = (10100010011111)$. *Do mesmo modo para* $\boldsymbol{v}^2 = \boldsymbol{u} * \boldsymbol{g}^2$, *tem-se que*

$$\boldsymbol{v}^2 = (1010011) * (10101011) = (10000110001101).$$

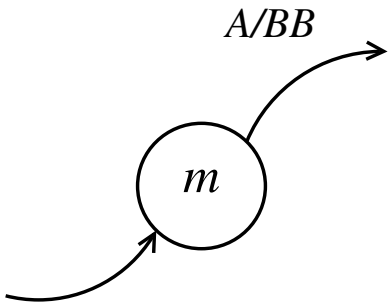
Logo,

$$\boldsymbol{v} = (\boldsymbol{v}_1\boldsymbol{v}_2) = (1100100000011100001011111011).$$

- **Diagrama de Estados**

  As operações realizadas por uma máquina de estados podem ser descritas por um diagrama de estados. Em um diagrama de estados podemos observar todos os estados da máquina do código convolucional em questão, os quais podem ser representados por circulos, e as possíveis transições, representadas por setas, tal que *A/BB*, onde *A* representa o bit atual que está sendo inserido na máquina e *BB* representa a saída do codificador, como pode ser observado na Figura 10.

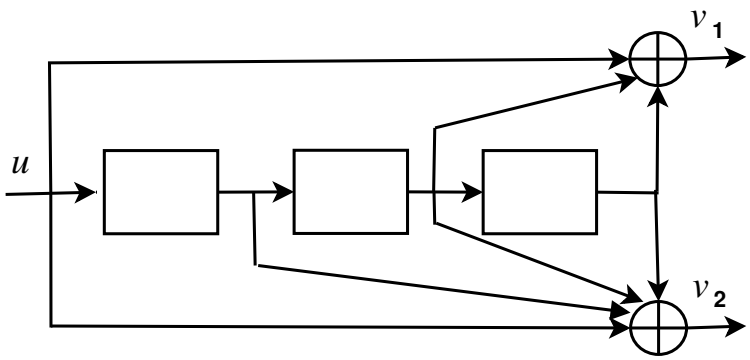
Figura 10 – Representação diagrama de estados.



Fonte: Próprio autor.

Exemplo 3.1.5 *Considere o código convolucional* $(2, 1, 3)$ *com máquina de estados como o da Figura 11.*

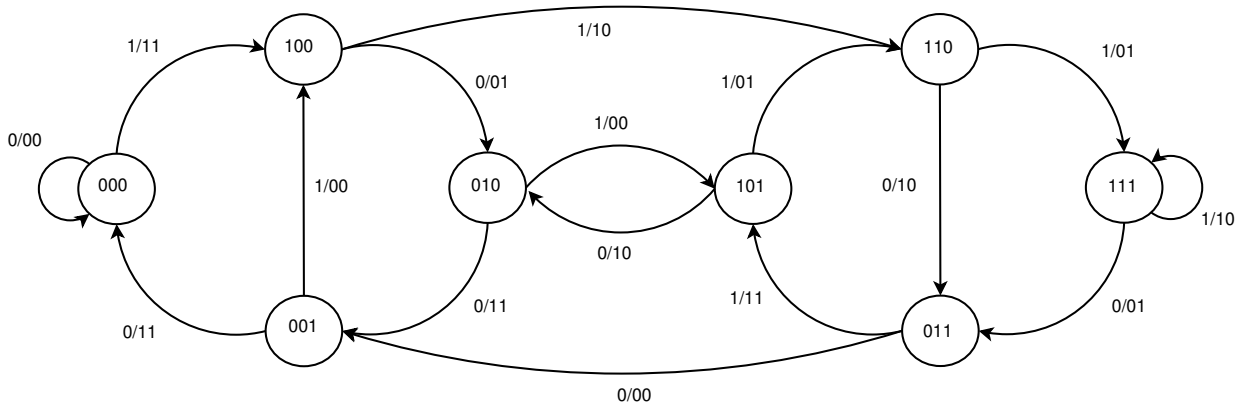
Figura 11 – Máquina de estados código convolucional (2,1,3).



Fonte: Próprio autor.

Podemos representar seu diagrama de estados como na Figura 12.

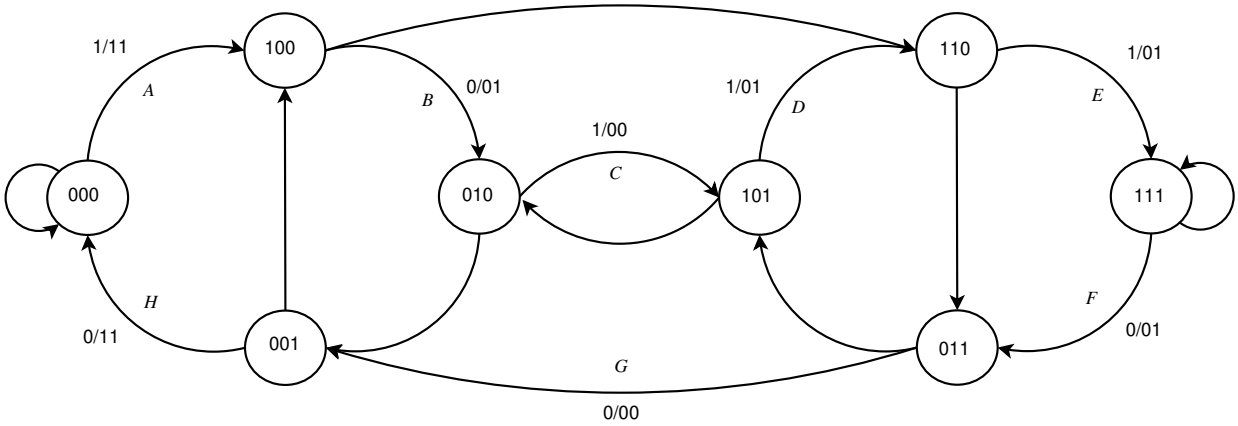
Figura 12 – Diagrama de estados código convolucional (2,1,3).



Fonte: Próprio autor.

Para codificar a mensagem $\boldsymbol{u} = (10111)$*, devemos partir do estado inicial da máquina, ou seja, com memória 000, e realizar as transições com a inserção de cada bit. Podemos observar as transições realizadas pela Figura 13 , onde a sequência de saltos realizados é dado por A, B, C, D, E, F, G e H, respectivamente. Vale ressaltar, que na codificação de um código convolucional é necessário sempre limpar a memória da máquina, ou seja, voltar ao estado inicial da máquina, para isto, são inseridos* M *bits zero, após toda a mensagem ser inserida.*

Figura 13 – Codificação da mensagem **u** = (10111).



Fonte: Próprio autor.

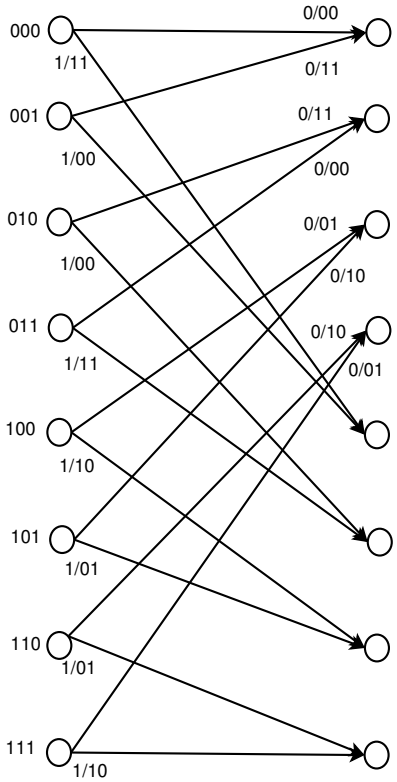
Deste modo, temos que $\boldsymbol{v} = (1101000101010011)$.

- **Diagrama de Treliça**

O diagrama de treliça permite observar, para cada instante t_n, todas as transições possíveis de estados. Cada uma das colunas deste diagrama representa todos os estados da máquina e suas conexões. Este diagrama pode ser utilizado tanto na codificação, quanto na decodificação. Para um código (n, k, M), se o tamanho da mensagem a ser codificada é k, então serão utilizadas $(k + M)$ cópias da treliça, pois cada cópia representa um bit de entrada e como as memórias do codificador devem sempre voltar para seu estado inicial, serão adicionadas M cópias da treliça para limpar as memórias.

Exemplo 3.1.6 *Considerando o código do Exemplo 3.1.5 e partindo da máquina de estados da Figura 11, pode-se representar seu diagrama de treliça como na Figura 14.*

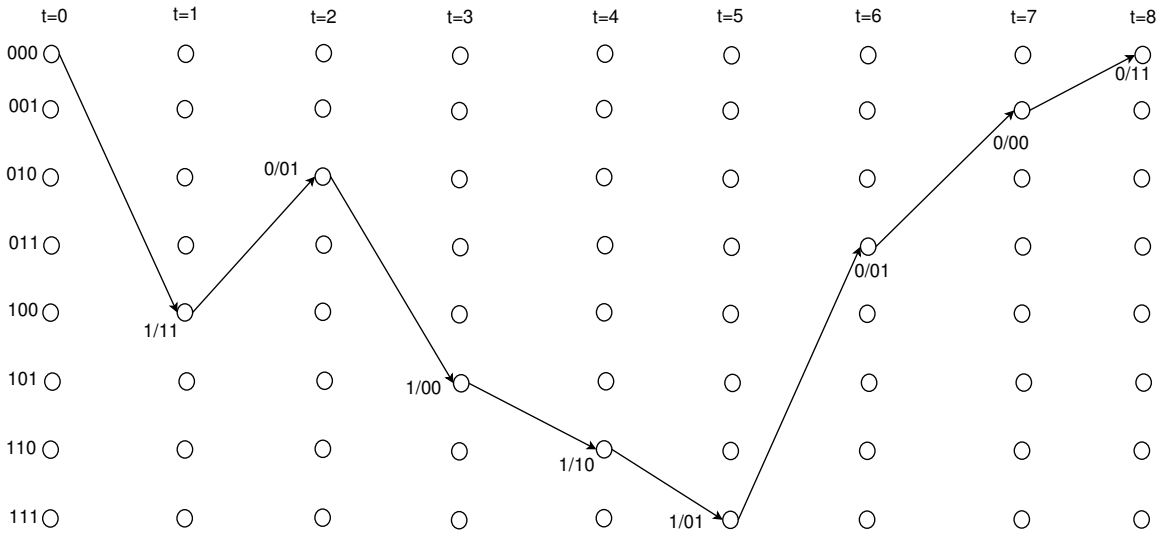
Figura 14 – Diagrama de treliça código convolucional (2,1,3).



Fonte: Próprio autor.

Vamos codificar a mensagem $\boldsymbol{u} = (10111)$ *utilizando o diagrama de treliça. Para isso, devemos partir do estado inicial da máquina, ou seja, com memória 000, e realizar as transições com a inserção de cada bit. Podemos observar as transições realizadas pela Figura 15, onde após inserirmos os bits da mensagem, foram inseridos 3 bits 0 para limpar a memória do codificador.*

Figura 15 – Codificação da mensagem u = (10111).



Fonte: Próprio autor.

Deste modo, temos que $\boldsymbol{v} = (1101000101010011)$.

3.2 DECODIFICAÇÃO

A decodificação mais utilizada para códigos convolucionais, é a baseada no algoritmo de Viterbi, que é uma técnica de decodificação por máxima verossimilhança (*Maximum Likelihood Decoding*). O funcionamento da decodificação por máxima verossimilhança baseia-se em comparar a soma das distâncias de Hamming, entre todos os dados da sequência de bits recebidos no receptor e todos os dados de cada uma das possíveis sequências que podem ser geradas no codificador convolucional do transmissor. A menor soma destas distâncias indica a possível sequência de dados que provavelmente teria sido gerada pelo codificador convolucional do transmissor.

Seja **m** uma mensagem e **c** uma palavra-código correspondente obtida através de um codificador que será transmitida por um canal discreto sem memória. A mensagem recebida **r** pode ser diferente de **c** devido a ruídos inseridos no canal. A partir de **r** o decodificador gera uma estimativa $\hat{\mathbf{c}}$ que é equivalente a uma estimativa da mensagem $\hat{\mathbf{m}}$. Idealmente $\mathbf{c} = \hat{\mathbf{c}}$ e $\mathbf{m} = \hat{\mathbf{m}}$, caso contrário ocorreu um erro de decodificação. Para que a decodificação seja ótima a probabilidade de erro deve ser minimizada e isto ocorre quando a função de verossimilhança é maximizada. Se $p(\mathbf{r}|\mathbf{c})$ é a probabilidade de receber uma mensagem **r** dado que **c** foi enviado, deste modo a função de verossimilhança será dada por $\log(p(\mathbf{r}|\mathbf{c}))$.

Seja um canal BSC, com duas possíveis entradas (1 ou 0), que são igualmente afetadas por erro, com probabilidade *p*, temos que

$$p(0|0) = 1 - p = p(1|1) \quad \text{e} \quad p(0|1) = p = p(1|0).$$

Se uma palavra-código **c** é transmitida por este canal e **r** é a mensagem recebida, ambas com mesmo comprimento *n*, então

$$p(\mathbf{r}|\mathbf{c}) = \prod_{i=1}^{n} p(ri|ci). \tag{3.11}$$

A função de verossimilhança será

$$\log p(\mathbf{r}|\mathbf{c}) = \log\left(\prod_{i=1}^{n} p(ri|ci)\right) = \sum_{i=1}^{n} \log(p(ri|ci)). \tag{3.12}$$

Sabendo que

$$p(ri|ci) = \begin{cases} p, & se \quad r_i \neq c_i \\ 1-p, & se \quad r_i = c_i \end{cases}, \tag{3.13}$$

que a distância de Hamming é dada por

$$d = d_h(\mathbf{c}, \mathbf{r}) = \#\{i \mid r_i \neq c_i\}, \tag{3.14}$$

onde # representa cardinalidade, e supondo que **r** difere de **c** em d posições, então

$$\log(p(\mathbf{r}|\mathbf{c})) = d\log(p) + (n-d)\log(1-p) = d\log\left(\frac{p}{1-p}\right) + n\log(1-p). \tag{3.15}$$

Podemos observar que $n\log(1-p)$ é constante, pois só depende da probabilidade de erro e do tamanho da palavra recebida. Como usualmente $p < 1/2$, então $\log\left(\frac{p}{1-p}\right) < 0$. Assim, um decodificador de máxima verossimilhança é aquele que a estimativa $\hat{\mathbf{c}}$ é escolhida com base na menor distância de Hamming entre o vetor recebido **r** e a palavra-código **c** transmitida.

Exemplo 3.2.1 *Considere* $\boldsymbol{c}_1 = (0000110111)$ *e* $\boldsymbol{c}_2 = (1101000111)$ *duas palavras-código de um código convolucional de parâmetro* $(2, 1, 2)$. *Supondo que a palavra recebida pelo decodificador é* $\boldsymbol{r} = (1001110111)$, *vamos determinar a estimativa* $\hat{\boldsymbol{c}}$ *no receptor por máxima verossimilhança. Como o decodificador por máxima verosimilhança determina a palavra-código pela menor distância de Hamming entre a palavra recebida e a possível palavra-código, temos que a distância de Hamming para* $\boldsymbol{c}_1$ *e* $\boldsymbol{c}_2$ *são dadas por*

$$d_h(\boldsymbol{r}, \boldsymbol{c}_1) = 2,$$
$$d_h(\boldsymbol{r}, \boldsymbol{c}_2) = 3.$$

Logo, por máxima verossimilhança fazemos $\hat{\boldsymbol{c}} = \boldsymbol{c}_1$

3.2.1 Algoritmo de Viterbi

O decodificador convolucional de Viterbi tem seu funcionamento baseado no conceito do diagrama de treliça, onde todas as possíveis transições entre todos os possíveis estados do codificador convolucional, localizado no transmissor, podem ser representados ao longo do tempo.

Para estimar $\hat{\mathbf{c}}$ a partir de uma mensagem **r** recebida por um codificador convolucional (n, k, M), as seguintes etapas são realizadas na decodificação:

1. Partindo do estado inicial da treliça, compara-se a saída de cada possível transição com os bits recebidos, calculando a distância de Hamming.

2. Soma-se a distância de Hamming entre os bits recebidos e as possíveis saídas e cada transição.

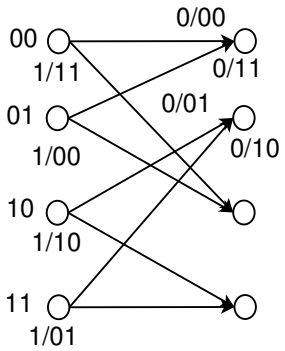
3. Esta operação é realizada até dois caminhos convergirem para um mesmo estado. O caminho que irá continuar é aquele com a menor distância de Hamming acumulada até o momento.

4. Estas operações são realizadas até terminarem os dados recebidos. Vários caminhos podem chegar ao final, mas o escolhido será aquele com a menor distância acumulada.

Assim, considerando todos os possíveis caminhos percorridos por **r**, a mensagem $\hat{\mathbf{c}}$ mais provável será aquela com a menor distância de **r**.

Exemplo 3.2.2 *Considere o código convolucional (2,1,2) com diagrama de treliça como da Figura 16. Vamos decodificar a mensagem* $\boldsymbol{r} = (1001110111)$ *utilizando o algortimo de Viterbi.*

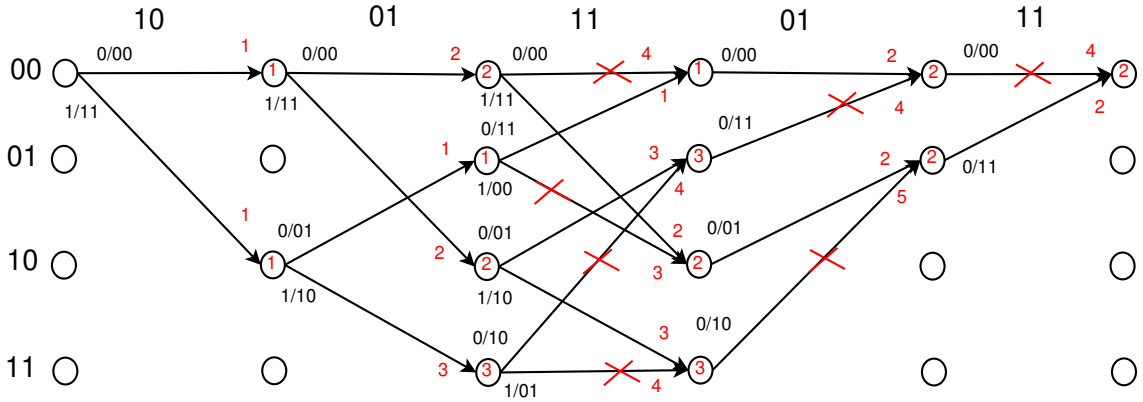
Figura 16 – Diagrama de treliça código convolucional (2,1,2).



Fonte: Próprio autor.

A Figura 17 foi construída seguindo os passos realizados pelo algoritmo de Viterbi.

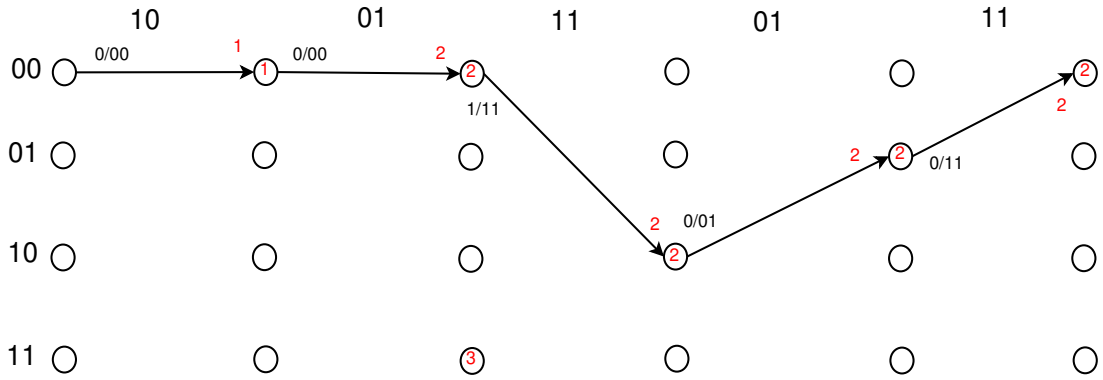
Figura 17 – Decodificação utilizando o algoritmo de Viterbi.



Fonte: Próprio autor.

Podemos observar que o caminho com a menor distância de Hamming possui um valor de $d = 2$*, e este será o caminho para o qual o decodificador irá escolher a palavra-código* $\hat{\boldsymbol{c}}$*. Na Figura 18 pode-se observar o caminho escolhido pelo decodificador.*

Figura 18 – Caminho escolhido pela decodificação utilizando o algoritmo de Viterbi.



Fonte: Próprio autor.

Assim, temos que $\hat{\boldsymbol{c}} = (0000110111)$ *e* $\hat{\boldsymbol{m}} = (001)$.

No Capítulo 5 será realizada uma implementação computacioanal dos códigos convolucionais, a máquina de estados da Figura 9 será utilizada para implementação, a codificação será baseada no diagrama de treliça e a decodificação no algoritmo de Viterbi. Esta implementação tem como objetivo observar o desempenho deste código na detecção e correção de erros, assim como sua latência e *throughput*.

4 CÓDIGOS POLARES

Neste capítulo, apresentaremos os códigos polares, que são códigos de bloco introduzidos por Arikan, [ARIKAN 2009]. Estes códigos são construídos a partir de uma estratégia proposta também por Arikan chamada de polarização de canal, onde cópias de um canal discreto sem memória são combinadas de modo que, após divididas, podem ser separados em canais bons e ruins. Deste modo, bits de informação são enviados nos canais confiáveis (bons) e bits congelados nos canais mais ruidosos (ruins).

Na Seção 4.1 será apresentada a estratégia de polarização de canal que auxilia na codificação deste código, na Seção 4.2 serão apresentados os métodos de codificação e na Seção 4.3 a decodificação dos códigos polares. As definições apresentadas neste capítulo podem ser encontradas em [ARIKAN 2009, TAHIR, SCHWARZ e RUPP 2017, OLIVEIRA e LAMARE 2017, ISCAN, LENTNER e XU 2016, SYBIS et al. 2016, HUIGOL 2017, NIU et al. 2014, SARKIS e GROSS 2013a, SARKIS e GROSS 2013b, LEROUX et al. 2011].

4.1 POLARIZAÇÃO DE CANAL

Seja um canal sem memória $W : X \to Y$, com probabilidade de transição

$$W(Y|x) = P(Y = y|X = x). \tag{4.1}$$

Vamos considerar *N* cópias do canal *W*, representado por

$$W^N : X^N \to Y^N, \tag{4.2}$$

onde $X^N = (X_1, X_2, ..., X_N)$ e $Y^N = (Y_1, Y_2, ..., Y_N)$, com probabilidade de transição dada por

$$W^N(\mathbf{y}_1^N|\mathbf{x}_1^N) = \prod_{i=1}^{N} W(y_i|x_i), \tag{4.3}$$

onde $\mathbf{x}_1^N = (x_1, x_2, ..., x_N)$, e $\mathbf{y}_1^N = (y_1, y_2, ..., y_N)$. A capacidade do canal W, que iremos denotar *I(W)*, representa como visto na Definição 2.1.6, a maior taxa que a comunicação alcança com *W* e pode ser calculada por

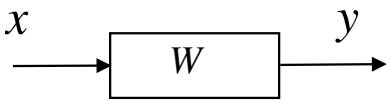
$$I(W) = \sum_{y \in Y} \sum_{x \in X} \frac{1}{2} W(y|x) \log \left(\frac{W(y|x)}{\frac{1}{2}W(y|0) + \frac{1}{2}W(y|1)} \right). \tag{4.4}$$

Agora, iremos realizar a combinação de *N* canais, de modo que, $W_N : X^N \to Y^N$, onde, $N = 2^n$, $n \geq 0$ com $n \in \mathcal{Z}^+$.

- **Canal N = 1**

Inicialmente para $N = 1$, teremos somente a cópia de um único canal, que pode ser representado pela Figura 19.

Figura 19 – Canal sem memória.



Fonte: Próprio autor.

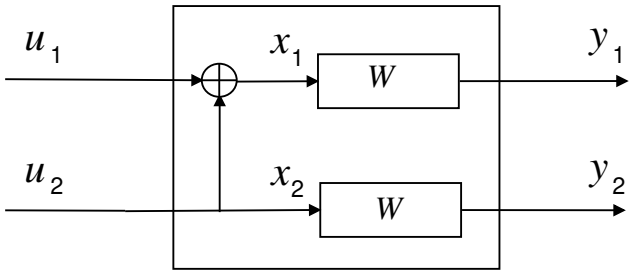
- **Canal N = 2**

Para $N = 2$, teremos a combinação de dois canais *W*,

$$W_2 : X^2 \to Y^2, \tag{4.5}$$

nesta combinação na entrada dos canais teremos x_1 e x_2 e na saída y_1 e y_2, como representado na Figura 20

Figura 20 – Canal para N = 2.



Fonte: Próprio autor.

Podemos observar que diferente do canal W_1, no canal W_2 temos uma operação de soma antes da entrada x_1. Deste modo, mapeando as entradas x_1 e x_2 a partir de u_1 e u_2, temos que, x_1 será a soma entre as entradas u_1 e u_2 e x_2 será a própria entrada u_2. Assim, as probabilidades de transição do canal W_2 serão obtidas por essas combinações

$$W_2(y_1, y_2|u_1, u_2) = W(y_1|u_1 \oplus u_2)W(y_2|u_2). \tag{4.6}$$

A partir disso conseguimos mapear as entradas $\mathbf{x} = (x_1, x_2)$ a partir de $\mathbf{u} = (u_1, u_2)$ e uma matriz geradora dada por essas combinações. Deste modo

$$\mathbf{x}_1^2 = \mathbf{u}_1^2 G_2, \tag{4.7}$$

onde G_2 é a matriz geradora dada a partir de W_2.

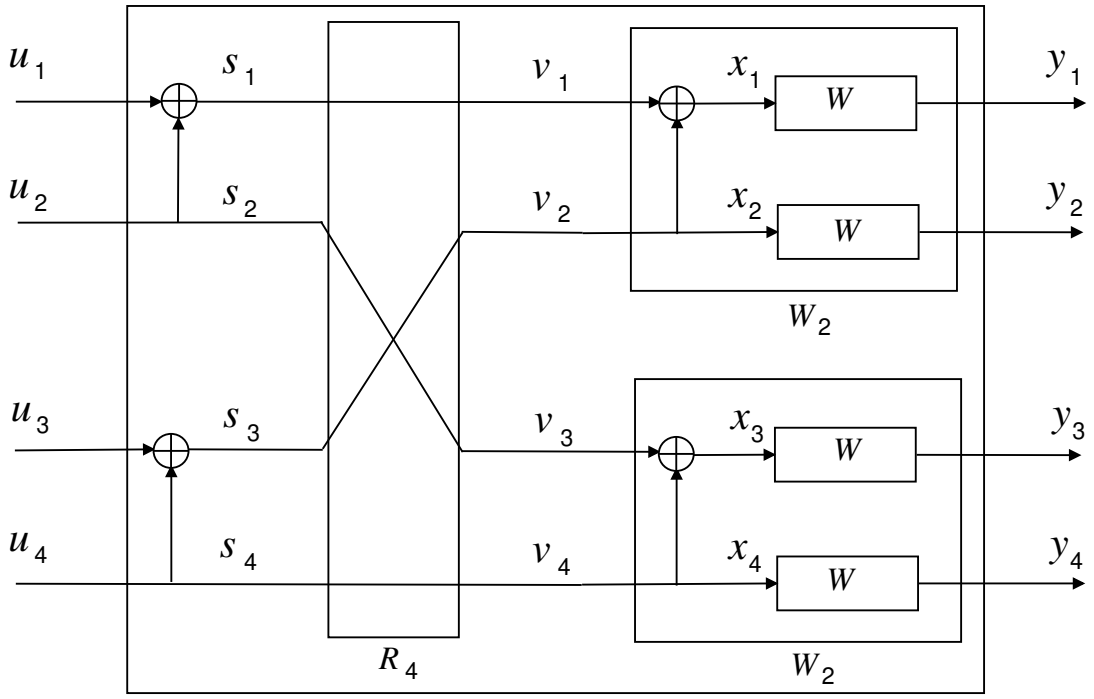
$$G_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}. \tag{4.8}$$

- **Canal N = 4**

Para $N = 4$, teremos a combinação de quatro canais *W*, ou a combinação de duas cópias de W_2 como representado pela Figura 21. Para esta construção temos 4 entradas **x** e 4 saídas **y**, de modo que

$$W_4 : X^4 \to Y^4. \tag{4.9}$$

Figura 21 – Canal para N = 4.



Fonte: Próprio autor.

Entretanto para $N = 4$, será necessário fazer uma operação de permutação para a combinação dos canais, como mostrado na Figura 21. Está permutação é representada por R_n e irá separar as entradas de índices ímpares das pares. O vetor de entrada $\mathbf{u}_1^4 = (u_1, u_2, u_3, u_4)$ em W_4, é transformado primeiramente em $\mathbf{s}_1^4 = (s_1, s_2, s_3, s_4)$ de tal forma que

$$s_{2i-1} = u_{2i-1} \oplus u_{2i} \text{ e } s_{2i} = u_{2i}, \text{ para } 1 < i < 4.$$

Após, o operador R_4 age sobre o vetor $\mathbf{s}_1^4$, de modo

$$\mathbf{s}_1^4 = (s_1, s_2, s_3, s_4) \to \mathbf{v}_1^4 = (s_1, s_3, s_2, s_4), \tag{4.10}$$

que também pode ser representado por

$$R_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.11}$$

As probabilidades de transição do canal serão obtidas da seguinte forma

$$W_4(\mathbf{y}_1^4|\mathbf{u}_1^4) = W_2(\mathbf{y}_1^2|u_1 \oplus u_2, u_3 \oplus u_4)W_2(\mathbf{y}_3^4|u_2, u_4). \tag{4.12}$$

Deste modo temos que

$$\mathbf{x}_1^4 = \mathbf{u}_1^4 G_4, \tag{4.13}$$

onde G_4 é dada pela seguinte matriz

$$G_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}. \tag{4.14}$$

- **Canal N = 8**

Para $N = 8$, teremos a combinação de oito canais *W*, ou a combinação de duas cópias de W_4 como representado pela Figura 22. Para esta construção temos 8 entradas **x** e 8 saídas **y**, de modo que

$$W_8 : X^8 \to Y^8. \tag{4.15}$$

Assim como no caso $N = 4$, será necessário fazer uma operação de permutação para a combinação dos canais, como representado na Figura 22. O vetor de entrada $\mathbf{u}_1^8 = (u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8)$ é transformado primeiramente em $\mathbf{s}_1^8 = (s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8)$ de tal forma que

$$s_{2i-1} = u_{2i-1} \oplus u_{2i} \text{ e } s_{2i} = u_{2i}, \text{ para } 1 < i < 8.$$

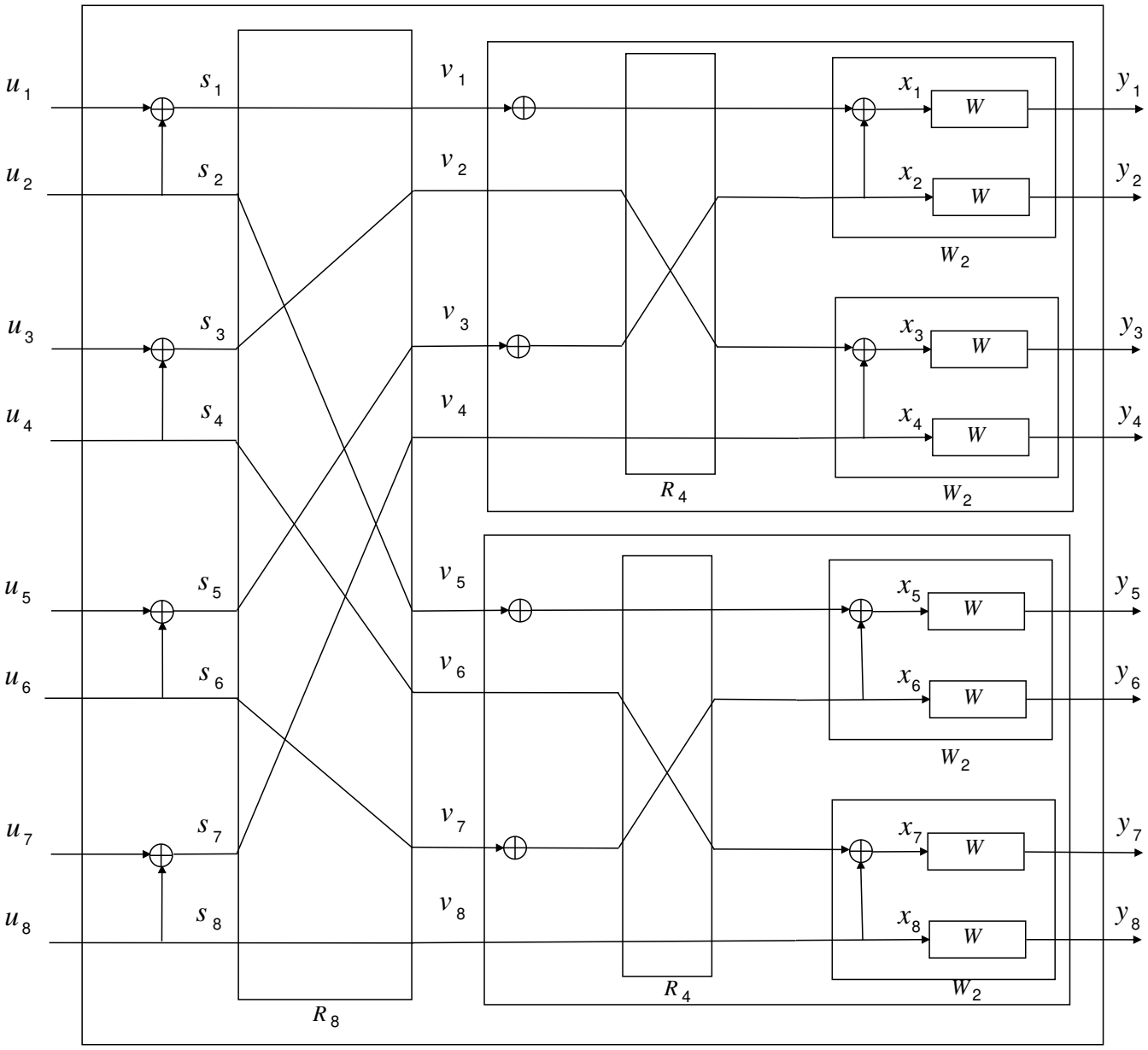
Após, o operador R_8 age sobre o vetor $\mathbf{s}_1^8$, de modo

$$\mathbf{s}_1^8 = (s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8) \to \mathbf{v}_1^8 = (s_1, s_3, s_5, s_7, s_2, s_4, s_6, s_8). \tag{4.16}$$

A matriz permutação R_8 será representada da seguinte forma:

$$R_8 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.17}$$

Figura 22 – Canal para N = 8.



Fonte: Próprio autor.

As probabilidades de transição do canal serão obtidas da seguinte forma

$$W_8(\mathbf{y}_1^8|\mathbf{u}_1^8) = W(\mathbf{y}_1^4|u_1 \oplus u_2, u_3 \oplus u_4, u_5 \oplus u_6, u_7 \oplus u_8)W(\mathbf{y}_5^8|u_{1,e}^8). \tag{4.18}$$

Deste modo temos que

$$\mathbf{x}_1^8 = \mathbf{u}_1^8 G_8, \tag{4.19}$$

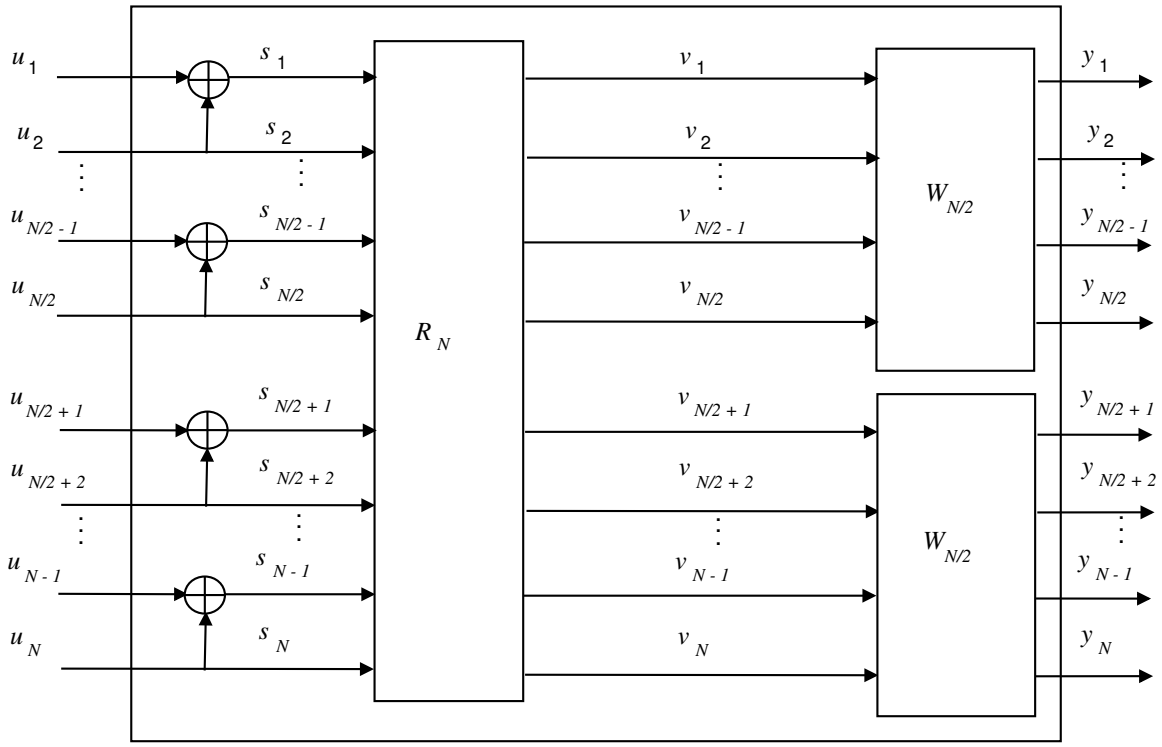
onde G_8 é dada pela seguinte matriz

$$G_8 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}. \tag{4.20}$$

- **Caso Geral -** W_N

  Para a construção do canal geral, temos N vetores de entrada $\mathbf{x}_1^N$ e N vetores de saída $\mathbf{y}_1^N$. A Figura 23 representa este canal. Como podemos observar, existe um bloco representado pela matriz de permutação R_N e depois duas cópias independentes de $W_{N/2}$.

Figura 23 – Canal geral W_N.



Fonte: Próprio autor.

A matriz de permutação R_N faz com que as entradas ímpares sejam as entradas da primeira cópia de $W_{N/2}$, enquanto que, as entradas pares sejam as entradas da segunda cópia de $W_{N/2}$. Para o vetor de entrada $\mathbf{u}_1^N$ em W_N, ele é transformado primeiramente em s_1^N de tal forma que

$$s_{2i-1} = u_{2i-1} \oplus u_{2i} \text{ e } s_{2i} = u_{2i}, \text{ para } 1 < i < N/2.$$

Assim, o operador R_N age na entrada $\mathbf{s}_1^N$ produzindo $\mathbf{v}_1^N = (s_1 s_3, ..., s_{N-1}, s_2 s_4, ..., s_N)$ que se

tornarão as entradas para as duas cópias de $W_{N/2}$. Utilizando R_N pode-se mapear os vetores de entrada $\mathbf{x}_1^N$ a partir dos vetores $\mathbf{u}_1^N$ a partir de uma matriz geradora G_N da seguinte forma

$$\mathbf{x}_1^N = \mathbf{u}_1^N G_N. \tag{4.21}$$

A divisão de canal tem como objetivo separar os canais W_N em N canais W por grau de polarização, a fim de observar quais canais possuem maior ou menor confiabilidade. Deste modo

$$W_N^i : X \to Y^N \times X^{i-1}, \tag{4.22}$$

onde, $1 \leq i \leq N$. As probabilidades de transição podem ser definidas por

$$W_N^i(\mathbf{y}_1^N, \mathbf{u}_1^{i-1}|u_i) = \sum_{u_{i+1}^N \in X^{N-1}} \frac{1}{2^{N-1}} W_N(\mathbf{y}_1^N|\mathbf{u}_1^N), \tag{4.23}$$

onde, u_i representa a entrada e $(\mathbf{y}_1^N, \mathbf{u}_1^{i-1})$ a saída de W_N^i.

Calcular as probabilidades de transição (4.23) de modo a obter a capacidade de canal para um canal qualquer não é simples. Porém, considerando um canal BEC com probabilidade de apagamento $e = 1/2$, em [ARIKAN 2009] foi apresentada uma fórmula fechada para o cálculo da capacidade para cada N. Neste caso, a capacidade $I(W_N)$ pode ser calculada de maneira recursiva por

$$I(W_N^{2i-1}) = I(W_{N/2})^2 \tag{4.24}$$

e

$$I(W_N^{2i}) = 2I(W_{N/2}) - I(W_{N/2})^2, \tag{4.25}$$

com $I(W_1) = 1 - e$. Por facilidade de notação os canais bons serão representados por W^+ e os ruins por W^-.

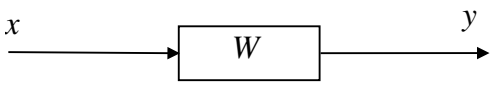
Exemplo 4.1.1 *Considerando canais BEC com $e = 1/2$, vamos calcular as capacidades destes canais utilizando a polarização de canal para diferentes valores de N.*

- *N = 1*

  Temos neste caso que

$$I(W) = 1 - e = 1 - 0.5 = 0.5.$$

Figura 24 – $I(W_1)$.

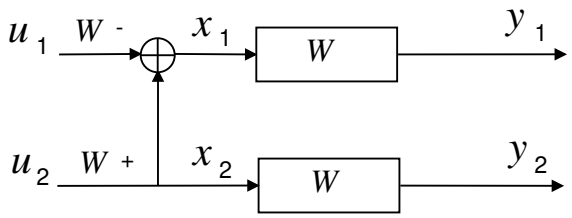


Fonte: Próprio autor.

- *N = 2*

  A capacidade total será dividida em dois canais, W^- e W^+, como pode ser observado na Figura 25.

Figura 25 – $I(W_2)$.



Fonte: Próprio autor.

Aplicando as Equações (4.24) e (4.25), temos

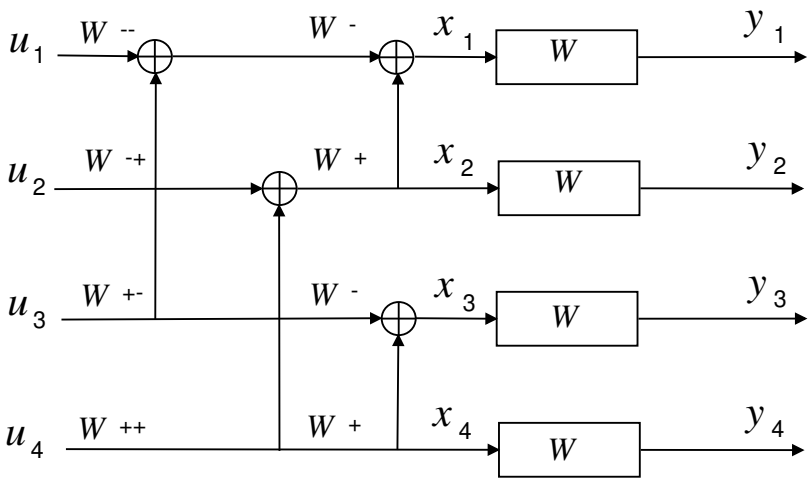
$$\begin{aligned} I(W^-) &= I(W)^2 = (0.5)^2 = 0.25, \\ I(W^+) &= 2I(W) - I(W)^2 = 1 - (0.5)^2 = 0.75. \end{aligned}$$

A maior capacidade representa aquele canal que possui uma confiabilidade maior, e é dado por aquele canal que possui mais combinações, no caso, u_1.

- *N = 4*

  A capacidade total será novamente dividida, entretanto agora entre 4 canais, como na Figura 26.

Figura 26 – $I(W_4)$.



Fonte: Próprio autor.

Os canais serão calculados da seguinte forma

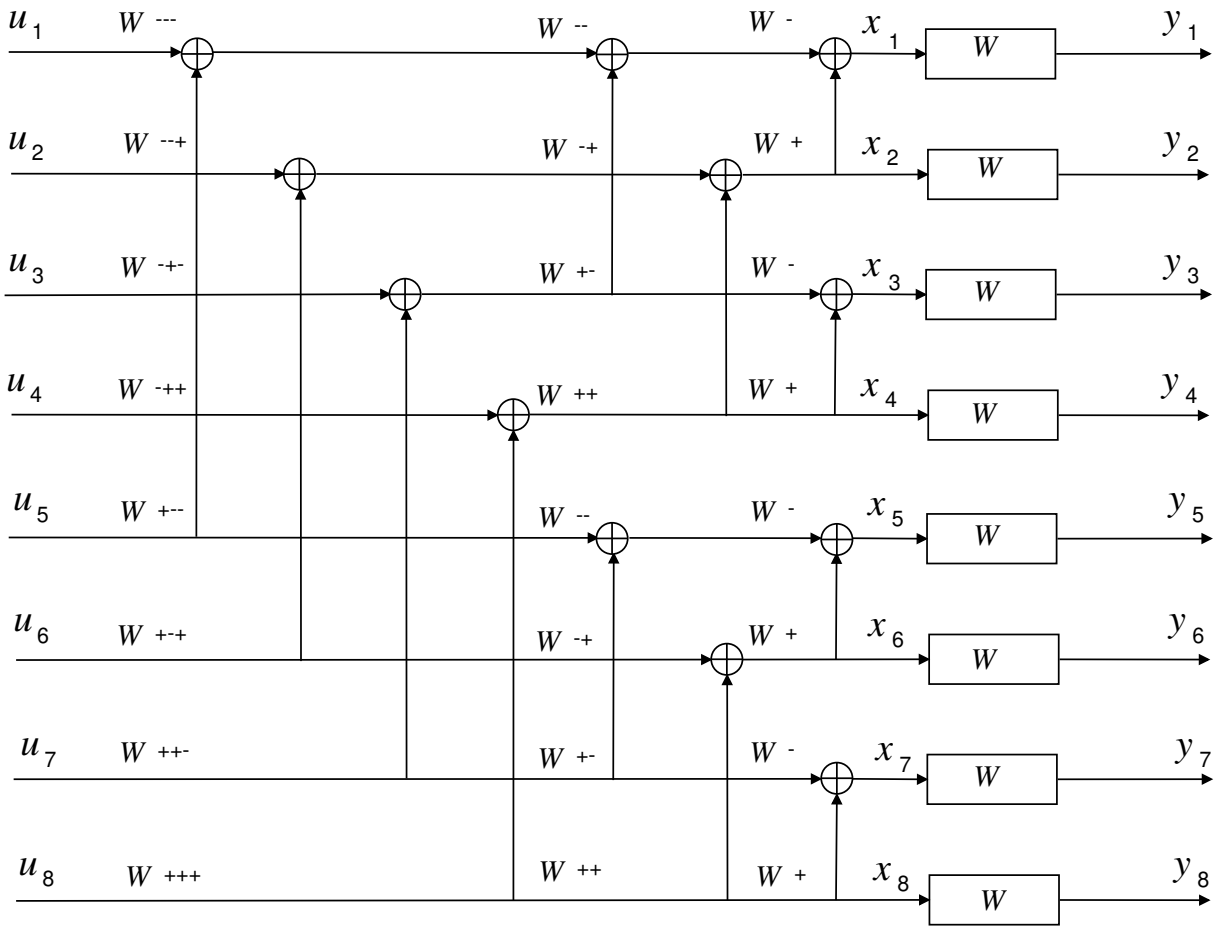
$$
\begin{aligned}
I(W^{--}) &= I(W^{-})^2 = (0.25)^2 = 0.0625 \\
I(W^{-+}) &= 2I(W^{-}) - I(W^{-})^2 = 2(0.25) - (0.25)^2 = 0.4375 \\
I(W^{+-}) &= I(W^{+})^2 = (0.75)^2 = 0.5625 \\
I(W^{++}) &= 2I(W^{+}) - I(W^{+})^2 = 2(0.75) - (0.75)^2 = 0.9375.
\end{aligned}
$$

Neste caso podemos observar a elevada diferença de confiabilidade entre canais, poderíamos, por exemplo, congelar a entrada u_1 e transmitir as informações pelas entradas u_2, u_3 e u_4.

- *N = 8*

  Neste caso a capacidade será novamente dividida entre 8 canais, como na Figura 27.

Figura 27 – $I(W_8)$.



Fonte: Próprio autor.

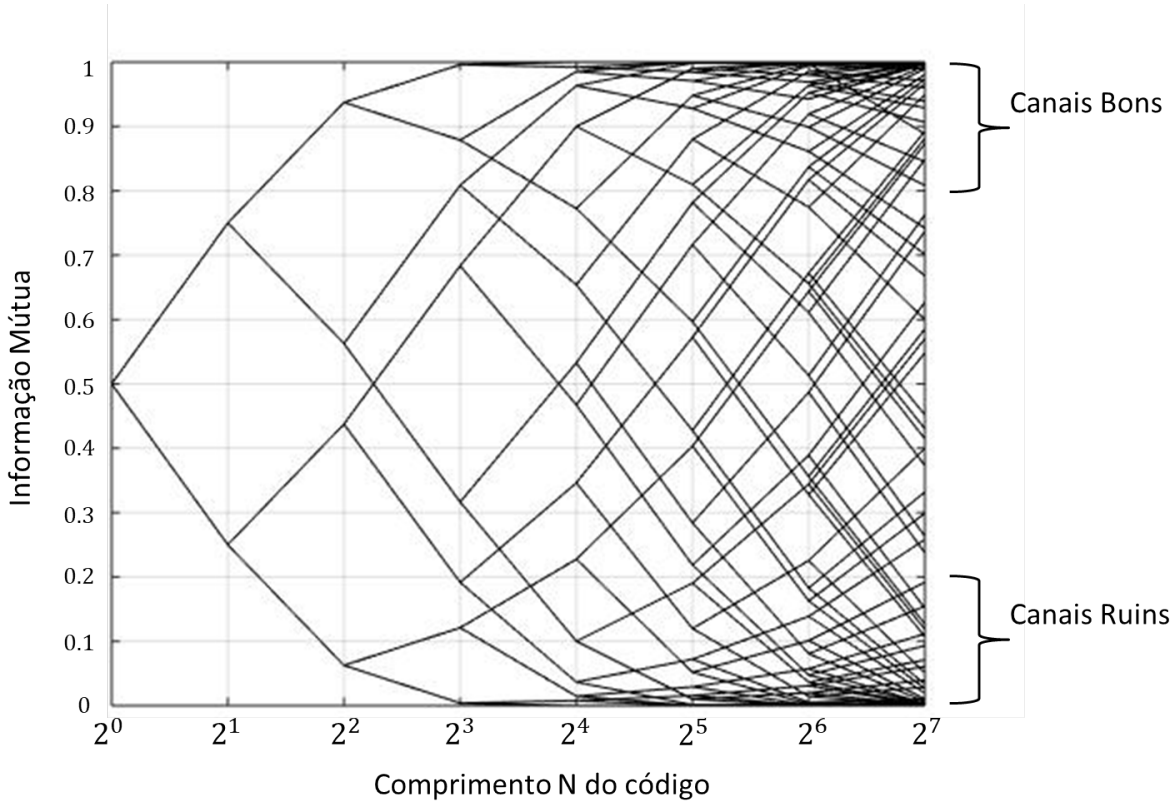
Analogamente aos casos anteriores, as capacidades serão

$$
\begin{aligned}
I(W^{---}) &= I(W^{--})^2 = (0.0625)^2 = 0.0039 \\
I(W^{--+}) &= 2I(W^{--}) - I(W^{--})^2 = 2(0.0625) - (0.0625)^2 = 0.1210 \\
I(W^{-+-}) &= I(W^{-+})^2 = (0.4375)^2 = 0.1914 \\
I(W^{-++}) &= 2I(W^{-+}) - I(W^{-+})^2 = 2(0.4375) - (0.4375)^2 = 0.6835 \\
I(W^{+--}) &= I(W^{+-})^2 = (0.5625)^2 = 0.3164 \\
I(W^{+-+}) &= 2I(W^{+-}) - I(W^{+-})^2 = 2(0.5625) - (0.5625)^2 = 0.8085 \\
I(W^{++-}) &= I(W^{++})^2 = (0.9375)^2 = 0.8789 \\
I(W^{+++}) &= 2I(W^{++}) - I(W^{++})^2 = 2(0.9375) - (0.9375)^2 = 0.9960.
\end{aligned}
$$

Neste caso, para uma transmissão de 4 bits de informação poderíramos congelar as entradas u_1, u_2, u_3 e u_5 e transmitir as informações pelas entradas u_4, u_6, u_7 e u_8

Os cálculos para a confiabilidade dos canais podem ser realiados para $N = 2^n$, e quanto maior o valor de n, ou seja, com $n \to \infty$, os canais polarizados vão para os extremos 0 e 1. Podemos observar pela Figura 28 uma simulação que realizamos para vários valores de N. Nela os canais bons são aqueles que possuem mais combinações entre os canais e maior informação mútua, já os canais ruins possuem menos combinações e uma informação mútua menor.

Figura 28 – Canais polarizados.



Fonte: Próprio autor.

4.2 CODIFICAÇÃO

Vimos na Seção 4.1 que para $N = 2^n$ com $n \geq 0$, a codificação de canal de um vetor de entrada $\mathbf{u}_1^N$ em $\mathbf{x}_1^N$ pode ser realizada a partir de uma matriz geradora G_N como em (4.21). A partir da polarização de canal podemos decidir quais serão os canais de informação e quais serão os canais congelados. Com esta informação, separamos as linhas da matriz G_N em dois conjuntos, o conjunto A representando as linhas da matriz relacionadas com os bits de informação e A^c representando as linhas relacionadas com os bits congelados. Do mesmo modo, a divisão do vetor $\mathbf{u}_1^N$ é realizada, divindo $\mathbf{u}_1^N$ em $\mathbf{u}_A$ que indica o vetor com os bits de informação e u_{A^c} que indica o vetor com os bits congelados. Assim, a codificação será dada por

$$\mathbf{x}_1^N = \mathbf{u}_A G_N(A) \bigoplus \mathbf{u}_{A^C} G_N(A^c), \tag{4.26}$$

onde $G_N(A)$ é a submatriz formada pelas linhas da matriz G_N com índices A e $G_N(A^c)$ é a submatriz formada pelas linhas da matriz G_N com índices A^c.

Deste modo, definimos um código polar como um código de bloco representado pelos parâmetros (N, k, A, μ_{A^C}), onde N é o comprimento da palavra-código $\mathbf{x}_1^N$ produzida, k a dimensão do código que especifica o tamanho de A, A são as linhas de G_N que representam os bits de informação que serão transmitidos e μ_{A^C} são os bits dos canais congelados.

Exemplo 4.2.1 *Considere o código polar com parâmetros (4, 2, {1,3}, (1,0)). Utilizando a matriz G_4 dada em (4.14), a codificação será realizada da seguinte forma*

$$\boldsymbol{x}_1^4 = (\boldsymbol{u}_1 \boldsymbol{u}_3) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \bigoplus (10) \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

Sabendo que ($\boldsymbol{u}_1$, $\boldsymbol{u}_3$) são entradas binárias, existem 4 combinações possíveis

$$(0,0), (0,1), (1,0), (1,1).$$

Sendo assim, substitui-se cada uma delas em ($\boldsymbol{u}_1$, $\boldsymbol{u}_3$) e encontramos os seguintes valores para $\boldsymbol{x}_1^4$

$$\begin{aligned}
(0,0) \rightarrow \boldsymbol{x}_1^4 &= (0,0,0,0) \bigoplus (1,0,1,0) = (1,0,1,0); \\
(0,1) \rightarrow \boldsymbol{x}_1^4 &= (1,1,0,0) \bigoplus (1,0,1,0) = (0,1,1,0); \\
(1,0) \rightarrow \boldsymbol{x}_1^4 &= (1,0,0,0) \bigoplus (1,0,1,0) = (0,0,1,0); \\
(1,1) \rightarrow \boldsymbol{x}_1^4 &= (0,1,0,0) \bigoplus (1,0,1,0) = (1,1,1,0).
\end{aligned}$$

Veremos a seguir que a matriz geradora G_N pode ser determinada utilizando expressões algébricas, que são baseadas no produto de Kronecker de matrizes. O produto de Kronecker, representado por $\otimes$, é uma operação realizada entre duas matrizes, *A'* com dimensão m x n e *B'* com dimensão p x q, da seguinte forma

$$A' \otimes B' = \begin{bmatrix} a'_{11}B' & \dots & a'_{1n}B' \\ \vdots & \ddots & \vdots \\ a'_{m1}B' & \dots & a'_{mn}B' \end{bmatrix}. \tag{4.27}$$

Ou seja

$$A' \otimes B' = \begin{bmatrix} a'_{11}B'_{11} & a'_{11}B'_{12} & \dots & a'_{1n}B'_{11} & a'_{1n}B'_{12} \\ a'_{11}B'_{21} & a'_{11}B'_{22} & \dots & a'_{1n}B'_{21} & a'_{1n}B'_{22} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a'_{m1}B'_{11} & a'_{m1}B'_{12} & \dots & a'_{mn}B'_{11} & a'_{mn}B'_{12} \\ a'_{m1}B'_{21} & a'_{m1}B'_{22} & \dots & a'_{mn}B'_{21} & a'_{mn}B'_{22} \end{bmatrix}. \tag{4.28}$$

Exemplo 4.2.2 *Considere a matriz*

$$F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}. \tag{4.29}$$

Vamos determinar o produto de Kronecker $F^{\otimes n}$, onde $F^{\otimes n}$ = $F \otimes F^{\otimes(n-1)}$, para $n = 1, 2, 3$ e considerando que $F^{\otimes 0} = 1$.

- *Para $n = 1$ temos que o produto de Kronecker é dado da seguinte forma*

$$F^{\otimes 1} = F \otimes F^{\otimes 0}.$$

  Assim,

$$F^{\otimes 1} = F \otimes [1] = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}.$$

- *Para n = 2 temos que*

$$F^{\otimes 2} = F \otimes F^{\otimes 1} = \begin{bmatrix} 1 * \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} & 0 * \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \\ 1 * \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} & 1 * \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

- *Para n = 3 temos*

$$F^{\otimes 3} = F \otimes F^{\otimes 2} = \begin{bmatrix} 1 * \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} & 0 * \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \\ 1 * \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} & 1 * \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \end{bmatrix}$$

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

Agora, veremos como obter G_N utilizando o produto de Kronecker de matrizes. Seja o canal W_2 apresentado na Figura 20 com matriz geradora G_2 dada por (4.8). Por facilidade de notação iremos considerar

$$F = G_2. \tag{4.30}$$

Assim, para o canal W_4 apresentando na Figura 21, a matriz geradora G_4 dada em (4.14), pode ser calculada por

$$G_4 = (I_2 \otimes F)R_4(I_2 \otimes F), \tag{4.31}$$

onde I_2 é a matriz identidade de ordem 2 e R_4 a matriz permutação dada em (4.11). Para o canal W_8 apresentando na Figura 22 com matriz G_8 dada em (4.20), temos que

$$G_8 = (I_4 \otimes F)R_8(I_2 \otimes G_4), \tag{4.32}$$

onde I_4 é a matriz identidade de ordem 4 e R_8 é a matriz de permutação dada em (4.17). Procedendo de modo análogo, a matriz geradoras G_N para um canal W_N dado em Figura 23, pode ser obtida através da relação

$$G_N = (I_{N/2} \otimes F)R_N(I_2 \otimes G_{N/2}). \tag{4.33}$$

A combinação de canal pode ser alterada, de modo a obter uma simplificação na expressão da matriz geradora. Algebricamente, esta simplificação significa realizar as seguintes manipulações em (4.33)

$$\begin{aligned} G_N &= (I_{N/2} \otimes F)R_N(I_2 \otimes G_{N/2}) \\ &= R_N(F \otimes I_{N/2})(I_2 \otimes G_{N/2}) \\ &= R_N(F \otimes G_{N/2}). \end{aligned}$$

Substituindo $G_{N/2} = R_{N/2}(F \otimes G_{N/4})$ temos

$$\begin{aligned} G_N &= R_N(F \otimes (R_{N/2}(F \otimes G_{N/4}))) \\ &= R_N(I_2 \otimes R_{N/2})(F^{\otimes 2} \otimes G_{N/4}). \end{aligned}$$

Substituindo $G_{N/4}$ e sucessivamente até chegar em $G_2 = F$ teremos

$$G_N = B_N F^{\otimes n}, \tag{4.34}$$

onde

$$B_N = R_N(I_2 \otimes B_{N/2}) = R_N(I_2 \otimes R_{N/2})(I_4 \otimes R_{N/4}) \ldots (I_{N/2} \otimes R_2).$$

Enquanto a matriz de permutação R_N opera como um operador de embaralhamento que mapeia as entradas do canal, a matriz de permutação B_N opera como um operador inversor de bits nas posições dos vetores de entrada. Conforme N aumenta, B_N fará cada vez mais trocas nas posições dos vetores.

Exemplo 4.2.3 *Para $N = 4$, temos que $B_4 = R_4(I_2 \otimes R_2)$, ou seja, $B_4 = R_4$ dada em (4.11). Para $N = 8$ temos*

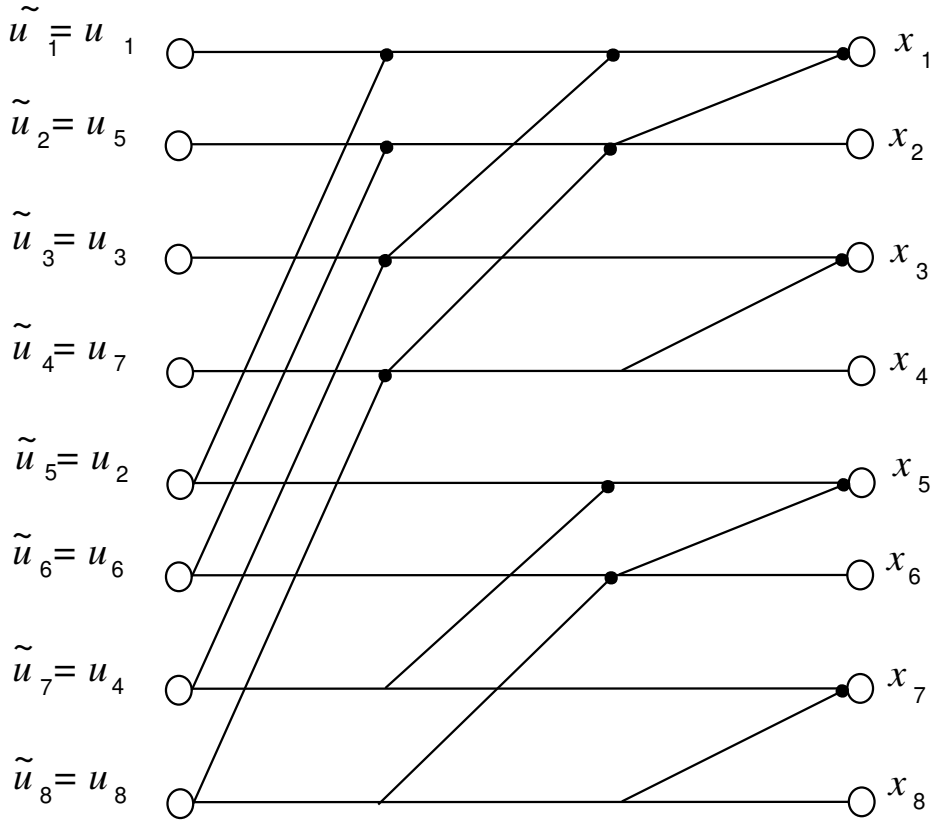
$$\begin{aligned}
B_8 &= R_8(I_2 \otimes B_4) \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.
\end{aligned}$$

Podemos observar pelo exemplo que B_8 realiza a inversão de bits. Um vetor de entrada $\mathbf{u}_1^8$ ao passar por B_8 será representado por $\tilde{\mathbf{u}}_1^8 = (u_1, u_5, u_3, u_7, u_2, u_6, u_4, u_8)$. Uma implementação com $N = 8$ representando esta inversão de bits, pode ser observada na Figura 29. Ao se realizar uma análise detalhada dos nós do codificador mostrado na Figura 29, observa-se que as saídas $\mathbf{x}_1^8$ irão corresponder as seguintes entradas de $\tilde{\mathbf{u}}_1^8$

- $x_1 \rightarrow \tilde{u}_1, \tilde{u}_2, \tilde{u}_3, \tilde{u}_4, \tilde{u}_5, \tilde{u}_6, \tilde{u}_7, \tilde{u}_8$;
- $x_2 \rightarrow \tilde{u}_2, \tilde{u}_4, \tilde{u}_6, \tilde{u}_8$;
- $x_3 \rightarrow \tilde{u}_3, \tilde{u}_4, \tilde{u}_7, \tilde{u}_8$;

- $x_4 \to \tilde{u}_4, \tilde{u}_8$;
- $x_5 \to \tilde{u}_5, \tilde{u}_6, \tilde{u}_7, \tilde{u}_8$;
- $x_6 \to \tilde{u}_6, \tilde{u}_8$;
- $x_7 \to \tilde{u}_7, \tilde{u}_8$;
- $x_8 \to \tilde{u}_8$.

Figura 29 – Exemplo circuito da transformação $F^{\otimes 3}$.



Fonte: Próprio autor.

Este formato do codificador mostra que para as entradas $\tilde{\mathbf{u}}_1^8$, as respectivas saídas serão

$$\mathbf{x}_1^8 = \tilde{\mathbf{u}}_1^8 F^{\otimes 3}, \tag{4.35}$$

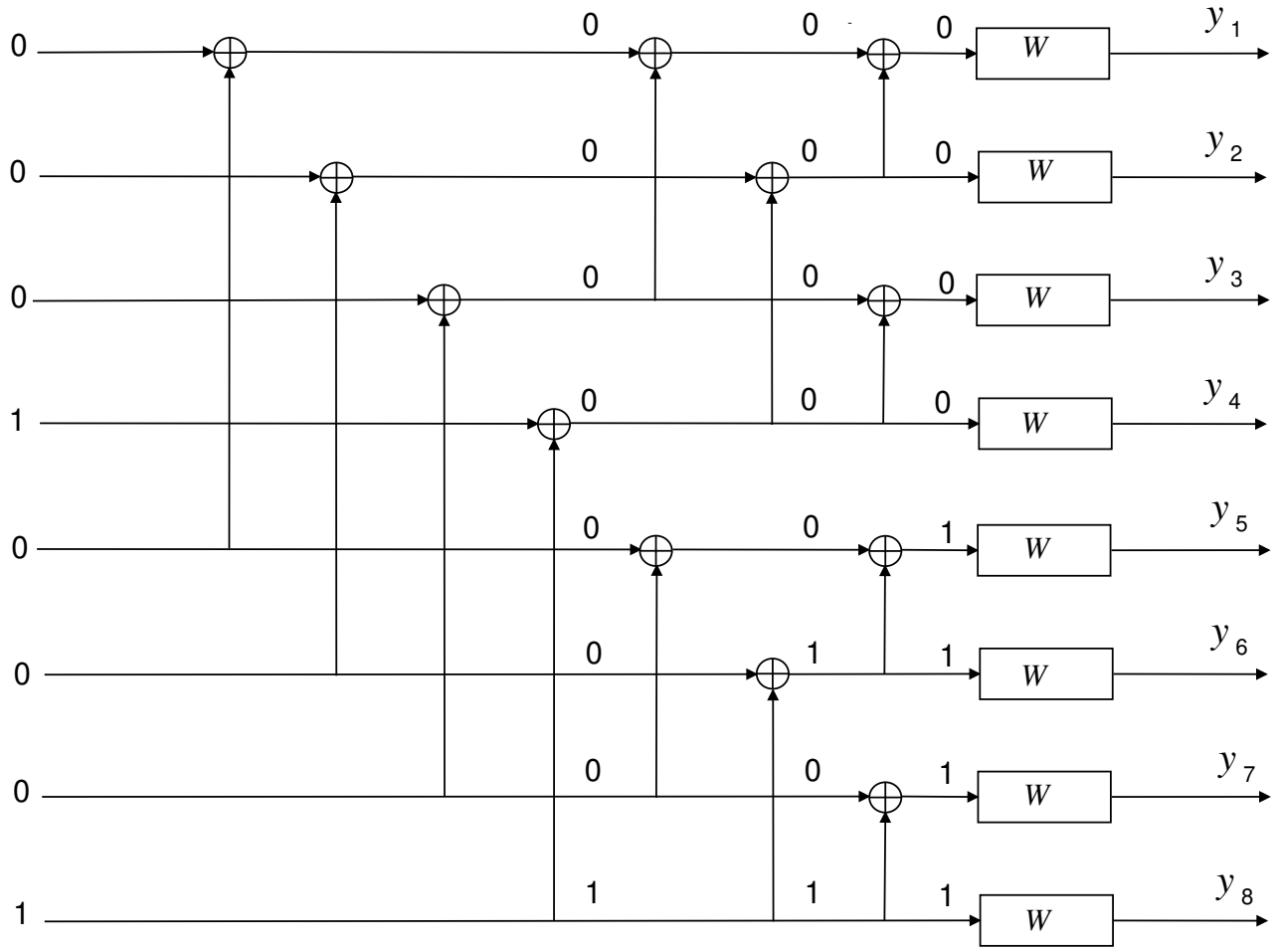
onde $F^{\otimes 3}$ é dada em (4.30).

Exemplo 4.2.4 *A partir dos cálculos do Exemplo 4.1.1 para a tomada de decisão da polarização de canal, considere um código polar* $(N, k, A, u_{A^c}) = (8, 4, \{4, 6, 7, 8\}, (0, 0, 0, 0))$*, ou seja, o código possui quatro entradas congeladas e quatro livres dadas por* u_4, u_6, u_7, u_8 *e os bits congelados são representados por* 0*. Vamos codificar a mensagem* $\boldsymbol{u} = (0, 0, 0, 0, 0, 0, 1, 1)$ *de duas formas. Inicialmente, utilizando a Equação 4.19, temos que*

$$
\begin{aligned}
\boldsymbol{x}_1^8 &= \boldsymbol{u}_1^8 G_8 \\
&= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \\
&= \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}
\end{aligned}
$$

Agora, após aplicar a inversão de bits do Exemplo 4.2.3 temos o vetor $\tilde{\boldsymbol{u}}_1^8 = (00010001)$. *Utilizando o circuito descrito na Figura 29 e a Equação 4.35 também podemos realizar a codificação. A Figura 30 mostra a codificação realizada.*

Figura 30 – Codificação **u**.



Fonte: Próprio autor.

$$
\begin{aligned}
\boldsymbol{x}_1^8 &= \tilde{\boldsymbol{u}}_1^8 F^{\otimes 3} \\
&= \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \\
&= \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix},
\end{aligned}
$$

onde $F^{\otimes 3}$ é dada em (4.30).

4.3 DECODIFICAÇÃO

Como a codificação de códigos polares é bem definida e facilmente implementada utilizando o produto de Kronecker de matrizes, a busca por melhorias no desempenho dos códigos polares é realizada na sua decodificação. Dessa forma, a decodificação dos códigos polares é um tópico muito estudado e métodos de decodificação mais eficientes buscam ser desenvolvidos para melhorar o desempenho de tradução de uma mensagem de interesse. Nesta seção serão apresentados dois decodificadores, o decodificador por cancelamento sucessivo (*Successive Cancellation Decoder - SC*) e o decodificador por cancelamento sucessivo simplificado (*Simplified Successive Cancellation Decoder - SSC*). Em ambas as técnicas, o decodificador dispõe das informações sobre os valores e posições dos bits congelados, u_{A^c} e A, respectivamente. Além disso, ambas as técnicas de decodificação são do tipo de máxima verossimilhança, análoga à vista na Seção 3.2.

Conforme é feita a decodificação do sinal, o vetor resultante $\hat{\mathbf{u}}_1^N$ é referente aos vetores $\mathbf{u}_1^N$. A razão de verossimilhança ($LikelihoodRatio$ - LR) é dada por

$$
L_N^{(i)}(\mathbf{y}_1^N, \mathbf{u}_1^{i-1}) = \frac{W_N^i(\mathbf{y}_1^N, \mathbf{u}_1^{i-1} | u_i = 0)}{W_N^i(\mathbf{y}_1^N, \mathbf{u}_1^{i-1} | u_i = 1)}. \tag{4.36}
$$

Considerando um decodificador de tamanho N, a Equação 4.36 pode ser calculada a partir das seguintes relações recursivas

$$
f(a, b) = \frac{(ab) + 1}{a + b} \tag{4.37}
$$

e

$$
g(a, b, \hat{u}_{i-1}) = a^{1-2\cdot\hat{u}_{i-1}} b, \tag{4.38}
$$

com $a, b \in \mathbb{R}$. A decodificação do sinal, ou seja, o vetor resultante $\hat{\mathbf{u}}_1^N$ com os bits estimados, é dado

por

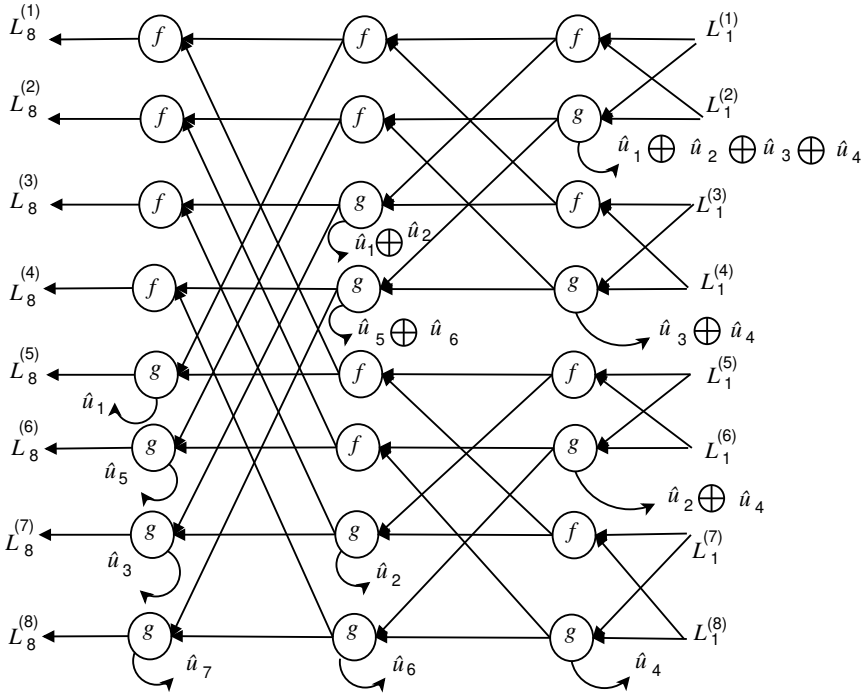
$$\hat{u}_i = \begin{cases} u_i, & \text{se } i \notin A^c \\ 0, & \text{se } i \in A^c \text{ e } L_N^{(i)}(\mathbf{y}_1^N, \hat{\mathbf{u}}_1^{i-1}) \geq 1 \\ 1, & \text{se } i \in A^c \text{e } L_N^{(i)}(\mathbf{y}_1^N, \hat{\mathbf{u}}_1^{i-1}) < 1 \end{cases} . \tag{4.39}$$

4.3.1 Decodificador por Cancelamento Sucessivo

Proposto por Arikan, [ARIKAN 2009], o decodificador SC foi o primeiro decodificador desenvolvido para ser utilizado em códigos polares. Este decodificador realiza as operações em todos os nós da arquiterura do código, ou seja, independente de saber os valores dos bits congelados este decodificador determina todos os nós para encontrar a mensagem de interesse.

A Figura 31 representa a decodificação SC para um canal com $N = 8$. Neste caso, a decodificação apresenta três estágios, os quais são vistos da direita para a esquerda, e cada estágio disponibiliza as informações para o próximo estágio. Em cada nó de cada estágio é realizada uma operação com as fórmulas recursivas (4.37) e (4.38).

Figura 31 – Exemplo arquitetura de decodificação SC com N = 8.



Fonte: Próprio autor.

No Exemplo 4.3.1 será realizado o passo a passo da decodificação.

Exemplo 4.3.1 *Considere um código de tamanho $N = 8$ com taxa 1/2, transmitido por um canal AWGN com $E_b/N_0 = 1$ dB. Baseando-se no Exemplo 4.1.1 as entradas (u_1, u_2, u_3, u_5) serão congeladas e serão transmitidos bits 0's. Supondo que na saída deste canal com ruído AWGN, foi recebido o seguinte vetor*

***y** = (-1.1241, 2.4896, 0.4090, 2.4171, 1.6714, -2.2074, 1.7172, 0.6302).*

Aplicando as Equações (4.37) e (4.38), juntamente da relação (4.39), iremos realizar a decodificação deste vetor por cancelamento sucessivo. Os valores encontrados em cada etapa da decodificação são apresentados na Figura 32 e seus cálculos explicados a seguir. Inicialmente, como a decodificação começa da direita para a esquerda, os valores recebidos são utilizados para calcular todos os valores de f como segue

- *Para* $L_2^{(1)} \longrightarrow f(a,b), a = L_1^{(1)}$ *e* $b = L_1^{(2)}$, *temos*

$$f(-1.1241, 2.4896) = \frac{-1.1241 \cdot (2.4896) + 1}{-1.1241 + (2.4896)} = -1.3172.$$

- *Para* $L_2^{(3)} \longrightarrow f(a,b), a = L_1^{(3)}$ *e* $b = L_1^{(4)}$, *temos*

$$f(0.4090, 2.4171) = \frac{0.4090 \cdot (2.4171) + 1}{0.4090 + (2.4171)} = 0.7036.$$

- *Para* $L_2^{(5)} \longrightarrow f(a,b), a = L_1^{(5)}$ *e* $b = L_1^{(6)}$, *temos*

$$f(1.6714, -2.2074) = \frac{1.6714 \cdot (-2.2074) + 1}{1.6714 + (-2.2074)} = 5.0183.$$

- *Para* $L_2^{(7)} \longrightarrow f(a,b), a = L_1^{(7)}$ *e* $b = L_1^{(8)}$, *temos*

$$f(1.7172, 0.6302) = \frac{1.7172 \cdot (0.6302) + 1}{1.7172 + (0.6302)} = 0.8870.$$

Após realizar todos esses cálculos, o decodificador avança de estágio apenas com esses valores definidos e realiza os seguintes cálculos

- *Para* $L_4^{(1)} \longrightarrow f(a,b), a = L_2^{(1)}$ *e* $b = L_2^{(3)}$, *temos*

$$f(-1.3172, 0.7036) = \frac{-1.3172 \cdot (0.7036) + 1}{-1.3172 + (0.7036)} = -0.1191.$$

- *Para* $L_4^{(5)} \longrightarrow f(a,b), a = L_2^{(5)}$ *e* $b = L_2^{(7)}$, *temos*

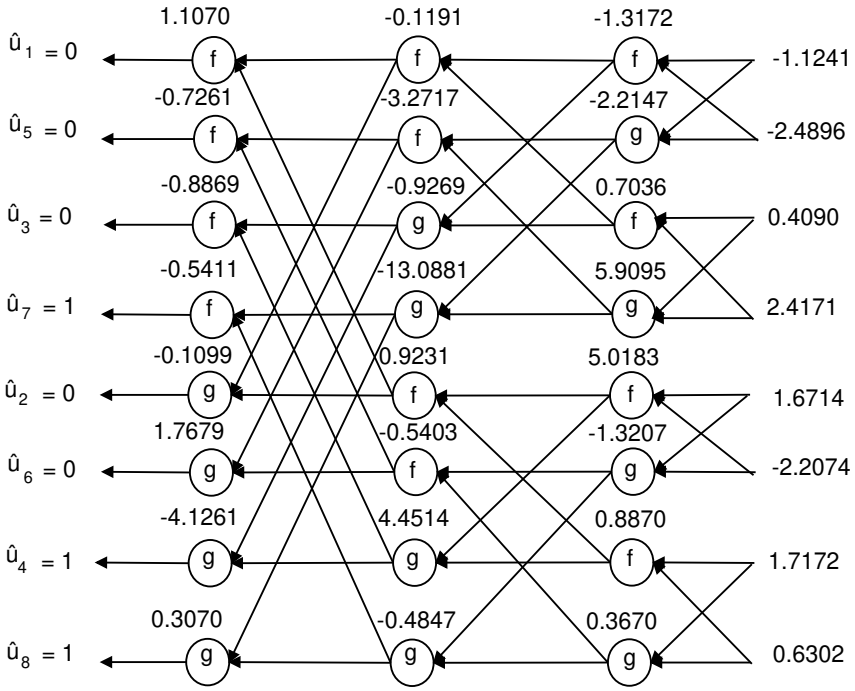
$$f(5.0183, 0.8870) = \frac{5.0183 \cdot (0.8870) + 1}{5.0183 + (0.8870)} = 0.9231.$$

Logo após, o decodificador avança mais uma vez de estágio e realiza os seguintes cálculos

- *Para* $L_8^{(1)} \longrightarrow f(a,b), a = L_4^{(1)}$ *e* $b = L_4^{(5)}$, *temos*

$$f(-0.1191, 0.9231) = \frac{-0.1191 \cdot (0.9231) + 1}{-0.1191 + (0.9231)} = 1.1070.$$

Figura 32 – Exemplo decodificação SC com N = 8.



Fonte: Próprio autor.

Nesse momento, o decodificador verifica primeiramente se o vetor recebido é um bit congelado ou um bit de informação. Dessa forma, como o exemplo considera que o vetor u_1 é um bit congelado, então $\hat{u}_1 = 0$. Seguindo, o decodificador calcula a segunda posição do vetor pela função g, que depende apenas da primeira posição do vetor

- *Para $L_8^{(5)} \longrightarrow g(a, b, \hat{u}_{i-1}), a = L_4^{(1)}$, $b = L_4^{(5)}$ e $\hat{u}_{i-1} = 1$, temos*

$$g(-0.1191, 0.9231, 0) = -0.1191^{1-2\cdot 0} \cdot 0.9231 = -0.1099.$$

Como consideramos o vetor u_2 também um bit congelado, então $\hat{u}_2 = 0$. Neste momento o decodificador tem os valores de $\hat{u}_1$ e $\hat{u}_2$, ele os utiliza para calcular a terceira posição do vetor.

- *Para $L_4^{(3)} \longrightarrow g(a, b, \hat{u}_{i-1}), a = L_2^{(1)}$, $b = L_2^{(3)}$ e $\hat{u}_{i-1} = 1$, temos*

$$g(-1.3172, 0.7036, 0) = -1.3172^{1-2\cdot 0} \cdot 0.7036 = -0.9269.$$

- *Para $L_4^{(7)} \longrightarrow g(a, b, \hat{u}_{i-1}), a = L_2^{(5)}$, $b = L_2^{(7)}$ e $\hat{u}_{i-1} = 0$, temos*

$$g(5.0183, 0.8870, 0) = 5.0183^{1-2\cdot 0} \cdot 0.8870 = 4.4514.$$

- *Para $L_8^{(3)} \longrightarrow f(a, b), a = L_4^{(3)}$ e $b = L_4^{(7)}$, temos*

$$f(-0.9269, 4.4514) = \frac{-0.9269 \cdot (4.4514) + 1}{-0.9269 + (4.4514)} = -0.8869.$$

Como o vetor u_3 é um bit congelado, $\hat{u}_3 = 0$. Seguindo, para calcular a quarta posição do vetor

- *Para $L_8^{(7)} \longrightarrow g(a, b, \hat{u}_{i-1}), a = L_4^{(3)}$, $b = L_4^{(7)}$ e $\hat{u}_{i-1} = 0$, temos*

$$g(-0.9269, 4.4514, 0) = -0.9269^{1-2\cdot 0} \cdot 4.4514 = -4.1261.$$

Por (4.39), concluímos que $\hat{u}_4 = 1$. Assim, como o decodificador identificou os quatro primeiros valores do vetor enviado, ele os utiliza para calcular a parte restante do primeiro estágio. Sendo assim, temos

- *Para $L_2^{(2)} \longrightarrow g(a, b, \hat{u}_{i-1}), a = L_1^{(1)}$, $b = L_1^{(2)}$ e $\hat{u}_{i-1} = 1$, temos*

$$g(-1.1241, 2.4896, 1) = -1.1241^{1-2\cdot 1} \cdot (2.4896) = -2.2147.$$

- *Para $L_2^{(4)} \longrightarrow g(a, b, \hat{u}_{i-1}), a = L_1^{(3)}$, $b = L_1^{(4)}$ e $\hat{u}_{i-1} = 0$, temos*

$$g(0.4090, 2.4171, 1) = 0.4090^{1-2\cdot 1} \cdot (2.4171) = 5.9095.$$

- *Para $L_2^{(6)} \longrightarrow g(a, b, \hat{u}_{i-1}), a = L_1^{(5)}$, $b = L_1^{(6)}$ e $\hat{u}_{i-1} = 0$, temos*

$$g(1.6714, -2.2074, 0) = 1.6714^{1-2\cdot 0} \cdot (-2.2074) = -1.3207.$$

- *Para $L_2^{(8)} \longrightarrow g(a, b, \hat{u}_{i-1}), a = L_1^{(7)}$, $b = L_1^{(8)}$ e $\hat{u}_{i-1} = 0$, temos*

$$g(1.7172, 0.6302, 0) = 1.7172^{1-2\cdot 1} \cdot (0.6302) = 0.3670.$$

Passando para o próximo estágio

- *Para $L_4^{(2)} \longrightarrow f(a, b), a = L_2^{(2)}$ e $b = L_2^{(4)}$, temos*

$$f(-2.2147, 5.9095) = \frac{(-2.2147) \cdot (5.9095) + 1}{(-2.2147) + (5.9095)} = -3.2717.$$

- *Para $L_4^{(6)} \longrightarrow f(a, b), a = L_2^{(6)}$ e $b = L_2^{(8)}$, temos*

$$f(-1.3207, 0.3670) = \frac{(-1.3207) \cdot (0.3670) + 1}{(-1.3207) + (0.3670)} = -0.5403.$$

- *Para $L_8^{(2)} \longrightarrow f(a, b), a = L_4^{(2)}$ e $b = L_4^{(6)}$, temos*

$$f(-3.2717, -0.5403) = \frac{(-3.2717) \cdot (-0.5403) + 1}{(-3.2717) + (-0.5403)} = -0.8809.$$

Como o vetor u_5 é um bit congelado, então $\hat{u}_5 = 0$. Assim, para calcular a sexta posição do vetor

- *Para* $L_8^{(6)} \longrightarrow g(a, b, \hat{u}_{i-1}), a = L_4^{(2)}$, $b = L_4^{(6)}$ *e* $\hat{u}_{i-1} = 1$, *temos*

$$g(-3.2717, -0.5403, 0) = (-3.2717)^{1-2\cdot 0} \cdot (-0.5403) = 1.7679.$$

Por (4.39), temos que $\hat{u}_6 = 0$. *Para calcular* $\hat{u}_7$, *temos*

- *Para* $L_4^{(4)} \longrightarrow g(a, b, \hat{u}_{i-1}), a = L_2^{(2)}$, $b = L_2^{(4)}$ *e* $\hat{u}_{i-1} = 1$, *temos*

$$g(-2.2147, 5.9095, 0) = (-2.2147)^{1-2\cdot 0} \cdot (5.9095) = -13.0881.$$

- *Para* $L_4^{(8)} \longrightarrow g(a, b, \hat{u}_{i-1}), a = L_2^{(6)}$, $b = L_2^{(8)}$ *e* $\hat{u}_{i-1} = 0$, *temos*

$$g(-1.3207, 0.3670, 0) = (-1.3207)^{1-2\cdot 0} \cdot (0.3670) = -0.4847.$$

- *Para* $L_8^{(4)} \longrightarrow f(a, b), a = L_4^{(4)}$ *e* $b = L_4^{(8)}$, *temos*

$$f(-13.0881, -0.4847) = \frac{-13.0881 \cdot -0.4847 + 1}{-13.0881 + -0.4847} = -0.5411.$$

Pela relação (4.39), concluímos que $\hat{u}_7 = 1$. *Por fim,*

- *Para* $L_8^{(8)} \longrightarrow g(a, b, \hat{u}_{i-1}), a = L_4^{(4)}$, $b = L_4^{(8)}$ *e* $\hat{u}_{i-1} = 0$, *temos*

$$g(-13.0881, -0.4847, 0) = -13.0881^{1-2\cdot 0} \cdot -0.4847 = 0.0370.$$

Novamente pela relação (4.39), temos que $\hat{u}_8 = 1$. *Portanto, o vetor decodificado é*

$$\hat{\boldsymbol{u}} = (0, 0, 0, 1, 0, 0, 1, 1).$$

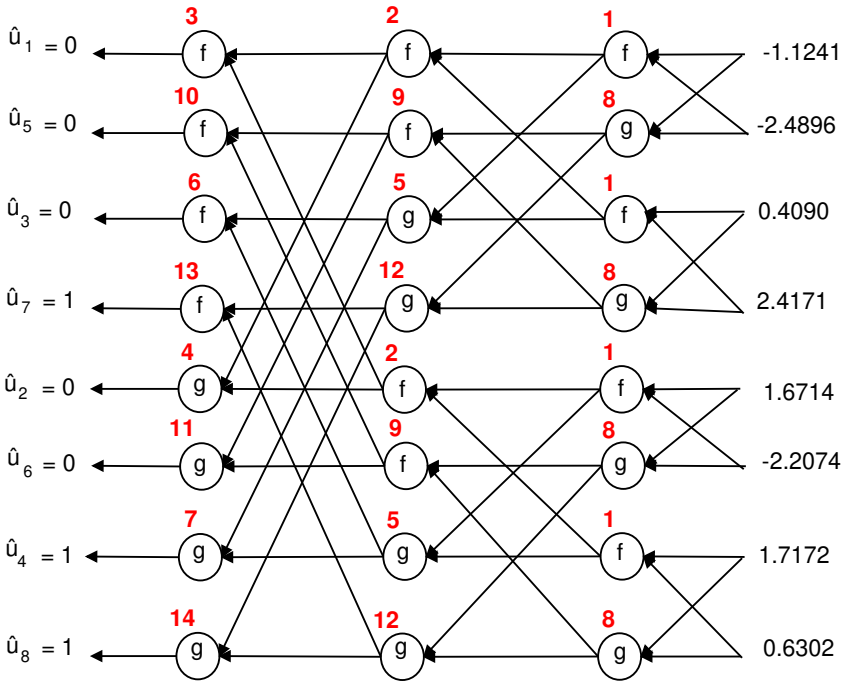
Podemos observar pelo exemplo que todos os nós do código foram determinados mesmo que o decodificador já tenha o valor do bit congelado, deste modo, pode-se dizer que este decodificador tem uma perda de processamento, afinal, aloca recursos para operações desnecessárias.

Uma análise importante do decodificador é determinar a quantidade de ciclos de *clocks* que o processador precisa para decodificar uma mensagem de interesse. Entende-se por *clock*, ou ciclo de *clock*, como sendo o número de ações que o processador consegue executar por segundo, como inicializar um programa, escrever algo na memória ou realizar alguns cálculos.

Na Figura 33 podemos observar uma representação da quantidade de *clocks* que o processador necessita para decodificar a mensagem do Exemplo 4.3.1.

Cada *clock* do processador calcula o máximo de nós possíveis com a informação que tem no momento. No primeiro *clock* temos o cálculo de 4 nós f, já no segundo, 2 nós f, no terceiro um nó f, determinando assim o primeiro bit de informação e assim por diante. Observa-se que são necessários 14 ciclos de *clock* para decodificar a mensagem completa.

Figura 33 – Ciclo de *clock* do processador nos decodificador SC.

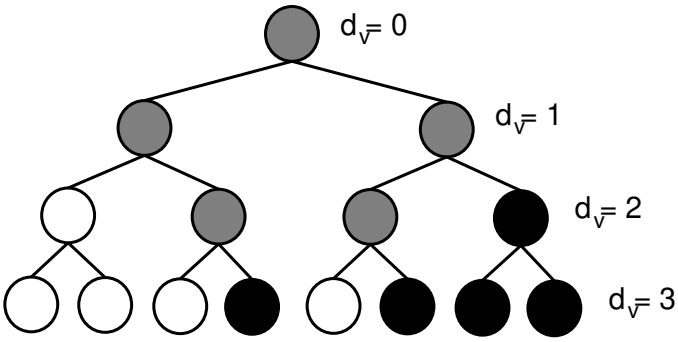


Fonte: Próprio autor.

A decodificação deste código pode ser representada a partir de um gráfico de árvore, nesta representação cada bit estimado $\hat{\mathbf{u}}_i$ é representado por um nó. Temos dois tipos de nós $\mathcal{N}_0$ que corresponde aos bits congelados e $\mathcal{N}_1$ que corrensponde aos bits de informação. A origem de dois nós $\mathcal{N}_0$ é também um nó $\mathcal{N}_0$ e possui taxa de informação igual a zero. A combinação de dois nós $\mathcal{N}_1$ correspodem a um nó origem também $\mathcal{N}_1$, que possui taxa de informação igual a 1. Já um nó origem de dois nós de diferentes tipos é um nó $\mathcal{N}_R$ e possui taxa $0 < R < 1$.

Na Figura 34 temos a representação de um código polar $(N, k) = (8, 4)$. Os circulos brancos são nós $\mathcal{N}_0$, os circulos pretos representam os nós $\mathcal{N}_1$, os circulos cinza representam os nós $\mathcal{N}_R$ e d_v é a profundidade de v no código.

Figura 34 – Arquiterura em árvore SC com N = 8 e R = 1/2.



Fonte: Próprio autor.

Cada um dos circulos representa um *clock* que o processador terá que dar para fazer os cálculos referente a este nó. Em decodificadores SC, todos os nós são calculados, então podemos dizer que a quantidade de *clocks* que um decodificador utiliza para decodificar uma mensagem é dada

por, [LEROUX et al. 2011, ZHANG e PARHI 2014],

$$clock_{SC} = 2(N-1). \tag{4.40}$$

Uma outra medida importante de um código é o *throughput*, ou seja, a taxa de processamento de informação. Este parâmetro mede a quantidade de bits que são decodificados por segundo levando em consideração um ciclo de *clock* do processador. O *throughput* de um decodificador SC é dado pela seguinte equação

$$T = \frac{k}{clock_{SC} * t_p}, \tag{4.41}$$

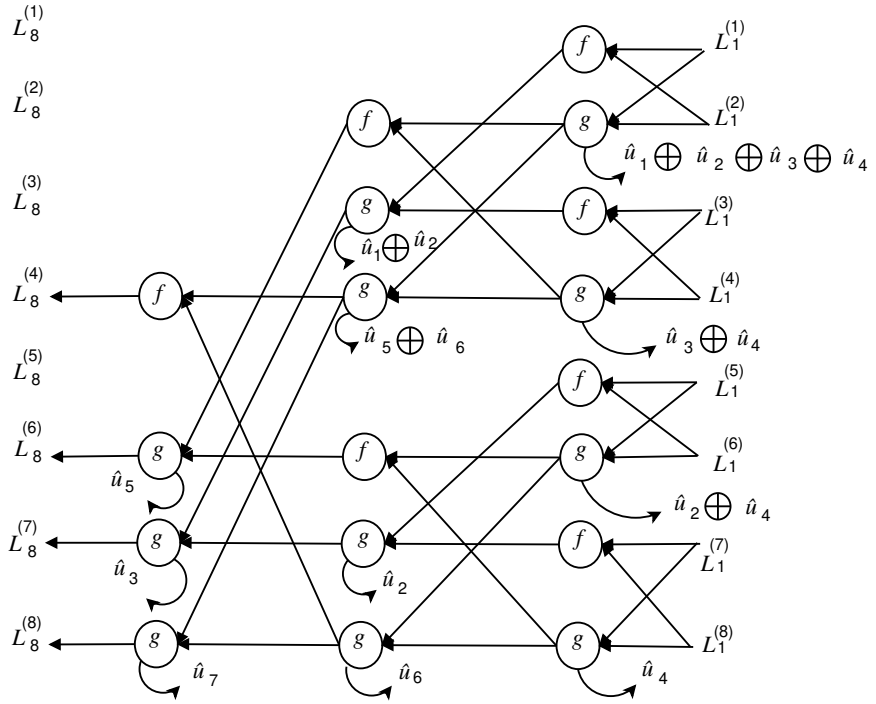
onde T é dado em bits/s e t_p é o tempo necessário para que o processador codifique e decodifique uma mensagem de interesse, [LEROUX et al. 2011, SARKIS e GROSS 2013a].

4.3.2 Decodificador por Cancelamento Sucessivo Simplificado

Este decodificador é uma melhoria do decodificador SC, pois sabendo a informação dos valores dos bits congelados, os nós que seriam necessários para determinar estes bits não são calculados. Deste modo, tem-se um ganho de processamento, pois recursos não são gastos desnecessariamente e há melhoria de latência, pois a mensagem de interesse é decodificada mais rapidamente.

A Figura 35 representa um canal $N = 8$, podemos observar que este decodificador também apresenta três estágios, entretanto, o cálculo de alguns nós não são necessários para determinar os bits congelados. No Exemplo 4.3.2 será realizado o passo a passo da decodificação.

Figura 35 – Exemplo arquitetura de decodificação SSC com N = 8.



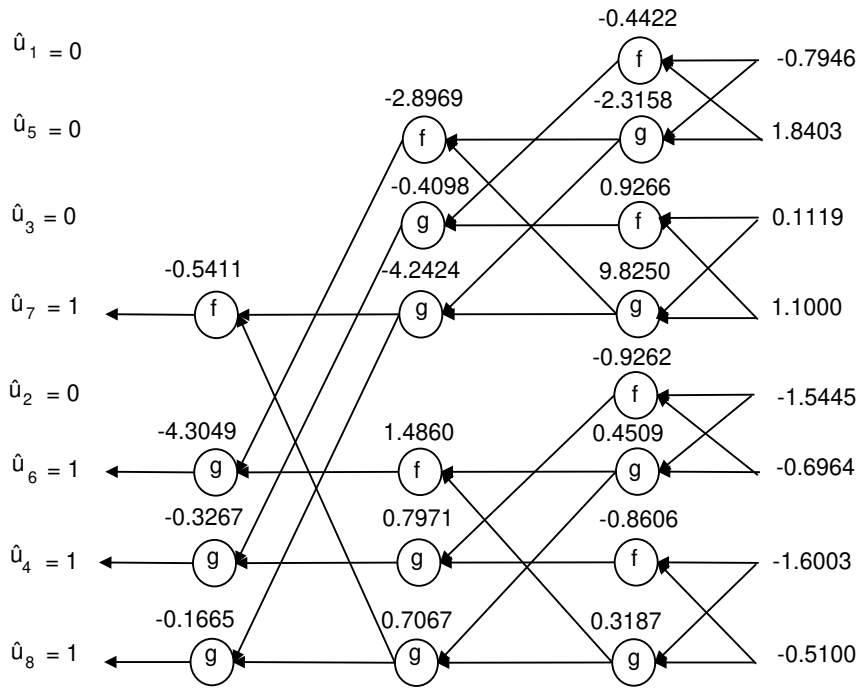
Fonte: Próprio autor.

Exemplo 4.3.2 *Considere um código de tamanho $N = 8$ com taxa 1/2, transmitido por um canal AWGN com $E_b/N_0 = 1$ dB. Baseando-se no Exemplo 4.1.1 as entradas (u_1, u_2, u_3, u_5) serão congeladas e serão transmitidos bits 0's. Supondo que na saída deste canal com ruído AWGN, foi recebido o seguinte vetor*

$$\mathbf{y} = (-0.7946,\ 1.8403,\ 0.1119,\ 1.1000,\ -1.5445,\ -0.6964,\ -1.6003,\ -0.5100).$$

Aplicando as Equações (4.37) e (4.38), juntamente da relação (4.39), e sabendo que $(u_1 = 0, u_2 = 0, u_3 = 0, u_5 = 0)$ iremos realizar a decodificação deste vetor por cancelamento sucessivo simplificado. Os valores encontrados em cada etapa da decodificação são apresentados na Figura 36 e explicados a seguir. Inicialmente, como a decodificação começa da direita para a esquerda, os valores recebidos são utilizados para calcular todos os valores de f como segue

Figura 36 – Exemplo decodificação SSC com N = 8.



Fonte: Próprio autor.

- *Para* $L_2^{(1)} \longrightarrow f(a,b), a = L_1^{(1)}$ *e* $b = L_1^{(2)}$, *temos*

$$f(-0.7946, 1.8403) = \frac{-0.7946 \cdot (1.8403) + 1}{-0.7946 + (1.8403)} = -0.4422.$$

- *Para* $L_2^{(3)} \longrightarrow f(a,b), a = L_1^{(3)}$ *e* $b = L_1^{(4)}$, *temos*

$$f(0.1119, 1.1000) = \frac{0.1119 \cdot (1.1000) + 1}{0.1119 + (1.1000)} = 0.9266.$$

- *Para* $L_2^{(5)} \longrightarrow f(a,b), a = L_1^{(5)}$ *e* $b = L_1^{(6)}$, *temos*

$$f(-1.5445, -0.6964) = \frac{-1.5445 \cdot (-0.6964) + 1}{-1.5445 + (-0.6964)} = -0.9262.$$

- *Para* $L_2^{(7)} \longrightarrow f(a,b), a = L_1^{(7)}$ *e* $b = L_1^{(8)}$, *temos*

$$f(-1.6003, -0.5100) = \frac{-1.6003 \cdot (-0.5100) + 1}{-1.6003 + (-0.5100)} = -0.8606.$$

Após realizar todos esses cálculos, o decodificador avança de estágio com esses valores definidos e realiza os seguintes cálculos

- *Para* $L_4^{(3)} \longrightarrow g(a,b,\hat{u}_{i-1}), a = L_2^{(1)}$, $b = L_2^{(3)}$ *e* $\hat{u}_{i-1} = 1$, *temos*

$$g(-0.4422, 0.9266, 0) = -0.4422^{1-2\cdot 0} \cdot 0.9266 = 0.4098.$$

- *Para* $L_4^{(7)} \longrightarrow g(a,b,\hat{u}_{i-1}), a = L_2^{(5)}$, $b = L_2^{(7)}$ *e* $\hat{u}_{i-1} = 0$, *temos*

$$g(-0.9262, -0.8606, 0) = -0.9262^{1-2\cdot 0} \cdot -0.8606 = 0.7971.$$

- *Para* $L_8^{(7)} \longrightarrow g(a,b,\hat{u}_{i-1}), a = L_4^{(3)}$, $b = L_4^{(7)}$ *e* $\hat{u}_{i-1} = 0$, *temos*

$$g(0.4098, 0.7971, 0) = 0.4098^{1-2\cdot 0} \cdot 0.7971 = -0.3267.$$

A partir de (4.39), temos que $\hat{u}_4 = 1$. *Assim, como o decodificador sabe os quatro primeiros valores do vetor enviado, ele os utiliza para calcular a parte restante do primeiro estágio. Sendo assim, temos*

- *Para* $L_2^{(2)} \longrightarrow g(a,b,\hat{u}_{i-1}), a = L_1^{(1)}$, $b = L_1^{(2)}$ *e* $\hat{u}_{i-1} = 1$, *temos*

$$g(-0.7946, 1.8403, 1) = -0.7946^{1-2\cdot 1} \cdot (1.8403) = -2.3158.$$

- *Para* $L_2^{(4)} \longrightarrow g(a,b,\hat{u}_{i-1}), a = L_1^{(3)}$, $b = L_1^{(4)}$ *e* $\hat{u}_{i-1} = 0$, *temos*

$$g(0.1119, 1.1000, 1) = 0.1119^{1-2\cdot 1} \cdot (1.1000) = 9.8250.$$

- *Para* $L_2^{(6)} \longrightarrow g(a,b,\hat{u}_{i-1}), a = L_1^{(5)}$, $b = L_1^{(6)}$ *e* $\hat{u}_{i-1} = 0$, *temos*

$$g(-1.5445, -0.6964, 1) = -1.5445^{1-2\cdot 1} \cdot (-0.6964) = 0.4509.$$

- *Para* $L_2^{(8)} \longrightarrow g(a,b,\hat{u}_{i-1}), a = L_1^{(7)}$, $b = L_1^{(8)}$ *e* $\hat{u}_{i-1} = 0$, *temos*

$$g(-1.6003, -0.5100, 0) = -1.6003^{1-2\cdot 1} \cdot (-0.5100) = 0.3187.$$

Passando para o próximo estágio

- *Para* $L_4^{(2)} \longrightarrow f(a,b), a = L_2^{(2)}$ *e* $b = L_2^{(4)}$, *temos*

$$f(-2.3158, 9.8250) = \frac{(-2.3158) \cdot (9.8250) + 1}{(-2.3158) + (9.8250)} = -2.8969.$$

- *Para* $L_4^{(6)} \longrightarrow f(a,b), a = L_2^{(6)}$ *e* $b = L_2^{(8)}$*, temos*

$$f(0.4509, 0.3187) = \frac{(0.4509)\cdot(0.3187)+1}{(0.4509)+(0.3187)} = 1.4860.$$

- *Para* $L_8^{(6)} \longrightarrow g(a,b,\hat{u}_{i-1}), a = L_4^{(2)}$*,* $b = L_4^{(6)}$ *e* $\hat{u}_{i-1} = 1$*, temos*

$$g(-2.8969, 1.4860, 0) = (-2.8969)^{1-2\cdot 0}\cdot(1.4860) = -4.3049.$$

Por (4.39), segue que $\hat{u}_6 = 1$*. Para calcular* $\hat{u}_7$*, temos*

- *Para* $L_4^{(4)} \longrightarrow g(a,b,\hat{u}_{i-1}), a = L_2^{(2)}$*,* $b = L_2^{(4)}$ *e* $\hat{u}_{i-1} = 1$*, temos*

$$g(-2.3158, 9.8250, 1) = (-2.3158)^{1-2\cdot 1}\cdot(9.8250) = -4.2424.$$

- *Para* $L_4^{(8)} \longrightarrow g(a,b,\hat{u}_{i-1}), a = L_2^{(6)}$*,* $b = L_2^{(8)}$ *e* $\hat{u}_{i-1} = 0$*, temos*

$$g(0.4509, 0.3187, 1) = (0.4509)^{1-2\cdot 1}\cdot(0.3187) = 0.7067.$$

- *Para* $L_8^{(4)} \longrightarrow f(a,b), a = L_4^{(4)}$ *e* $b = L_4^{(8)}$*, temos*

$$f(-4.2424, 0.7067) = \frac{-4.2424\cdot 0.7067+1}{-4.2424+0.7067} = 0.5652.$$

Pela relação (4.39), temos que $\hat{u}_7 = 1$*.*

- *Para* $L_8^{(8)} \longrightarrow g(a,b,\hat{u}_{i-1}), a = L_4^{(4)}$*,* $b = L_4^{(8)}$ *e* $\hat{u}_{i-1} = 0$*, temos*

$$g(-4.2424, 0.7067, 1) = -4.2424^{1-2\cdot 1}\cdot 0.7067 = -0.1665.$$

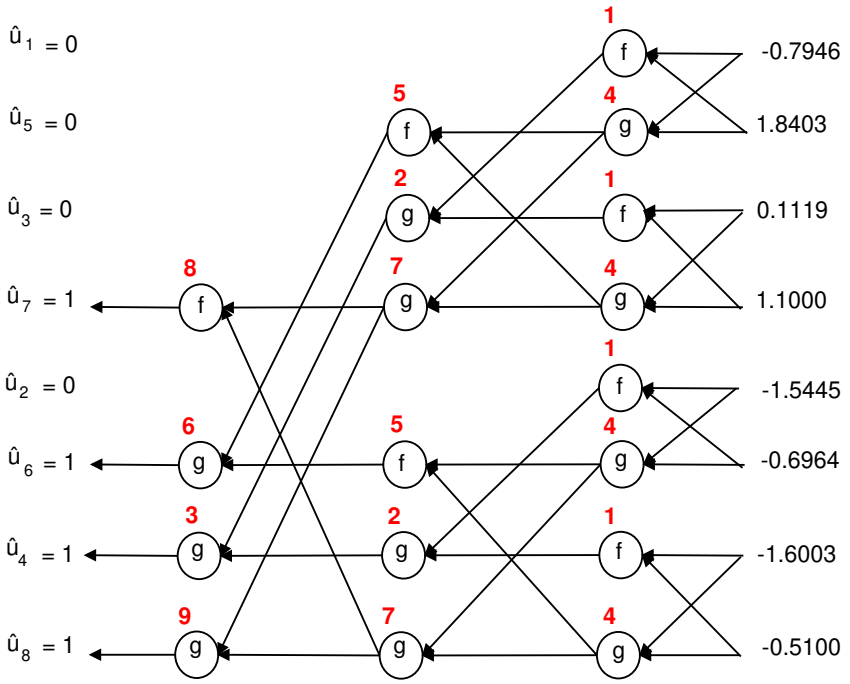
Novamente pela relação (4.39), concluímos que $\hat{u}_8 = 1$*. Portanto, o vetor original é*

$$\hat{\boldsymbol{u}} = (0,0,0,1,0,1,1,1).$$

Podemos observar pelo exemplo que nem todos os nós do código foram determinados, pois o decodificador já possui a informação dos bits congelados. Na Figura 37 podemos observar uma imagem representando a quantidade de *clocks* que um processador necessita para decodificar a mensagem do Exemplo 4.3.2.

No primeiro *clock* temos o cálculo de 4 nós f, já no segundo, 2 nós g, no terceiro um nó g, determinando assim o primeiro bit de informação e assim por diante. Observa-se que são necessários 9 ciclos de *clock* para decodificar a mensagem completa. Logo, decodificadores SSC utilizam menos *clocks* do processador para decodificar uma mesma mensagem, visto que não necessitam calcular os nós $\mathcal{N}_0$.

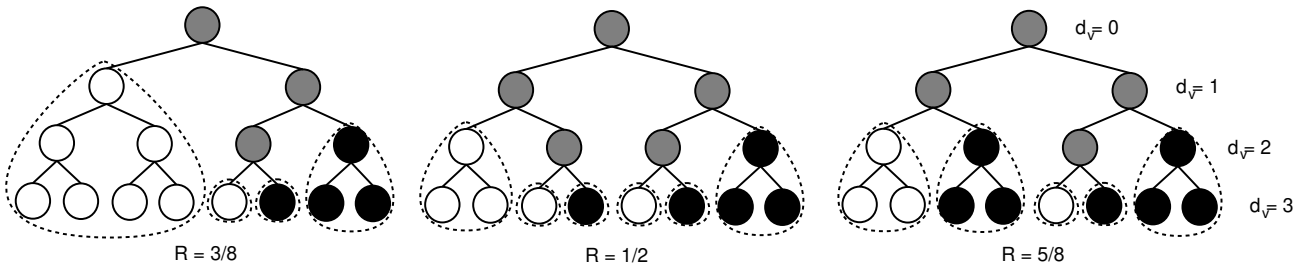
Figura 37 – Ciclo de *clock* do processador no decodificador SSC.



Fonte: Próprio autor.

Na Figura 38 podemos observar a representação de arvores de decodificadores SSC com diferentes taxas.

Figura 38 – Decodificadores em árvore SSC para diferentes taxas.



Fonte: Próprio autor.

A quantidade de ciclos de *clock* necessários para decodificar um decodificador SSC é dada pela seguinte equação

$$clock_{SSC} = 2(N-1) - \sum_i (2^{d_i+1} - 1) - \sum_j [(2^{d_j+1} - 1) - (d_j + 1)], \tag{4.42}$$

onde i e j, representam a quantidade de sub árvores formadas por nós $\mathcal{N}_0$ e $\mathcal{N}_1$, respectivamente, que estão contidas na árvore de decodificação, [ZHANG e PARHI 2014]. Vale ressaltar que esta equação conta o primeiro *clock* como sendo o *clock* 0, assim, vamos somar 1 *clock* na Equação 4.42 para contabilizar este *clock*, de modo

$$clock_{SSC} = 1 + 2(N-1) - \sum_i (2^{d_i+1} - 1) - \sum_j [(2^{d_j+1} - 1) - (d_j + 1)]. \tag{4.43}$$

Exemplo 4.3.3 *Considerando a Figura 38 vamos realizar a contagem de clocks para diferentes taxas de codificação.*

- *Para R = 3/8, temos duas sub-árvores de $\mathcal{N}_0$, uma com $d_i = 2$ e outra com $d_i = 0$, duas $\mathcal{N}_1$, uma com $d_i = 1$ e outra com $d_i = 0$. Assim,*

$$clock_{SSC} = 1 + 14 - (7 + 1) - [(3 + 1) - (2 + 1)] = 6.$$

- *Para R = 1/2, temos três sub-árvores de $\mathcal{N}_0$, uma com $d_i = 1$ e duas com $d_i = 0$. Para $\mathcal{N}_1$ temos uma com $d_i = 1$ e duas com $d_i = 0$. Assim,*

$$clock_{SSC} = 1 + 14 - (3 + 1 + 1) - [(3 + 1 + 1) - (2 + 1 + 1)] = 9.$$

- *Para R = 5/8, temos duas sub-árvores de $\mathcal{N}_0$, uma com $d_i = 1$ e outra com $d_i = 0$. Para $\mathcal{N}_1$ temos duas com $d_i = 1$ e uma com $d_i = 0$. Assim,*

$$clock_{SSC} = 1 + 14 - (3 + 1) - [(3 + 3 + 1) - (2 + 2 + 1)] = 9.$$

Utilizando (4.41) e (4.42), o *throughput* de um decodificador SSC será

$$T = \frac{k}{clock_{SSC} * t_p}. \tag{4.44}$$

No Capítulo 5 será realizada uma implementação computacional dos códigos polares que tem como objetivo comparar o desempenho deste código com os códigos convolucionais para correção e detecção de erros, assim como a latência e o *throughput*.

5 IMPLEMENTAÇÃO, ANÁLISES E RESULTADOS

Neste capítulo, apresentaremos as simulações realizadas e os resultados obtidos de uma implementação computacional no software MATLAB. Inicialmente, será realizada a implementação dos códigos convolucionais e polares, para avaliação da taxa de erro de bit (*Bit Error Rate* - BER), para posteriormente, comparar o desempenho destes códigos em cenários URC (*Ultra Reliability Communications*), com transmissão de pacotes curtos, analisando a latência e o *throughput* como previsto nos sistemas 5G.

A abordagem de Monte Carlo foi utilizada na implementação, a mesma baseia-se na Lei dos Grandes Números, que afirma que ao realizar um mesmo experimento um elevado número de vezes a média da estatística de interesse converge para o parâmetro, ou seja, a probabilidade do evento de interesse ocorrer é dado pela frequência relativa, [JACOBINI e LUGLI 2012]. As simulações de Monte Carlo são definidas como um método estatístico que se baseia na sucessiva simulação de várias realizações de um mesmo experimento aleatório a fim de obter um resultado numérico aproximado, ou seja, repete-se uma mesma simulação aleatória um elevado número de vezes para encontrar uma estimativa do parâmetro de interesse a partir da estatística correspondente calculada da distribuição amostral. A estimativa de um evento de interesse (A) utilizando o método de Monte Carlo é dada da seguinte forma

$$P_r(A) = \lim_{N' \to \infty} \frac{N'_A}{N'}, \tag{5.1}$$

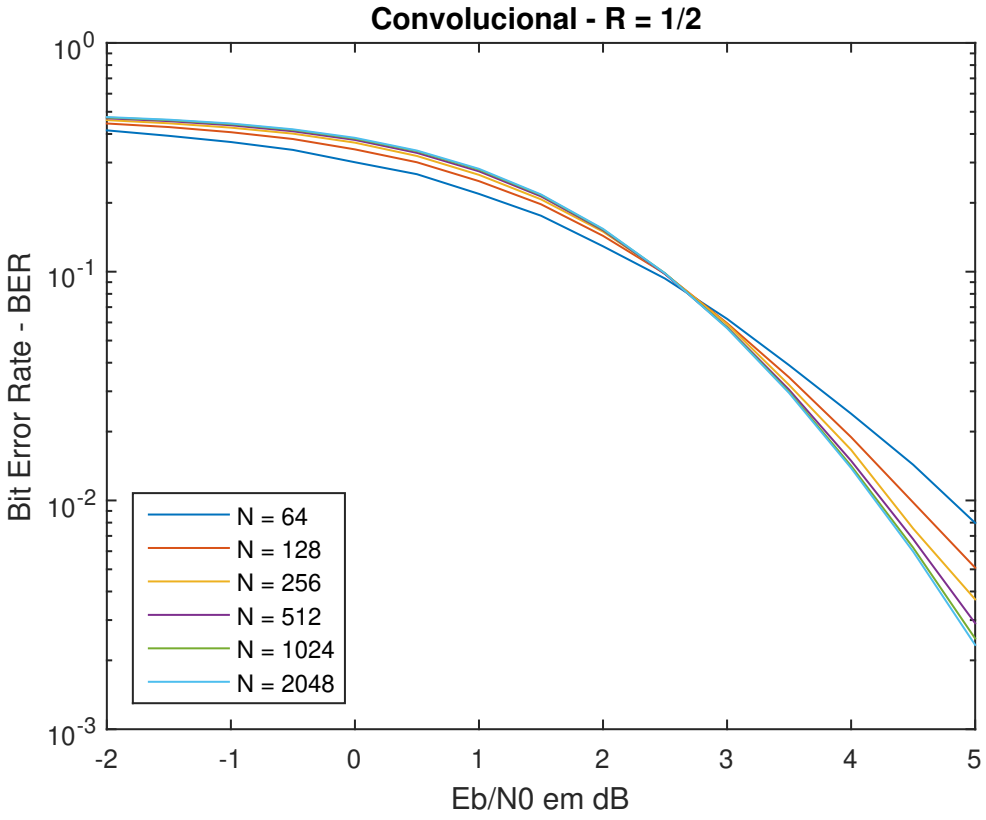
onde N' é a quantidade de simulações realizadas e N'_A o número de ocorrências de um determinado evento de interesse A. Sabendo que a estimativa leva em consideração a Lei dos Grandes Números, resultados precisos podem ser tomados ao simular o experimento um elevado número de vezes e fazendo N' tender ao infinito, o resultado torna-se preciso.

Para as simulações realizadas, a fim de obter um resultado representativo estatisticamente cada ponto na curva tem 10^5 realizações.

O simulador implementado gera k bits aleatórios, que passam pelos codificadores (convolucional e polar), ambos com taxa $R = k/N$ = 1/2. Após uma mensagem **u** ser codificada em **c**, ela é inserida em um modulador BPSK (*Binary Phase Shift Keying*), que foi escolhido pois os bits serão transmitidos um a um e este modulador pode ser usado de forma simples para esta aplicação, posteriormente a mensagem é transmitida por um canal que irá inserir ruído AWGN. Uma mensagem **r** é recebida, demodulada e decodificada em uma mensagem $\hat{\mathbf{u}}$. Para a simulação da BER, a mensagem $\hat{\mathbf{u}}$ que foi decodificada será comparada com a palavra-código **u** transmitida, para contabilizar, caso sejam diferentes, a ocorrência de erros de decodificação.

Para a implementação dos códigos convolucionais, utilizamos o código convolucional $(2, 1, 7)$ que possui máquina de estados apresentada na Figura 9. O codificador utilizado foi baseado no diagrama de treliça e o decodificador no algortimo de Viterbi. A transmissão de mensagens para diferentes comprimentos de palavras foram realizadas, com taxa de transmissão fixa $R = 1/2$ e os valores de E_b/N_0 foram variados de -2 a 5 dB, com a densidade de potência do ruído fixo N_0 = 2. Na Figura 39 encontram-se os resultados obtidos.

Figura 39 – Simulação da BER para códigos convolucionais com (N,k) = (64,32; 128,64; 256,128; 512,256; 1024,512; 2048,1024)



Fonte: Próprio autor.

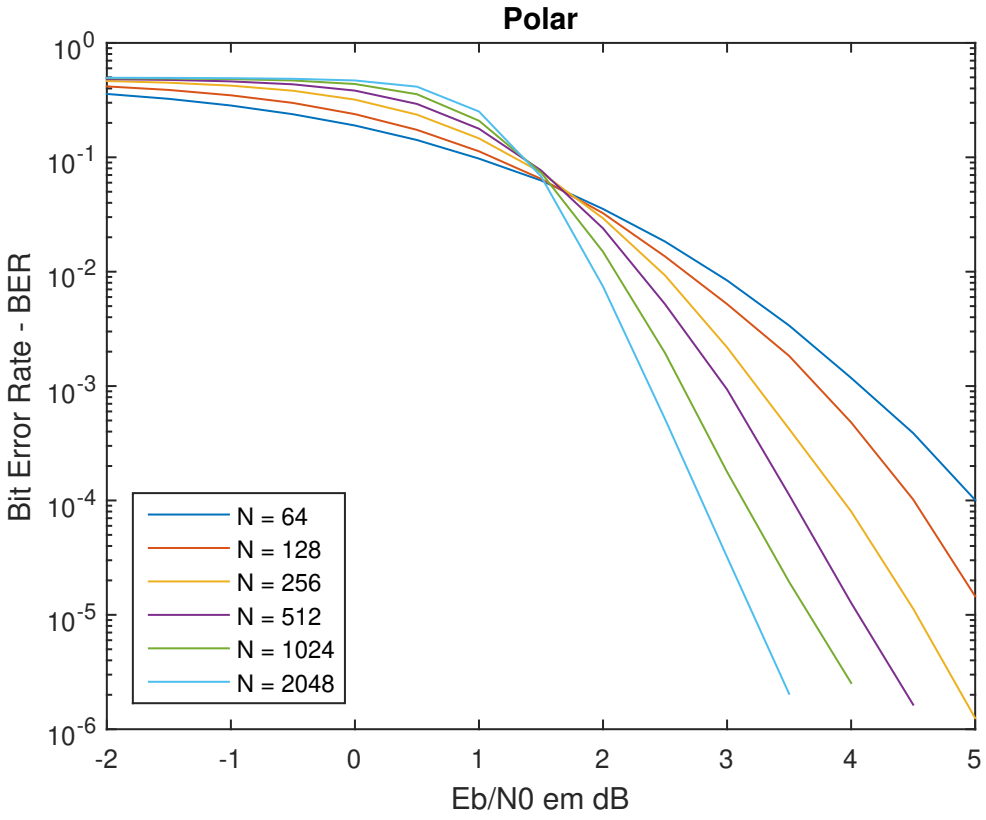
Pode-se observar que as curvas geradas não possuem grande alteração com a variação do comprimento da palavra código transmitida, entretanto, caso a potência de transmissão seja maior, o desempenho do código em correção e detecção de erro aumenta. Existe um ponto interessante nas curvas obtidas, quando E_b/N_0 é aproximadamente 2.5 dB, independente do tamanho da palavra transmistida o código terá o mesmo desempenho em correção e detecção de erros.

Após a implementação dos códigos convolucionais, implementou-se os códigos polares. Tanto o codificador, quanto o decodificador SC implementado foi baseado em [VANGALA 2018]. Na implementação, a transmissão de mensagens para diferentes comprimentos de palavras também foram realizadas, com taxa de transmissão fixa $R = 1/2$ e os valores de E_b/N_0 variados de -2 a 5 dB, com a densidade de potência do ruído fixo N_0 = 2. Na Figura 40 encontram-se os resultados obtidos.

Podemos observar que os códigos polares apresentam menores valores de BER conforme o comprimento da palavra código aumenta, pois as curvas convergem rapidamente para valores de BER de 10^{-6}. Quando o comprimento das palavras código é elevado, menor é o valor de E_b/N_0 necessário para atingir a mesma BER de palavras menores, o que representa um ganho energético na transmissão de palavras maiores.

É válido ressaltar que, a partir da polarização do canal que é utilizada na codificação e de acordo com a teoria descrita na Seção 4.1, para os valores de N alto, ou seja, N tendendo ao infinito, os canais polarizados tenderão para os extremos correspondentes a 0 ou 1. Deste modo, quanto maior os valores de N mais confiável será a transmissão.

Figura 40 – Simulação da BER para códigos polares com (N,k) = (64,32; 128,64; 256,128; 512,256; 1024,512; 2048,1024)



Fonte: Próprio autor.

Um ponto interessante na Figura 40 encontra-se quando E_b/N_0 é aproximadamente 1.8 dB, pois neste ponto, independete do comprimento da palavra transmitida, este código terá a mesma BER.

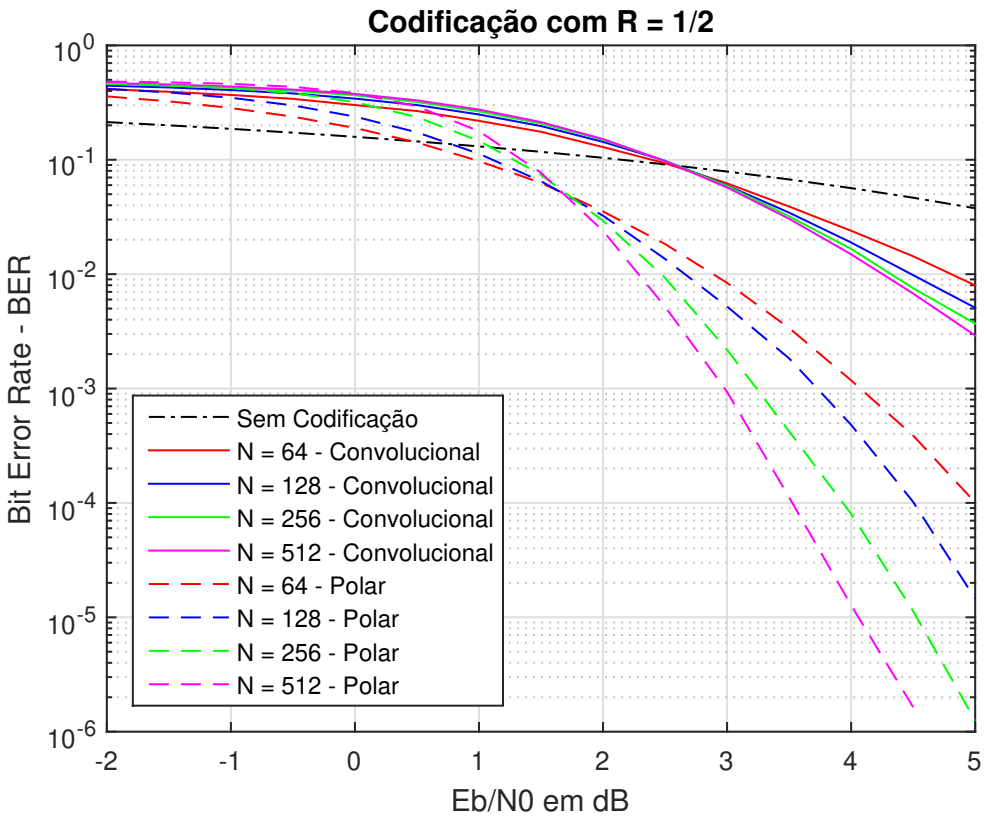
Com a implementação dos códigos convolucionais e polares, pôde-se comparar o desempenho em cenários URC com transmissão de pacotes curtos previsto nos sistemas 5G. Estes cenários possuem algumas métricas necessárias, como latência de 1 ms a 10 ms dependendo a aplicação e BER $\leqslant 10^{-6}$, [POPOVSKI 2015, LEMA et al. 2017].

A latência prevista nestes cenários levam em consideração o tempo necessário para a transmissão fim a fim da mensagem, as simulações realizadas analisam somente o tempo de processamento que é o tempo em que as codificações tem influência, é válido ressaltar que este tempo de processamento depende das especificações da máquina onde as simulações foram realizadas. A máquina utilizada tem um processador AMD Ryzen 3 1200 Quad-Core Processor 3.10 GHz e 8 GB de RAM. Para simular a transmissão de pacotes curtos previsto neste tipo de cenário, adota-se o comprimento da palavra-código enviada de no máximo 512 bits, [SYBIS et al. 2016, DURISI, KOCH e POPOVSKI 2016].

Assim, serão realizadas três análises: a comparação da BER, latência e *throughput* das duas codificações. Como anteriormente, para a simulação da BER, a mensagem $\hat{\mathbf{u}}$ que foi decodificada será comparada com a palavra-código $\mathbf{u}$ transmitida, para contabilizar, caso sejam diferentes, a ocorrência de erros de decodificação.

Podemos observar na Figura 41 os resultados obtidos. As linhas tracejadas representam os códigos polares e as linhas continuas os códigos convolucionais. Como previsto, comparando as BER's

Figura 41 – BER com taxa 1/2 para (N,k) = (64,32; 128,64; 256,128; 512,256)



Fonte: Próprio autor.

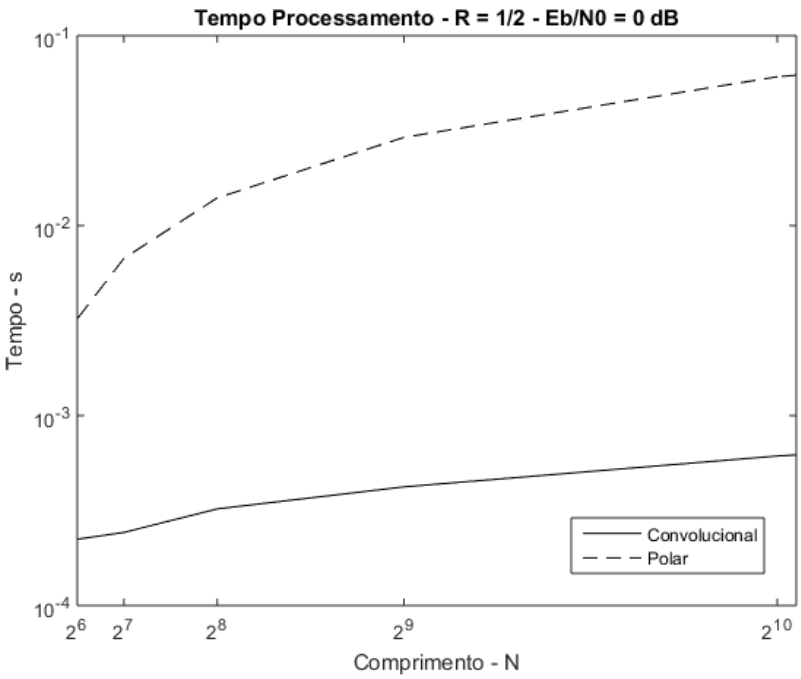
encontradas, os códigos polares possuem um desempenho melhor que os códigos convolucionais, que melhora conforme o comprimento da palavra transmitida aumenta, atingindo a métrica de BER $\leqslant 10^{-6}$ com uma potência de transmissão cada vez menor. Já os códigos convolucionais possuem um desempenho similar, independente do comprimento da palavra e para os parâmetros utilizados não atingem a métrica esperada. Deste modo, na comparação da BER, a utilização dos códigos convolucionais não seriam interessantes para a transmissão de pacotes curtos em sistemas 5G no canal simulado.

A segunda análise feita, tem como objetivo avaliar a latência de processamento destes códigos, deste modo, uma nova simulação contabilizando o tempo necessário para codificar e decodificar uma mensagem de interesse foi realizada. Podemos observar na Figura 42 os resultados obtidos com as simulações, as linhas contínuas representam os códigos convolucionais, as tracejadas os códigos polares. Foram realizadas as simulações variando-se E_b/N_0 de -2 a 5 dB, entrentanto constatou-se que este parâmetro não afetava o tempo de processamento, deste modo, os resultados apresentados são para E_b/N_0 = 0 dB.

Podemos observar que os códigos convolucionais possuem uma latência de processamento bem baixa, menor que 1 ms. Já os códigos polares possuem uma latência de processamento muito alta, o que não é interessante, pois somente o tempo de processamento já supera o limite de 10 ms do cenário.

Vale ressaltar, que o maior tempo de processamento dos códigos polares, encontra-se na decodificação, deste modo, os códigos polares, utilizando a decodificação SC, não é viável para estes cenários em estudo.

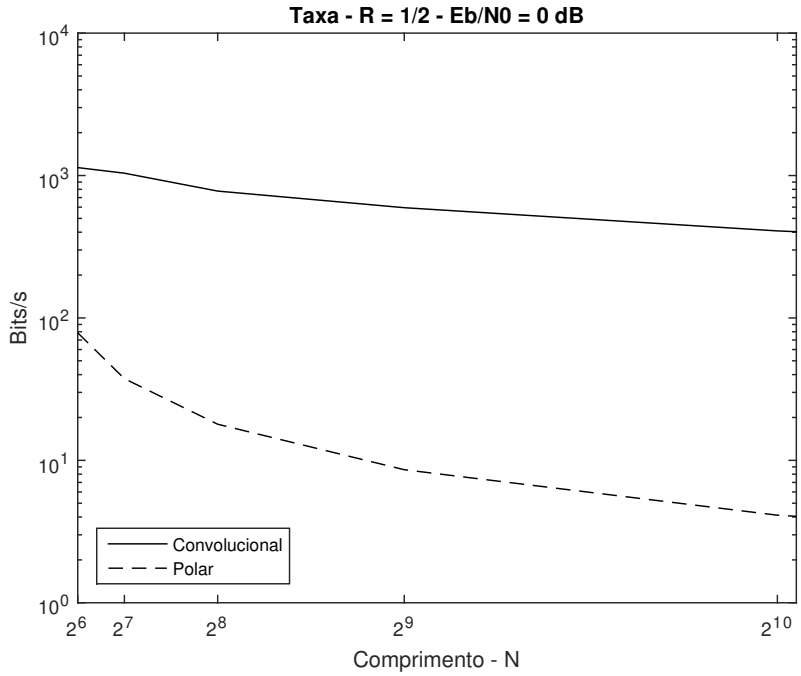
Figura 42 – Latência de processamento com taxa 1/2



Fonte: Próprio autor.

A terceira análise realizada foi com relação ao *throughput*, este parâmetro é muito importante, pois com ele é possível observar a quantidade de bits de informação que são decodificados por segundo pelo decodificador, [LEROUX et al. 2011, SARKIS e GROSS 2013b]. Deste modo, novas simulações foram realizadas, utilizando os dados coletados com a Figura 42 e a Equação 4.41, gerando os resultados apresentados na Figura 43.

Figura 43 – *Throughput* com taxa 1/2

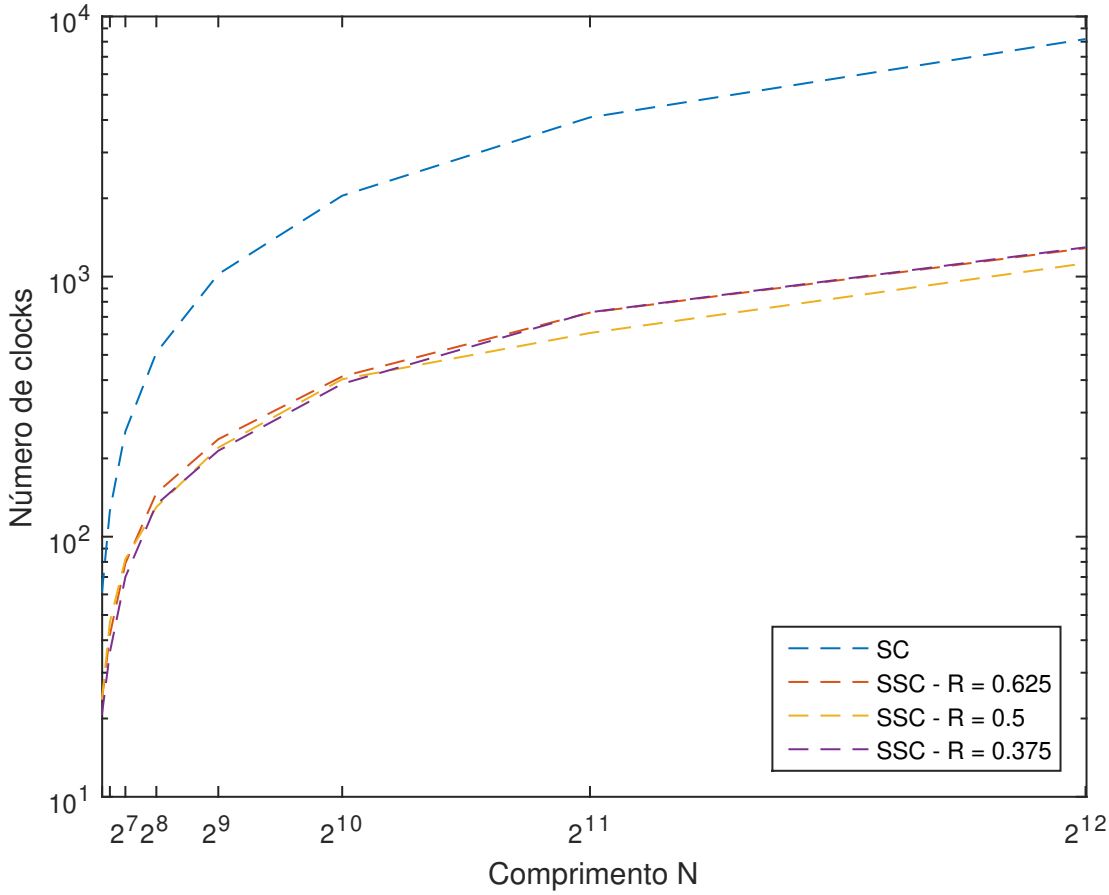


Fonte: Próprio autor.

Novamente, as linhas contínuas representam os códigos convolucionais e as tracejadas os códigos polares. Podemos observar que os códigos convolucionais conseguem decodificar mais bits por segundo que os códigos polares, isto ocorre pelo fato da latência de processamento dos códigos convolucionais ser menor, assim estes códigos possuem mais tempo para processar bits de informação. Além disso, quanto maior a palavra enviada pior será o desempenho dos códigos polares, isto ocorre novamente pela latência de processamento destes códigos, pois com o aumento da palavra-código a latência também é maior. Se compararmos um comprimento de palavra de 64 com 256 bits, o *throughput* decaí quase pela metade.

Um método para melhorar a latência dos códigos polares é a utilização de outro método de decodificação, visto que a decodificação SC demanda de maior tempo de processamento pela necessidade de efetuar todas as etapas da decodificação mesmo conhecendo a posição e os valores dos bits congelados. Assim, realizou-se uma comparação entre os dois decodificadores SC e SSC. Implementando as Equações 4.40 e 4.42 foi possível comparar a quantidade de *clocks* necessários para decodificar uma mesma mensagem de interesse. Podemos observar na Figura 44 os resultados obtidos.

Figura 44 – *Clocks* necessários para decodificar diferentes comprimentos de palavras



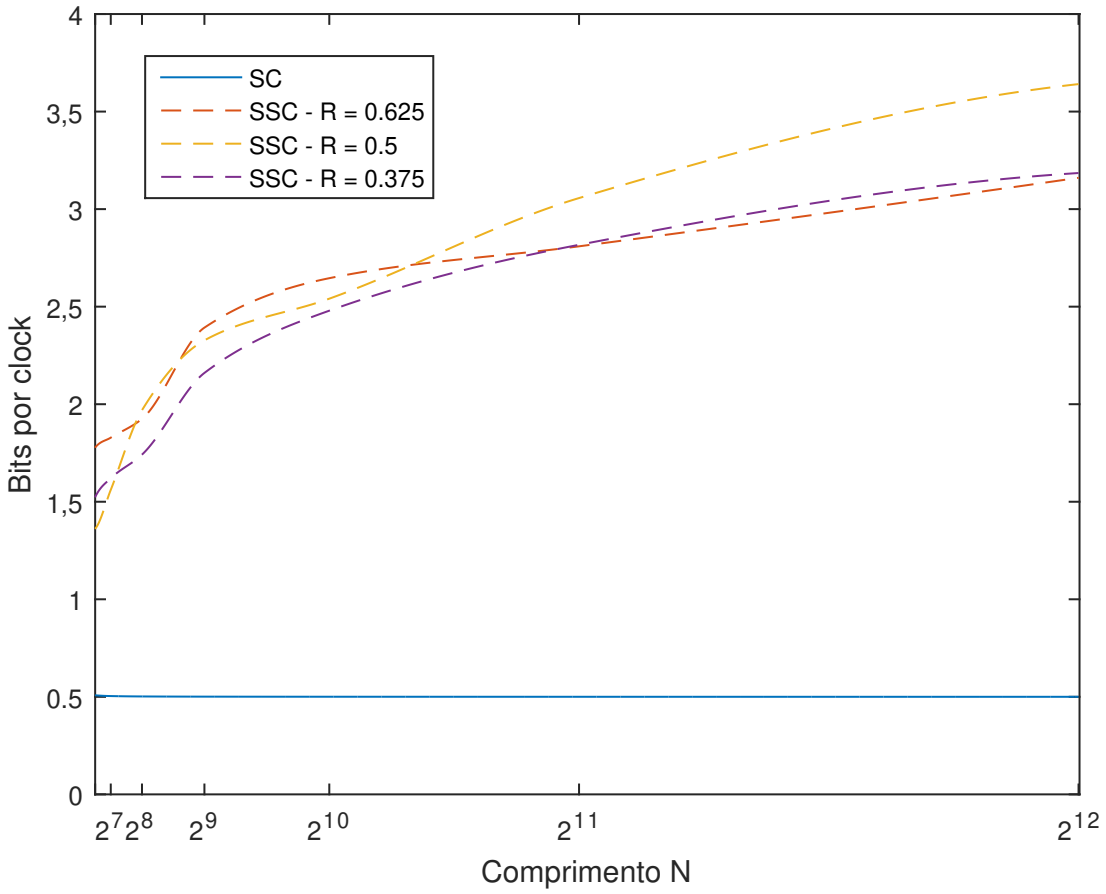
Fonte: Próprio autor.

Como vimos, a quantidade de *clocks* necessários para a decodificação SSC é bem menor se comparada com a decodificação SC e diminui pela metade para palavras-código de comprimento N pequeno e em até 5x para palavras-código de comprimentos maiores como $N = 1024$. Comparando

as diferentes taxas de codificação para a decodificação SSC, constata-se que não há uma variação acentuada entre as curvas, concluindo que a taxa de codificação possui pouca influência na quantidade de *clocks* para a decodificação.

Para comparar a quantidade de bits que são decodificados em média por ciclo de *clock* em cada decodificação, novas curvas foram geradas, como pode ser observado na Figura 45.

Figura 45 – Bits decodificados por *clock* para diferentes comprimento de palavra



Fonte: Próprio autor.

Podemos observar que a decodificação SC possui uma quantidade média de bits que são decodificados por segundo constante, independente do comprimento da palavra este decodificador decodifica aproximadamente 0.5 bit por *clock*. Já os decodificadores SSC, possuem um desempenho superior e que melhora conforme o comprimento da palavra-código aumenta, pois neste caso, mais bits são decodificados em média por ciclo de *clock*. Além disso, o decodificador SSC com taxa 0.5 possui um desempenho melhor se comparado com as outras duas taxas simuladas, pois decodifica até meio bit a mais para comprimentos maiores de palavra.

6 CONCLUSÕES

No presente trabalho foi apresentado um estudo de duas estratégias de codificação de canal: os códigos convolucionais e os códigos polares. Ambos os códigos são cogitados para serem utilizados nos sistemas 5G pela baixa complexidade de implementação e fácil adaptação da taxa de codificação. Foram apresentadas as formas de codificação e decodificação de ambos os códigos.

Os códigos convolucionais possuem várias formas de codificação, mas sua principal forma de decodificação é pelo algoritmo de Viterbi, que utiliza do diagrama de treliça, o qual também pode ser utilizado para codificar a mensagem. Observa-se que tanto a codificação quanto a decodificação deste código é realizada bit a bit.

Os códigos polares utilizam a técnica de polarização de canal para realizar a codificação. Com esta técnica é possível separar os canais de transmissão em dois grupos, os considerados bons e os ruins. Tal classificação é realizada através do cálculo de suas capacidades, assim, os bits de informação serão transmitidos nos canais bons para que não sejam degradados e bits conhecidos, chamados de congelados, nos canais ruins, que podem sofrer degradação do canal, pois o decodificador já possui os valores desses bits.

Para realizar a decodificação dos códigos polares duas formas foram apresentadas, a técnica por cancelamento sucessivo (SC) e a técnica por cancelamento sucessivo simplificada (SSC). Ambas as técnicas são decoficadores de máxima verossimilhança e as combinações realizadas pelo codificador são determinadas no decodificador pelo caminho reverso, para que deste modo, os bits sejam encontrados um a um e que ao determinar um bit, o próximo bit dependente do mesmo possa ser encontrado. A diferença entre as técnicas SC e SSC está principalmente no ganho de processamento, ou seja, diminuição na latência. Os códigos SSC determinam somente alguns nós, que são os caminhos necessários para determinar os bits de informação, pois os valores e as posições dos bits congelados já são conhecidos. Já o decodificador SC determina todos os nós do código, mesmo conhecendo os valores e as posições dos bits congelados.

Foi realizada uma implementação computacional da codificação e decodificação dos códigos. Pôde-se observar que as curvas de taxa de erro de bit geradas pelos códigos convolucionais não possuem grande variação com o comprimento da palavra transmitida, entretanto, caso a potência de transmissão seja maior, o desempenho do código em correção e detecção de erro aumenta. Já para os códigos polares, vimos que quanto maior o valor de N, menor é a taxa de erro de bit e para atingir uma mesma métrica é necessário uma menor relação sinal ruído, o que representa um ganho energético para palavras maiores.

Foram realizadas também comparações de ambos os códigos em cenários de transmissão de pacotes curtos em sistemas 5G, que prevêem transmissões de até 512 bits, com uma taxa de erro de bit $\leq 10^{-6}$ e latência ≤ 10 ms. Ao comparar-se os códigos observou-se que os códigos polares atingem a métrica estabelecida para taxa de erro, já os códigos convolucionais possuem um desempenho bem inferior, com valores de BER de até 10^{-3}. No quesito de latência foi observado que os códigos polares não possuem um desempenho bom, ultrapassando valores de 10 ms, já os códigos convolucionais possuem

um latência abaixo de 1 ms. Como o $throughput$, que é a quantidade de bits que são decodificados por segundo depende da latência de processamento, os códigos convolucionais se sobressaíram aos polares.

Esta grande diferença encontrada na latência nos códigos polares é dada principalmente pelo tipo de decodificação utilizado, no caso das simulações realizadas implementou-se um decodificador SC. Para comparar o desempenho de dois decodificadores polares foram realizadas algumas simulações para analisar a quantidade de *clocks* necessários nas decodificações SC e SSC para diferentes comprimentos N de palavra e observou-se que o decodificador SSC necessita de uma quantidade menor de *clocks* do processador para decodificar uma mesma mensagem, sendo assim, este decodificador possui um desempenho melhor em latência. Comparou-se também se a variação da taxa de codificação de um decodificador SSC têm impacto na latência ou na quantidade de bits decodificados por *clock* e constatou-se que não há grande variação destes parâmetros, mas o decodificador com taxa 0.5 possui o melhor desempenho.

Assim, é possível concluir que os códigos convolucionais não atingem a métrica de correção de erro, mas são códigos com baixa latência, uma forma de talvez melhorar seu desempenho é a utilização de códigos turbo, que são códigos derivados dos códigos convolucionais e possuem melhor correção e detecção de erro. Já os códigos polares atingem a métrica de correção de erro, mas a técnica de decodificação SC demanda muito tempo, tornando sua utilização inviável, o método encontrado para contornar este problema é a utilização de decodificadores mais rápidos como o SSC. Além disso, outras estratégias de decodificação de códigos polares veem sendo estudadas e podem melhorar ainda mais os requisitos de URC para sistemas 5G. Portanto, a codificação e decodificação de estratégias de codificação de canal é um tema com diversos problemas em aberto e com muitas aplicações práticas nas novas gerações de sistemas de comunicações móveis.

REFERÊNCIAS

5GPPP. *The 5G Infrastructure Public Private Partnership: the next generation of communication networks and services.* 2015. Disponível em: <https://5g-ppp.eu/wp-content/uploads/2015/02/5G-Vision-Brochure-v1.pdf>. Acesso em: 08 de Novembro de 2019.

ARIKAN, E. *Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels*. **IEEE Transactions on Information Theory**, v. 55, n. 7, p. 3051–3073, 2009.

COVER, T. M.; THOMAS, J. A. **Elements of Information Theory**. [S.l.]: Wiley Interscience, 2006.

DURISI, G.; KOCH, T.; POPOVSKI, P. *Toward massive, ultrareliable, and low-latency wireless communication with short packets*. **Proceedings of the IEEE**, v. 104, p. 1711–1726, 2016.

FORNEY, G. *Convolutional codes I: Algebraic structure*. **IEEE Transactions on Information Theory**, v. 16, p. 720 – 738, 1970.

HAYKIN, S. S. **Communication systems**. [S.l.]: Wiley Interscience, 2001.

HUIGOL, S. ***Channel Coding Techniques for 5G Using Polar Codes***. Dissertação (Mestrado) — The University of Texas at Arlington, 2017.

ISCAN, O.; LENTNER, D.; XU, W. *A Comparison of Channel Coding Schemes for 5G Short Message Transmission*. **IEEE Globecom Workshops (GC Wkshps)**, 2016.

JACOBINI, C.; LUGLI, P. **The Monte Carlo method for semiconductor device simulation**. [S.l.]: Springer Science & Business Media, 2012.

LEMA, M. A. et al. *Business Case and Technology Analysis for 5G Low Latency Applications*. **IEEE Access**, p. 5917 – 5935, 2017.

LEROUX, C. et al. *Hardware architectures for successive cancellation decoding of polar codes*. **IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**, 2011.

METIS. ***The Metis2020 project – laying the foundation of 5G***. 2012 – 2015. Disponível em: <https://www.metis2020.com>. Acesso em: 30 de Outubro de 2019.

METIS. ***Mobile and wireless communications enablers for twenty-twenty (2020) information society***. 2015 – 2017. Disponível em: <https://metis-ii.5g-ppp.eu>. Acesso em: 30 de Outubro de 2019.

NIU, K. et al. *Polar codes: Primary concepts and practical decoding algorithms*. **IEEE Communications Magazine**, v. 52, p. 192–203, 2014.

OLIVEIRA, R. M.; LAMARE, R. C. *Códigos Polares e Puncionamento Baseado em Polarizacao para Sistemas 5G*. **XXXV SIMPOSIO BRASILEIRO DE TELECOMUNICACOES E PROCESSAMENTO DE SINAIS**, p. 629–633, 2017.

OMURA, J. *On the Viterbi decoding algorithm*. **IEEE Transactions on Information Theory**, v. 15, p. 177–179, 1969.

POPOVSKI, P. *Ultra-reliable communication in 5G wireless systems*. **1st International Conference on 5G for Ubiquitous Connectivity**, 2015.

RAPPAPORT, T. S. **Comunicações sem fio: Princípios e Práticas**. [S.l.]: Pearson, 2008.

RYAN, W.; LIN, S. **Channel Codes: Classical and Modern**. [S.l.]: Cambridge University Press, 2009.

SARKIS, G.; GROSS, W. J. *Increasing the Throughput of Polar Decoders*. **IEEE Communications Letters**, v. 17, p. 725–728, 2013a.

SARKIS, G.; GROSS, W. J. *Polar codes for data storage applications*. **International Conference on Computing, Networking and Communications (ICNC)**, 2013b.

SHANNON, C. E. *A Mathematical Theory of Communication*. **The Bell System Technical Journal**, v. 27, p. 397–423 *and* 623–656, 1948. *Reprinted in*: C.E.Shannon and W.Weaver, eds., *A Mathematical Theory of Communication*, (Univ. of Illinois Press, Urbana, Illinois, (1963)).

SYBIS, M. et al. *Channel coding for ultra-reliable low-latency communication in 5G systems.* **IEEE 84th Vehicular Technology Conference (VTC-Fall)**, 2016.

TAHIR, B.; SCHWARZ, S.; RUPP, M. *BER comparison between Convolutional, Turbo, LDPC, and Polar codes*. **24th International Conference on Telecommunications (ICT)**, p. 1–7, 2017.

VANGALA, H. ***Polar Codes***. 2018. Disponível em: <http://www.polarcodes.com/>.

ZHANG, C.; PARHI, K. K. *Latency Analysis and Architecture Design of Simplified SC Polar Decoders*. **IEEE Transactions on Circuits and Systems II: Express Briefs**, v. 61, p. 115–119, 2014.