REDUCED-COMPLEXITY REINFORCEMENT LEARNING-BASED POLAR CODE CONSTRUCTION

by
Ezgi Oral
B.S., Electronics and Communication Engineering, Istanbul Technical University, 2019

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical and Electronics Engineering
Boğaziçi University
2022

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor Prof. Ali Emre Pusane for his support and mentoring. I would also like to thank my thesis co-supervisor Prof. İbrahim Altunbaş for his guidance throughout this study. Besides my advisors, I would like to thank Oğuzhan Aydoğan for his help and support.

Most importantly, I am thankful to my family for their endless support and encouragement. If it were not for them, I would not be who and where I am today.

ABSTRACT

REDUCED-COMPLEXITY REINFORCEMENT LEARNING-BASED POLAR CODE CONSTRUCTION

Due to the increasing number of users in the communication systems, efficient spectrum usage, high-speed data transfer, and better error performance became a necessity. Therefore, it is aimed to design error control codes that have low error rates and have a capacity close to the Shannon's limit. As a consequence of Erdal Arıkan's work, polar codes, a coding technique that is theoretically proven to achieve Shannon's limit, are introduced. After polar codes are used in fifth-generation new radio (5G NR) technology, more studies are done about the decoding of polar codes and the polar code construction. The scope of the thesis is on reinforcement learning-based polar code construction. Initially, the preliminaries of polar codes and reinforcement learning are given. Then, several reinforcement learning-based polar code construction methods are introduced. It is shown that a reinforcement learning-based method found in the literature performs weakly for long block lengths due to high complexity and therefore, two new methods are introduced to reduce the complexity. First, a method that groups the channels into clusters and predetermines some channels as frozen or information is proposed. For long block lengths, it had a better performance than the one proposed in the literature, but its performance was unsatisfactory for much longer block lengths. To further reduce the complexity, neighbor dependency is introduced to the first method. It is shown that the performance of the neighbor dependent method is better than both methods and its performance is satisfactory for longer block lengths.

ÖZET

DÜŞÜK KARMAŞIKLIKLI PEKİŞTİRMELİ ÖĞRENME TABANLI KUTUPSAL KOD TASARIMI

Haberleşme sistemlerinden faydalanan kullanıcı sayısının artmasıyla birlikte spektrumun etkin kullanılması, yüksek hızlarda veri iletimi ve yüksek hata başarımlarına sahip sistemler tasarlamak bir gereklilik haline gelmiştir. Bu sebeple yüksek hata başarımına sahip ve Shannon sınırına yakın kodlar tasarlamak amaçlanmış ve bu alanda literatürde birçok çalışma yapılmıştır. Erdal Arıkan'ın bu alandaki çalışmaları sonucunda teorik olarak Shannon limitine ulaştığı ispatlanan bir kodlama türü olan kutupsal (polar) kodlar ortaya çıkmıştır. Kutupsal kodların 5G NR teknolojisinde kullanılmaya başlanmasıyla birlikte kutupsal kodların çözümü ve tasarımı hakkında yapılan çalışmalarda artış olmuştur. Bu tez kapsamında pekiştirmeli öğrenme kullanarak kutupsal kod tasarımı üzerine çalışılmıştır. İlk olarak kutupsal kodlar ve pekiştirmeli öğrenme hakkında temel bilgiler verilmiş olup ardından kutupsal kod tasarımını pekiştirmeli öğrenme kullanarak yapan yöntemler tanıtılmıştır. Literatürde bulunan pekiştirmeli öğrenme kullanan bir kutupsal kod tasarım yönteminin uzun kod blokları için artan karmaşıklık sebebiyle hata başarımının yetersiz kaldığı gösterilmiş ve karmaşıklığı azaltmak için iki yeni yöntem önerilmiştir. Öncelikle kanalları kümelere ayırıp önceden bazı kanalları dondurulmuş (frozen) veya bilgi (information) olarak belirleyen bir yöntem önerilmiştir. Bu yöntemin sağladığı başarım her ne kadar uzun kod blokları için literatürdeki yöntemden daha iyi olsa da daha uzun kod blokları için yetersiz kalmaktadır. Karmaşıklığı daha da düşürmek için ilk önerilen yönteme komşu kanallara dayalı bir eleme yöntemi eklenmiş ve uzun kod blokları için yüksek hata başarımları elde edilmiştir.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii
ABSTRACT iv
ÖZET v
LIST OF FIGURES viii
LIST OF TABLES x
LIST OF SYMBOLS xi
LIST OF ACRONYMS/ABBREVIATIONS xii
1. INTRODUCTION 1
1.1. Purpose of the Thesis 1
1.2. Literature Review 1
1.3. Thesis Outline 3
2. POLAR CODES 5
2.1. Communication Channel Models 6
2.1.1. Binary Discrete Memoryless Channel (B-DMC) 6
2.1.1.1. Binary Symmetric Channel (BSC) 7
2.1.1.2. Binary Erasure Channel (BEC) 9
2.1.2. Additive White Gaussian Noise (AWGN) Channel 10
2.2. Channel Polarization 11
2.3. Polar Code Construction Methods 16
2.3.1. Bhattacharyya Method 16
2.3.2. Monte Carlo Method 17
2.4. Decoding of Polar Codes 18
2.4.1. Successive Cancellation (SC) Decoder 19
2.4.2. Successive Cancellation List (SCL) Decoder 21
2.4.3. CRC-Aided (CA) Decoding 21
3. REINFORCEMENT LEARNING (RL) 23
3.1. Definitions in RL 26
3.2. Fundamentals of RL 27

3.3. Markov Decision Process 29
3.4. RL Algorithms 31
3.4.1. Monte Carlo 32
3.4.2. Q-Learning 32
3.4.3. State Action Reward State Action (SARSA) 33
3.4.4. Watkins's Q(λ) 34
3.4.5. SARSA(λ) 35
3.5. Deep RL 42
4. POLAR CODE CONSTRUCTION WITH REINFORCEMENT LEARNING 44
4.1. Maze Approach 44
4.2. Reduced Cluster-Based Maze Approach 49
4.3. Reduced Cluster-Based Maze Approach with Neighbor Dependency . . 56
5. CONCLUSION 60
REFERENCES 62
APPENDIX A: CHANNEL INDICES PER CLUSTER WHEN $N = 128$ AND $N = 256$ 67

LIST OF FIGURES

Figure 2.1. Block diagram for a communication system. 6

Figure 2.2. Binary symmetric channel. 8

Figure 2.3. Binary erasure channel. 9

Figure 2.4. AWGN channel. 10

Figure 2.5. Symmetric capacity for a BEC when $N = 1024$ and $\varepsilon = \{0.3, 0.7\}$. 12

Figure 2.6. Channel combination (W_2). 12

Figure 2.7. Channel combination (W_{16}). 14

Figure 2.8. Channel polarization evaluation for a BEC with $\varepsilon = 0.3$. 15

Figure 2.9. SC decoding tree for $N = 16$. 20

Figure 3.1. The relationship between ML and its subcategories. 24

Figure 3.2. A maze game. 27

Figure 3.3. RL mechanism. 28

Figure 3.4. An example environment for SARSA(λ) algorithm. 36

Figure 4.1. Environment of the maze game for $\mathbb{P}(16, 8)$. 45

Figure 4.2. Evaluation of learning after 100 episodes. 47

Figure 4.3. Evaluation of learning after 2000 episodes. 47

Figure 4.4. FER performance of the maze approach. 49

Figure 4.5. Environment of the reduced cluster-based maze game for $\mathbb{P}(16, 8)$. 50

Figure 4.6. Environment of the reduced cluster-based maze game for $\mathbb{P}(16, 12)$. 51

Figure 4.7. FER performance of the reduced cluster-based maze approach. . . 55

Figure 4.8. Total number of episodes needed for convergence for several block lengths. 56

Figure 4.9. FER performance of the reduced cluster-based maze approach with neighbor dependency. 59

LIST OF TABLES

Table 2.1. Number of errors per channel for $N = 16$ at 1 dB. 18

Table 3.1. $Q(s, a)$ and $E(s, a)$ after Step 1. 38

Table 3.2. $Q(s, a)$ and $E(s, a)$ after Step 2. 39

Table 3.3. $Q(s, a)$ and $E(s, a)$ after Step 3. 41

Table 3.4. $Q(s, a)$ and $E(s, a)$ at the end of the first episode (after Step 5). . 42

Table 4.1. Parameter set of the maze game for various block lengths. 48

Table 4.2. Channel indices per cluster when $N = 16$. 52

Table 4.3. Channel indices per cluster when $N = 64$. 53

Table 4.4. Number of channel indices in every cluster for several code lengths. 54

Table 4.5. Channel indices per cluster when $N = 512$ (first 10 rows). 58

Table A.1. Channel indices per cluster when $N = 128$. 67

Table A.2. Channel indices per cluster when $N = 256$. 69

LIST OF SYMBOLS

a	Action
a'	Next action
A	Action set
$\mathcal{A}$	The set of information channel indices
$\mathcal{A}_c$	The set of frozen channel indices
$\mathcal{C}_i$	Cluster i
G_t	Total discounted reward at t
K	Information bit length
N	Code length
$\mathcal{P}$	State transition matrix
R	Code rate
$\mathcal{R}$	Reward
s	State
s'	Next state
S	State set
t	Time step
$q_\pi(s, a)$	Value of state-action pair under π
$Q(s, a)$	Estimate of $q_\pi(s, a)$
$v_\pi(s)$	Value of s under π
W	Channel
α	Learning rate
ϵ	Exploration rate
ε	Erasure probability
γ	Discount rate
λ	Eligibility decay factor
π	Policy

LIST OF ACRONYMS/ABBREVIATIONS

5G	Fifth-Generation
AI	Artificial Intelligence
AWGN	Additive White Gaussian Noise
B-DMC	Binary Discrete Memoryless Channel
BEC	Binary Erasure Channel
BPSK	Binary Phase Shift Keying
BSC	Binary Symmetric Channel
CA	CRC-Aided
CRC	Cyclic Redundancy Check
dB	Decibel
DMC	Discrete Memoryless Channel
FER	Frame Error Rate
LDPC	Low-Density Parity-Check
LLR	Log-Likelihood Ratio
MDP	Markov Decision Process
ML	Machine Learning
NR	New Radio
PM	Path Metric
RL	Reinforcement Learning
SARSA	State Action Reward State Action
SC	Successive Cancellation
SCL	Successive Cancellation List
SNR	Signal to Noise Ratio

1. INTRODUCTION

1.1. Purpose of the Thesis

In today's communication systems, there is a demand for higher data rates and better error performance. New error control codes are developed over time to meet these demands. Since polar codes have achieved the Shannon's limit, they gained popularity over time. Many studies are found in the literature to further increase the error performance of polar codes. These improvements are being done both in terms of polar code construction and decoding of the polar codes.

In this thesis, the main focus is on polar code construction. A good performing polar code construction method that uses reinforcement learning fails to construct reliable codes for longer block lengths. This thesis aims to construct polar codes that have low error rate performance for longer block lengths. Therefore, to further improve the performance of the reinforcement learning-based polar code construction method, new methods are proposed.

1.2. Literature Review

Polar codes are invented by Erdal Arıkan and they are proven to achieve the Shannon's channel capacity for binary discrete memoryless channels [1]. They have a similar structure to Reed-Muller codes however, they have better error performance than Reed-Muller codes [2].

Throughout the years, polar code construction evolved and the error performance of the polar codes improved. The first proposed construction technique was based on the Bhattacharya parameters with a Monte Carlo approach [1]. Afterward, several different polar code construction methods like density evaluation [3–5] and Gaussian approximation [6, 7] were proposed for a successive cancellation decoder. Furthermore,

the construction of polar codes was also considered for non-binary discrete input channels in [8, 9].

The decoding method used for polar codes also affects the error correction performance. When polar codes were first introduced, a successive cancellation decoder was proposed [1]. However, the error correction performance of the successive cancellation decoder was not sufficient for shorter block lengths. Therefore, new decoding methods were proposed to obtain satisfactory error performance. In [10], a belief propagation decoding was introduced. It is reported that the belief propagation based decoder has a better performance than the successive cancellation decoder [10, 11]. In addition to belief propagation, linear programming decoders were also proposed and they outperformed successive cancellation decoders [12]. Later, an enhanced version of the successive cancellation decoder, the successive cancellation list decoder, was introduced in [13]. It was observed that as the list size increases, the error performance of the polar decoder gets better. To further improve the performance of the successive cancellation decoder, cyclic redundancy check (CRC) was included to the successive cancellation list decoder [14]. It was proven that the presence of CRC bits improves the performance of the successive cancellation list decoder [13, 15]. Lastly, a stack decoder was presented in [16]. The stack decoder had a similar performance as the successive cancellation list decoder.

Studies in the literature on polar codes are designed for lower-order modulations and for binary discrete memoryless channels or additive white Gaussian noise channels. In [17] and [18] polar codes are examined for higher-order modulations. A modulation strategy, displacement of balanced modulation, that reduces the complexity of the Gaussian approximation is proposed in [17]. In [18], the design and analysis of polar coded modulation schemes are introduced. As the satellite communication gained popularity over the last decade, polar codes are being proposed for satellite communications as well. The performance of polar codes for satellite communication are studied in [19–22].

Over time, the computational capabilities of computers have increased and the machine learning techniques have developed. Machine learning is now being used in many fields as well as in error control coding. There are several studies about the polar codes that use machine learning methods. For example, in [23] deep learning is used for channel decoding. The polar code decoding for shorter block lengths is studied in [24]. Also, belief propagation decoding of polar codes by using reinforcement learning is studied in [25, 26]. In [27] genetic algorithm and in [28] reinforcement learning is used for polar code construction. Additionally, in [29], the polar code construction problem was formulated as a maze game and solved by using reinforcement learning. Decoding of polar codes for Rayleigh fading channel are examined in [30] by using deep neural networks.

1.3. Thesis Outline

The next chapter gives the essential to understand polar codes. First, the place of polar codes in the error control coding and their importance are explained. Then several fundamental channel models like binary-discrete memoryless channels are explained. Chapter 2 mainly focuses on channel polarization, polar code construction, and decoding of polar codes. Main polar code construction methods and several fundamental decoding techniques are introduced. Furthermore, the effect of using CRC bits on the performance of polar codes are given.

Chapter 3 provides the necessary information about reinforcement learning. First, the difference between reinforcement learning and machine learning is explained. Then, fundamentals of reinforcement learning are given and key parameters are defined. Several commonly used reinforcement learning algorithms are introduced. Additionally, the calculation steps of a reinforcement learning algorithm are illustrated with an example.

In Chapter 4, several methods that combine the polar code construction problem with reinforcement learning methods are presented. Initially, a method found in the

literature that uses a maze game to construct polar codes is explained in detail. To deal with the complexity problem that occurs when the block length increases, two new approaches are proposed. First, a reduced cluster-based maze approach is presented. Then, to construct polar codes with higher block lengths, a neighbor dependent method is introduced. Lastly, the performance of the proposed approaches are compared to the performance of polar code construction methods found in the literature.

Finally, in Chapter 5, the conclusion of the thesis is provided. Also, future work recommendations to increase the error performance and handle the complexity problem are given.

2. POLAR CODES

Coding approaches can be grouped into two main categories: source coding and error control coding. In source coding, the goal is to reduce the size of transmitted data by compressing it. For example, Huffman coding and Shannon's source coding algorithms can be typically used for data compression. In error control coding, the main purpose is to protect the data against errors. So, unlike source coding, error control codes add redundant bits to the messages to assure reliable communication. For instance, when a message is sent from the transmitter, the signal that carries the message goes through a channel and is then delivered to the receiver. In the channel, the signal is affected by noise, fading, interference, etc. These effects cause bit errors in the received message. To detect and correct errors, error control coding techniques are applied.

Error control codes are also divided into two main categories: block codes and convolutional codes [31]. In block codes, the message is divided into fixed-length message blocks and then these message blocks are converted into codewords that include redundancy. On the other hand, in convolutional codes, the sequence of the information bits is maintained and continuously encoded. Redundant bits are included by using linear shift registers. Reed-Muller codes, low-density parity-check (LDPC) codes, and polar codes are some examples of linear block codes.

Lately, the number of users that benefit from wireless communication technologies has increased tremendously. To fulfill users' demands, higher data rates, higher capacity, and low error rates are required. New error control codes are designed over time to meet these requirements and ensure reliable data transmission. As coding techniques improved, their performance got closer to Shannon's limit and even theoretically achieved it. For example, LDPC codes' performance is very close to Shannon's limit and polar codes have achieved Shannon's limit. Today, in 5G NR technology, LDPC and polar codes are used in traffic and control channels, respectively [32].

The performance of the communication system depends on the medium that the signal is transmitted over. Therefore, the quality of the received messages/signals changes depending on the channel model. In the following section, fundamental and commonly used communication channel models are examined.

2.1. Communication Channel Models

Channel in information theory is defined as the medium that is used to send a message/signal from a transmitter to a receiver [33]. In Figure 2.1, a basic block diagram for a communication system is shown. In a digital communication system, typically a message is first encoded and then it is modulated to build a waveform. During the transmission, the waveform encounters distortions in the channel. The effect of the distortions on the signal changes with the channel model. In the following subsections, several fundamental communication channel models are introduced and the channel capacities of those channels are given.

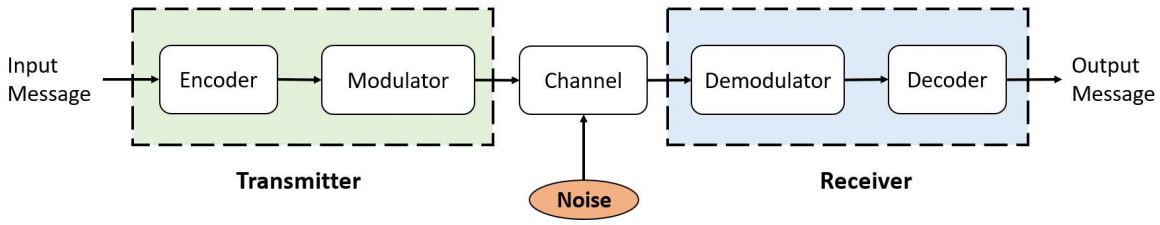

Figure 2.1. Block diagram for a communication system.

2.1.1. Binary Discrete Memoryless Channel (B-DMC)

A discrete channel has a discrete input alphabet $X = \{x_1, x_2, \cdots, x_j\}$ and a discrete output alphabet $Y = \{y_1, y_2, \cdots, y_k\}$. When the probability distribution is independent from previous inputs, the discrete channel is considered as memoryless [34]. The transition probability distribution of a discrete memoryless channel (DMC) is $P(y|x) = P(Y = y \mid X = x)$. In a binary DMC (B-DMC), the input alphabet is finite

and discrete. Binary symmetric channel (BSC) and binary erasure channel (BEC) are the most commonly utilized examples of B-DMCs.

The maximum information rate that can be reliably transmitted over channels may differ. The channel capacity is defined as the maximum possible data rate that a channel can have. For DMCs, the channel capacity is defined as

$$C = \max_{p(x)} I(X;Y), \tag{2.1}$$

where $I(X;Y)$ is the mutual information and $p(x)$ is the input distortion [34]. For a B-DMC, the channel W can be represented in terms of its input and output alphabets as $W : X \to Y$. The mutual information or the symmetric capacity, $I(W)$, is defined as

$$I(W) \triangleq \sum_{y\in Y}\sum_{x\in X} \frac{1}{2} P(y|x) \log \frac{P(y|x)}{\frac{1}{2}P(y|0) + \frac{1}{2}P(y|1)}. \tag{2.2}$$

$I(W)$ indicates the maximum rate for reliable communication across the channel. Note that $I(W)$ is equal to the Shannon capacity when W is symmetric. In addition to symmetric capacity, another channel parameter used for B-DMC is the Bhattacharya parameter and it is denoted as

$$Z(W) \triangleq \sum_{y\in Y} \sqrt{P(y|0)P(y|1)}. \tag{2.3}$$

$Z(W)$ defines the upper limit for maximum-likelihood decision error probability and is a metric for reliability [1].

2.1.1.1. Binary Symmetric Channel (BSC). BSC is commonly used in the coding theory. Since it is a binary channel, $X = \{0, 1\}$ and $Y = \{0, 1\}$. The schematic of a BSC is given in Figure 2.2. The probability of error in the channel is represented by ρ. In other words, ρ is the probability of receiving 1 while 0 is transmitted and receiving 0

while 1 is transmitted.

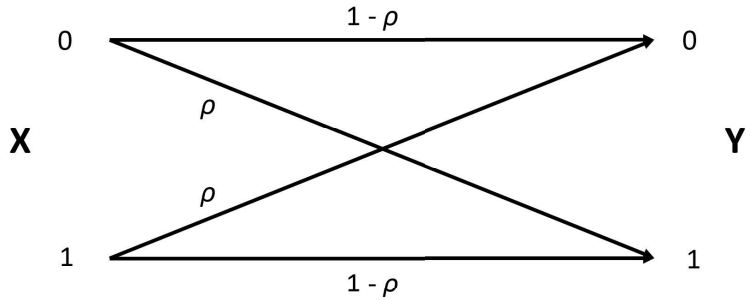

Figure 2.2. Binary symmetric channel.

For a BSC, the transition probabilities are given as

$$\begin{aligned} P(Y=0 \mid X=0) &= 1-\rho, \\ P(Y=0 \mid X=1) &= \rho, \\ P(Y=1 \mid X=0) &= \rho, \\ P(Y=1 \mid X=1) &= 1-\rho. \end{aligned}$$

The channel is symmetric because the columns and rows of the channel matrix

$$\begin{array}{c} \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{cc} y_1 & y_2 \end{array} \\ \begin{bmatrix} 1-\rho & \rho \\ \rho & 1-\rho \end{bmatrix} \end{array}$$

involve the same set of values [34]. The channel capacity of a BSC is given as

$$C = 1 - H(\rho), \tag{2.4}$$

where $H(\rho)$ is the marginal entropy and is defined as

$$H(\rho) = \rho \log_2 \left(\frac{1}{\rho}\right) + (1-\rho) \log_2 \left(\frac{1}{1-\rho}\right). \tag{2.5}$$

2.1.1.2. Binary Erasure Channel (BEC). Another commonly used channel model in the coding theory is BEC. In a BEC, if an error occurs during the transmission of the symbol, the receiver receives the erasure symbol, "?". Therefore, $X = \{0, 1\}$ and $Y = \{0, ?, 1\}$. The channel is called erasure because when it receives "?", the decision is erased. With the erasure, the receiver is protected from faulty transmissions. In Figure 2.3, the schematic of a BEC is shown. Here, the parameter ε is defined as the erasure probability.

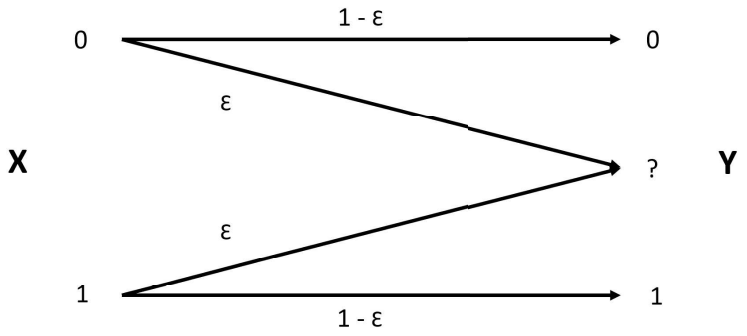

Figure 2.3. Binary erasure channel.

For a BEC, the transition probabilities are given as

$$\begin{aligned} P(Y = 0 \mid X = 0) &= 1 - \varepsilon, \\ P(Y = 0 \mid X = 1) &= 0, \\ P(Y = ? \mid X = 0) &= \varepsilon, \\ P(Y = ? \mid X = 1) &= \varepsilon, \\ P(Y = 1 \mid X = 0) &= 0, \\ P(Y = 1 \mid X = 1) &= 1 - \varepsilon. \end{aligned}$$

The channel capacity of a BEC is

$$C = 1 - \varepsilon. \tag{2.6}$$

2.1.2. Additive White Gaussian Noise (AWGN) Channel

AWGN is frequently used in communication theory. When a signal, *s(t)*, goes through an AWGN channel, a noise term, *n(t)*, is added. The noise is considered as white because it has a uniform power spectral density. The mean of the white Gaussian noise is zero and the power spectral density is $N_0/2$ [35].

The Gaussian distributed white noise, *n(t)*, can be represented as $N(\mu, \sigma^2)$. Parameters μ and σ^2 stand for the mean and the variance of this random process, respectively. The Gaussian probability density function is given as

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}. \tag{2.7}$$

When the transmitted signal *s(t)* goes through the AWGN channel, it is affected from *n(t)* and the received signal, *r(t)*, becomes different from *s(t)*. The received signal, *r(t)*, is then given as

$$r(t) = s(t) + n(t). \tag{2.8}$$

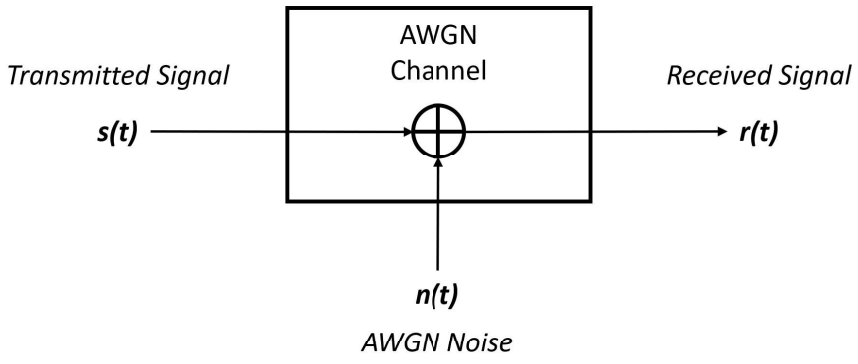

Figure 2.4. AWGN channel.

Block diagram of an AWGN channel is plotted in Figure 2.4. The channel capacity for an AWGN channel is

$$C = B \log_2 \left(1 + \frac{P}{N_0 B}\right), \tag{2.9}$$

where B represents the channel bandwidth and P describes the received signal power [35].

2.2. Channel Polarization

Polar codes made a great improvement in the coding theory by achieving Shannon's channel capacity. They are the first provably capacity-achieving codes in the literature for B-DMCs [10]. Erdal Arıkan introduced polar codes by first proposing the channel polarization phenomenon [1]. Channel polarization is applied by taking N duplicates of a B-DMC, to form a vector of $W_N^{(i)}$, $1 \leq i \leq N$, by using linear transformations. In other words, it is a recursive operation of first combining and then splitting N separate channels. As $N \to \infty$, $I(W)$ values get much closer to either 0 or 1. In Figure 2.5, $I(W)$ values with respect to sorted channel indices are given for a BEC when $N = 1024$ and $\varepsilon = \{0.3, 0.7\}$. If the channel capacity is closer to 1, the channel is considered to be suitable for data transmission. However, if the channel capacity is closer to 0, it is considered not suitable for data transmission.

The number of component channels, N, value in the vector channel $W_N : X_N \to Y_N$ is $N = 2^n$ for $n \geq 0$. At the beginning of the recursive combination, $n = 0$, $W_1 \stackrel{\Delta}{=} W$. At $n = 1$, W_2 is calculated as

$$W_2(y_1, y_2|u_1, u_2) = W(y_1|u_1 \oplus u_2)W(y_2|u_2). \tag{2.10}$$

Here, the $\oplus$ operator denotes the exclusive-or (XOR) operation or modulo-2 addition. Channel combination at $n = 1$ is illustrated in Figure 2.6.

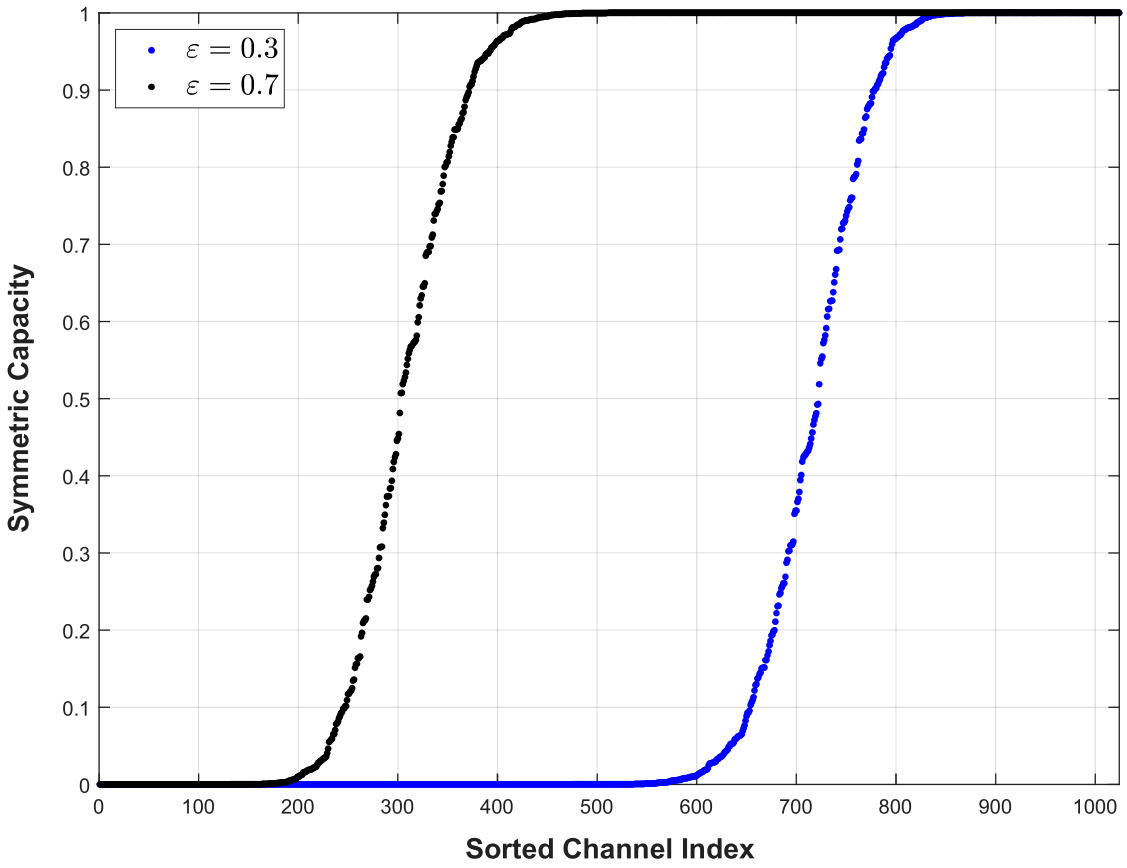

Figure 2.5. Symmetric capacity for a BEC when $N = 1024$ and $\varepsilon = \{0.3, 0.7\}$.

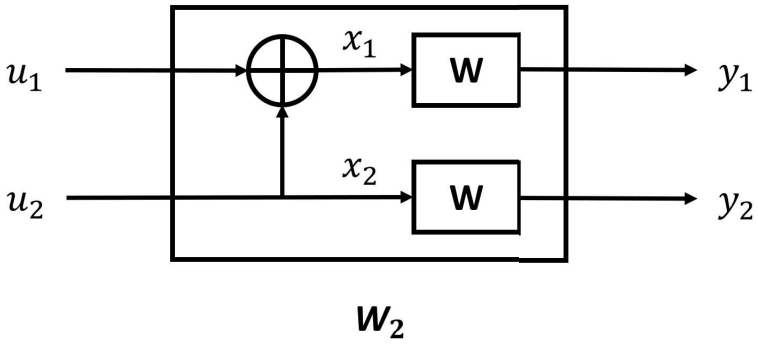

Figure 2.6. Channel combination (W_2).

The input vector $\mathbf{u} = \{u_1, u_2, \ldots, u_N\}$ is encoded into the channel input vector $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}$. The channel input vector goes through a channel and the channel output vector $\mathbf{y} = \{y_1, y_2, \ldots, y_N\}$ is formed. The relationship between $\mathbf{u}$ and $\mathbf{x}$ can

be given as

$$\mathbf{x} = \mathbf{u}\mathbf{G}_N, \tag{2.11}$$

where $\mathbf{G}_N$ is the generator matrix and it is defined as

$$\mathbf{G}_N = \mathbf{B}_N\mathbf{F}^{\otimes n}. \tag{2.12}$$

Here, $\mathbf{B}_N$ is the bit reversal operator, the $\otimes$ operator denotes the Kronecker product, and $\mathbf{F}$ is the kernel matrix given as

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

For example, $\mathbf{G}_2 = \mathbf{F}$ and $\mathbf{G}_4$ is calculated as

$$\begin{aligned} \mathbf{G}_4 &= \mathbf{F}^{\otimes 2} \\ \mathbf{G}_4 &= \mathbf{G}_2 \otimes \mathbf{G}_2 \\ \mathbf{G}_4 &= \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \\ \mathbf{G}_4 &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}. \end{aligned}$$

The transition probabilities can be written by using $\mathbf{G}_N$ as

$$W_N(\mathbf{y}|\mathbf{u}) = W^N(\mathbf{y}|\mathbf{u}\mathbf{G}_N). \tag{2.13}$$

For illustration purposes, in Figure 2.7, the channel combination (W_{16}) at $n = 4$ is depicted. Here, the recursive structure of the channel combination is discernible.

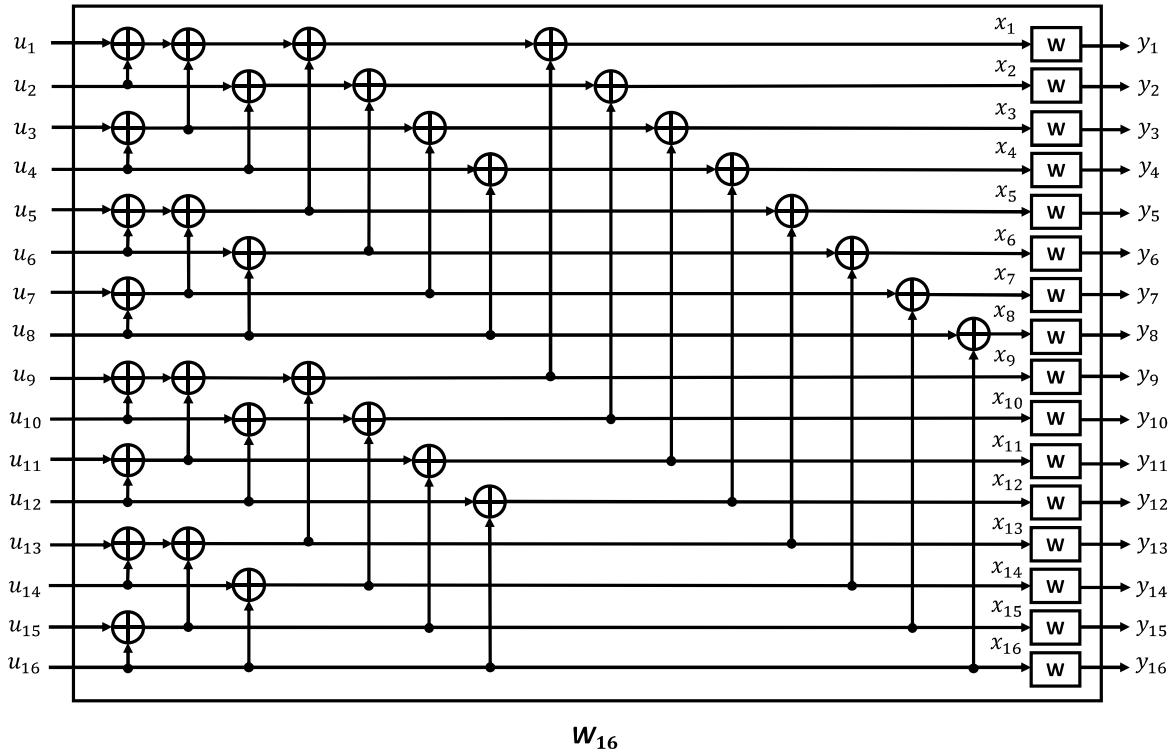

Figure 2.7. Channel combination (W_{16}).

After channel combination, channel splitting follows to obtain channel polarization. It is the process of separating W_N into N channels. The transition probabilities $W_N^{(i)} : X \to Y^N \times X^{i-1}$ are defined as

$$W_N^{(i)}(\mathbf{y}, \mathbf{u}^{i-1}|u_i) \triangleq \sum_{u_{i+1}^N \in X^{N-i}} \frac{1}{2^{N-1}} W_N(\mathbf{y}|\mathbf{u}), \tag{2.14}$$

where $1 \leq i \leq N$ [1]. The fundamentals of a polar code construction technique, Monte Carlo based construction, depends on channel splitting and it is explained in detail in Subsection 2.3.2.

B-DMCs, $(W_N^{(i)})$, become polarized when N is a power of 2 and $N \to \infty$. During the polarization process, for a constant $\delta \in (0, 1)$, $I(W_N^{(i)}) \in [0, \delta)$ converges to $1 - I(W)$

and $I(W_N^{(i)}) \in (1-\delta, 1]$ converges to $I(W)$. In [1], $I(W_N^{(i)})$ for a BEC is given as

$$\begin{aligned} I(W_N^{(2i-1)}) &= I(W_{N/2}^{(1)})^2 \\ I(W_N^{(2i)}) &= 2I(W_{N/2}^{(i)})^2 - I(W_{N/2}^{(i)})^2. \end{aligned} \tag{2.15}$$

Recall from (2.6) that $I(W) = 1 - \varepsilon$, therefore, $I(W_1^{(1)}) = 1 - \varepsilon$.

In Figure 2.8, the polarization steps for a BEC with $\varepsilon = 0.3$ for 10 steps ($n = 10$) are plotted. The $I(W)$ values in each step are calculated using (2.15). From the figure, it is seen that as the steps proceeds channels get more polarized, which means they are getting closer to 0 or 1 in each step.

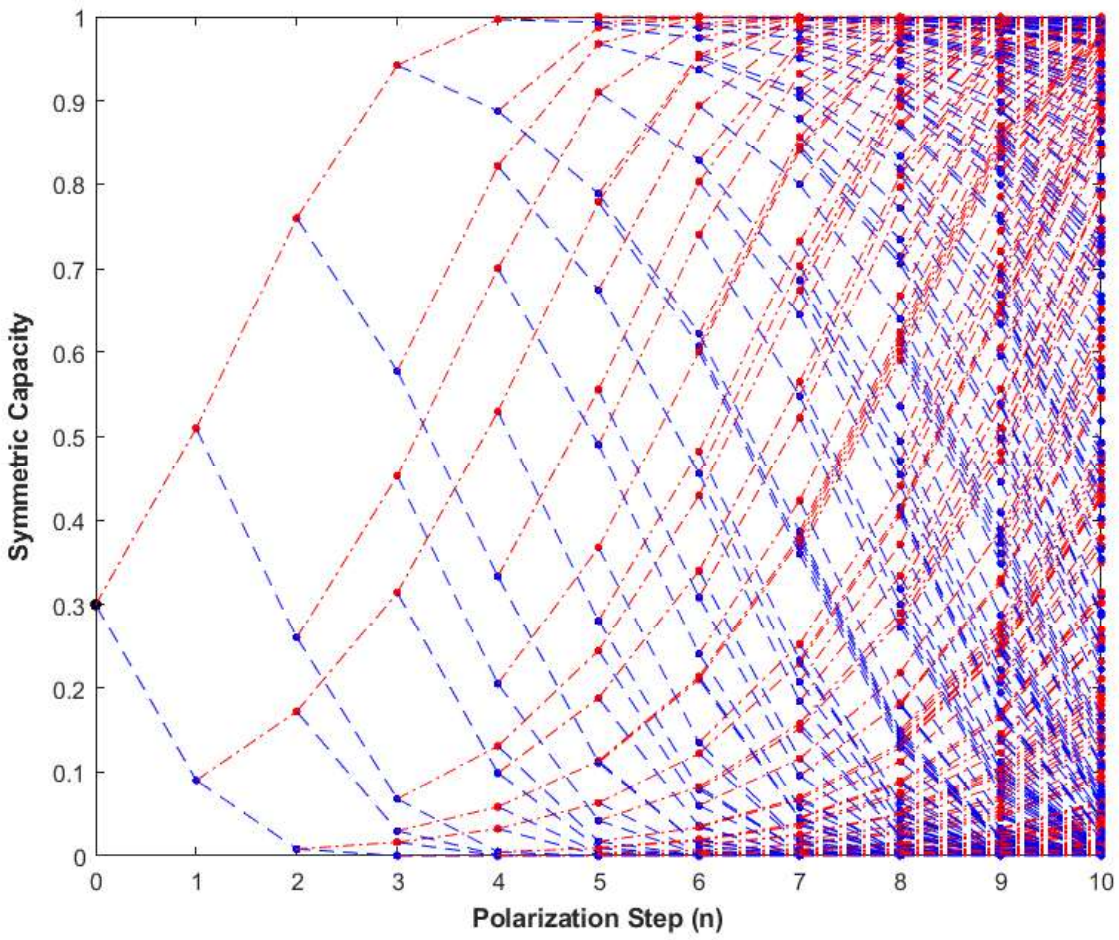

Figure 2.8. Channel polarization evaluation for a BEC with $\varepsilon = 0.3$.

2.3. Polar Code Construction Methods

Polar code construction depends on the channel model and the channel polarization. The challenge in polar code construction is to determine how to design a N-channel system so that the information can be delivered to the receiver with fewer errors. It can simply be described as the selection of K reliable channels out of N available channels. A polar code with a block length of N bits and information bit length of K bits is represented as $\mathbb{P}(N, K)$. The ratio of useful information in polar codes, the code rate, is denoted by R and calculated as $R = K/N$.

The set of information (non-frozen) indices is described by $\mathcal{A}$ and the set of frozen indices is described by $\mathcal{A}_c$. The size of $\mathcal{A}$ is K and the size of $\mathcal{A}_c$ is $N - K$. Information channels are used for data transmissions and frozen channels do not involve useful information. On the frozen channels, depending on the communication system, a constant 0 bit or 1 bit is transmitted. The receiver has prior knowledge of these constant bits and also knows $\mathcal{A}$ and $\mathcal{A}_c$.

There are various polar code construction methods found in the literature. Several fundamental polar code construction methods are given in the following subsections. All these methods propose a rule for selecting the indices of the K reliable channels, to ensure reliable communication with low error rates.

2.3.1. Bhattacharyya Method

Polar code construction with Bhattacharyya parameters was first proposed by Arıkan [1]. It has a basic structure and because of this, it was commonly used in the literature to construct polar codes [36]. Bhattacharyya based polar code construction uses Bhattacharyya parameters to differentiate channel performances. Bhattacharyya

parameters can recursively be calculated as

$$Z(W_N^{(2i-1)}) \leq 2Z(W_{N/2}^{(i)}) - Z(W_{N/2}^{(i)})^2, \tag{2.16}$$

$$Z(W_N^{(2i)}) = Z(W_{N/2}^{(i)})^2, \tag{2.17}$$

where $1 \leq i \leq N/2$ [1]. After Bhattacharyya parameters are obtained for every channel, K channels with the lowest Bhattacharyya parameter values are selected for information channels and the remaining $N - K$ channels are selected for frozen channels. The complexity of this method is $\mathcal{O}(N \log N)$. Here, $\mathcal{O}(\cdot)$ is the big O notation.

2.3.2. Monte Carlo Method

Monte Carlo simulation based polar code construction was first proposed in [1]. The first step in this method is generating N information bits to obtain a codeword. This method uses an all-zero codeword since polar codes are linear codes and all-zero sequence is always a codeword in a linear block code. All-zero codeword is transmitted through a channel under a predetermined signal to noise ratio (SNR) value. Then, on the receiver side, the genie-aided decoder decodes the transmitted codeword. A genie-aided decoder requires the knowledge of the actual codeword. For example, considering a successive cancellation list decoder, if the actual codeword is on the list, the genie-aided decoder decodes the codeword. Since a genie-aided decoder needs the actual codeword, it is impractical and only used in the polar code construction steps. After the decoding process, the decoded codeword is compared with the transmitted codeword to detect channel indices that encountered an error. This process is repeated until the maximum number of iterations defined for the simulation is achieved.

This method is applicable to a limited number of channels and the complexity of this method is very high. The complexity of Monte Carlo simulation based polar code construction is $\mathcal{O}(MN \log N)$ [36]. Here, M denotes the number of iterations for a Monte Carlo simulation. Since the complexity is high, this method is especially not suitable for polar code construction at high SNR values, where the simulations need

much more data for statistical convergence.

In Table 2.1, the results obtained from a Monte Carlo based polar code construction are given. This simulation was done for an AWGN channel when SNR is 1 dB, $N = 16$, and $M = 10^6$. From Table 2.1, it is seen that every channel has different amount of errors. Channels with the most errors are considered for frozen channels and the ones with the least errors are considered for information channels. For example, to construct a code with $R = 1/2$, the frozen channels should be selected as $\mathcal{A}_c = \{1, 2, 3, 4, 5, 6, 7, 9\}$ and the information channels should be selected as $\mathcal{A} = \{8, 10, 11, 12, 13, 14, 15, 16\}$.

Table 2.1. Number of errors per channel for $N = 16$ at 1 dB.

Channel Index	**Number of Errors**	**Channel Index**	**Number of Errors**
1	120887	**9**	74753
2	103876	**10**	25025
3	98245	**11**	17779
4	48469	**12**	1497
5	88880	**13**	11579
6	37078	**14**	671
7	28426	**15**	351
8	3595	**16**	1

2.4. Decoding of Polar Codes

Polar codes are channel dependent codes. Unlike other block codes (i.e., Reed-Muller codes), they use channel dependent parameters like Bhattacharyya parameters. In addition to the channel model, the performance of the polar codes also changes depending on the decoding method. The performance of a constructed polar code can be observed from the bit error rate (BER) or frame error rate (FER) obtained after the

decoding of polar codes. Preliminaries of successive cancellation decoder and successive cancellation list decoder are given in the following subsections.

2.4.1. Successive Cancellation (SC) Decoder

Successive cancellation (SC) decoder is the first decoder proposed for polar codes [1]. SC decoder knows the indices of $N - K$ frozen channels. Before the decoding begins, the receiver determines the log-likelihood ratio (LLR) vector [37]. The LLR vector, $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_N)$, is calculated as

$$\ell_i = \ln\left(\frac{P(y_i|x_i = 0)}{P(y_i|x_i = 1)}\right). \tag{2.18}$$

There are n steps defined in the SC decoder and for each step LLR of every bit is calculated by using two functions: $f(l_1, l_2)$ and $g(l_1, l_2, v)$ [37]. The f function is denoted as

$$f(\ell_1, \ell_2) = 2\tanh^{-1}\left(\tanh\left(\frac{\ell_1}{2}\right)\tanh\left(\frac{\ell_2}{2}\right)\right), \tag{2.19}$$

and the g function is defined as

$$g(\ell_1, \ell_2, v) = \ell_1(-1)^v + \ell_2, \tag{2.20}$$

where v is the hard decision and $v = [0, 1]$. It is seen that the complexity of implementing f and g functions is high. For physical implementations, the alternatives of these functions are given in [37] as

$$f(\ell_1, \ell_2) \approx (1 - 2 \cdot \text{sign}(\ell_1)) \cdot (1 - 2 \cdot \text{sign}(\ell_2)) \cdot \min\{|\ell_1|, |\ell_2|\} \tag{2.21}$$

and

$$g(\ell_1, \ell_2, v) = \ell_2 + \ell_1(1 - v). \tag{2.22}$$

SC decoder sequentially decodes the information channels. If the channels are frozen, decoding is not performed on those channels. This allows avoiding errors encountered on the frozen channels. The SC decoder tries to estimate the $\mathbf{u}$ vector. A block error occurs when the estimate $\hat{\mathbf{u}}$ is not equal to $\mathbf{u}$.

Decoding structure of SC can be represented by a binary tree [38]. The SC decoding tree for $N = 16$ is given in Figure 2.9. On the binary tree, there are $\log_2(N)-1$ stages and at every stage (m) for every node, LLR values are received from the parent node. Then, the child nodes send the hard decision vector, $\boldsymbol{v} = \{v_1, v_2, \ldots, v_{2^m}\}$, back to the parent node. The tree is analyzed starting from the left branches. The f function is used for left nodes, and g function is used for right nodes. In every step, LLR vector $\boldsymbol{f}_m$ and $\boldsymbol{g}_m$ are used as seen from Figure 2.9. Hard decision vector $\boldsymbol{v}$ can be computed as

$$v_i = \begin{cases} v_i^{\text{left}} \oplus v_i^{\text{right}}, & \text{if } i < 2^m \\ v_{i-2^m}^{right}, & \text{otherwise.} \end{cases} \tag{2.23}$$

In short, principals of the SC decoding depends on analyzing a binary tree sequentially by f and g operators to make a hard decision.

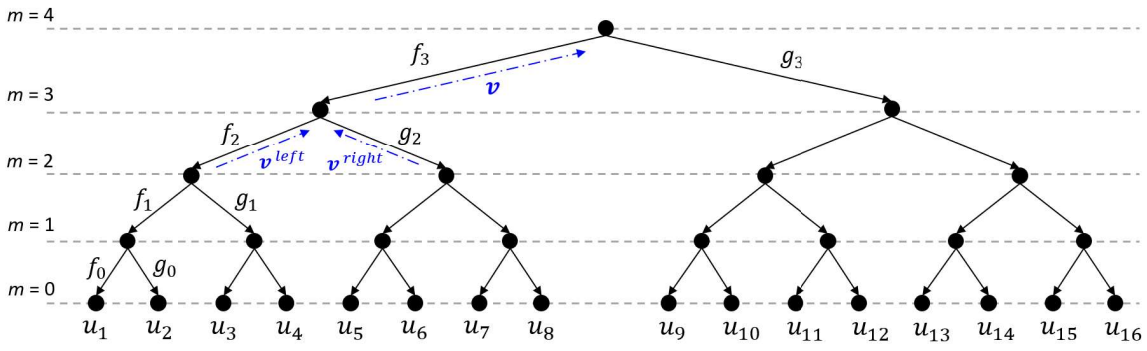

Figure 2.9. SC decoding tree for $N = 16$.

2.4.2. Successive Cancellation List (SCL) Decoder

Successive cancellation list (SCL) decoder is designed to reduce the erroneous decisions encountered during SC decoding, therefore it has a better decoding performance than the SC decoder. It has a very similar decoding structure to the SC decoder. However, the main difference compared to the SC decoder is in the decision of the leaf nodes, $\hat{u}_i$. In the SCL decoder, unlike the SC decoder, $\hat{u}_i$ is not determined immediately. If $\hat{u}_i$ is an information bit, it is estimated as 0 and 1 for different two paths. This method increases the complexity exponentially. To control the complexity, a path metric (PM) is introduced and the path is limited with a list size of L.

PM is used to determine which L paths are better and is calculated as

$$\mathrm{PM}_i = \begin{cases} \mathrm{PM}_{i-1} + \ell_i, & \text{if } \hat{u}_i \neq \begin{cases} 0, & \ell_i > 0 \\ 1, & \text{otherwise,} \end{cases} \\ \mathrm{PM}_{i-1}, & \text{otherwise,} \end{cases} . \tag{2.24}$$

where $\mathrm{PM}_0 = 0$ [38]. L paths that have the lowest PMs are kept in the decoding process. It has been shown in [13] that, as the list size L increases, the performance of the SCL decoder increases.

2.4.3. CRC-Aided (CA) Decoding

CRC-Aided (CA) decoding is introduced to increase the performance of the SCL decoder. In [13], it is shown that CRC improves the SCL decoder performance. In SCL decoding, there is a possibility of having an actual codeword that has a higher PM value with respect to the other L paths. Because of this, the actual codeword might be dropped from the list. CA decoding is expected to eliminate the poor paths with low PMs cleverly to contain better paths on the list. In the CA-SCL decoding, r bits are used for CRC. More specifically, the last r information bits hold the r-bit CRC.

In the presence of CRC bits, the code rate R changes and becomes $R = (K-r)/N$. Also, the polar code representation is now denoted as $\mathbb{P}(N, (K-r)+r)$. The value of r can change depending on the N. For example, for $N = 128$ and $L = 8$, the CRC bit size can be selected as $r = 4$ and for $N = 2048$ and $L = 32$, the CRC bit size can be selected as $r = 16$. The CRC polynomial used for decoding can also be changed. However, using different CRC polynomials does not make a noticeable difference in the performance of the SCL decoder [21].

3. REINFORCEMENT LEARNING (RL)

Artificial intelligence (AI) is a branch of computer science. Systems or machines that involve AI try to mimic human intelligence by collecting data over time. AI has the ability to improve itself over the collected data. The main purpose of AI is to solve tasks where human intelligence is required.

The basis of AI started in 1950 with Alan Turing's question, "Can machines think?". In 1956, a group of mathematicians and scientists made a research project on self-learning systems that can do tasks that depend upon human intelligence [39]. At the Dartmouth Conference, they presented this capability as artificial intelligence. After several decades, in 1997, finally, a machine was able to win a chess game over a chess master. As the technology improved over time, the memory capacity of devices increased tremendously. This increase in capacity has led the AI algorithms to learn faster and pave the way for the huge growth in AI techniques and their use in daily life problems.

Today, AI is being used in various parts of daily life from marketing to security, autonomous vehicles to telecommunications. Smart assistants, facial recognition algorithms, email spam filters, object detection algorithms, self-driving cars, and medical diagnosis algorithms are some examples of AI technology that are used in daily life. Over the years, AI techniques have become more complex and comprehensive. As they evolved, they have divided into subcategories like robotics, natural language processing, machine learning (ML), and deep learning. In this study, a particular type of machine learning algorithms are used.

ML is the ability to self-learn from experiences without having pre-programmed instructions. ML algorithms generally require input data to further use during the learning process. They are commonly used for predicting outcomes. Some fields of data science use ML as a tool to forecast classification or regression problems.

ML has three main categories. These are supervised learning, unsupervised learning, and reinforcement learning (RL). In Figure 3.1, subcategories of ML and their relations between AI and deep learning are illustrated. The subcategories of ML and their differences are explained below.

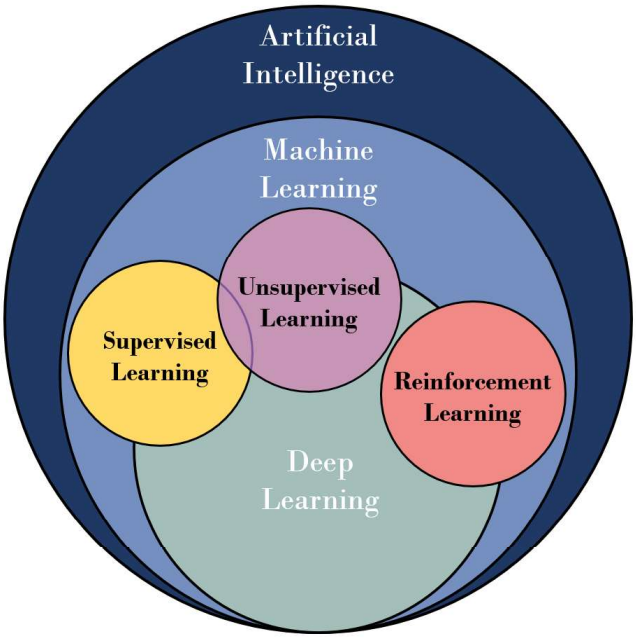

Figure 3.1. The relationship between ML and its subcategories.

In supervised learning, a large amount of labeled input data is required so that supervised algorithms can build a model and forecast outputs. During the learning process, some instances are used as a training set and throughout trials, the algorithm tries to find similarities between the labeled inputs in the training set. For example, random forest, decision trees, **k**-nearest neighbors, and logistic regression are some commonly used supervised learning algorithms. This machine learning method is often used for generalization purposes. With supervised learning, both classification and regression problems can be solved.

In contradiction to supervised learning, in unsupervised learning, a labeled input is not required to forecast outputs. In other words, without having prior knowledge, only by using the instances similarities based on several parameters, unsupervised

learning algorithms can predict the outcome. In unsupervised learning, the main goal is to construct a model which groups the instances into categories. Principal components analysis, singular value decomposition, and **k**-means clustering algorithms are some examples of unsupervised learning algorithms. Like supervised learning algorithms, unsupervised learning methods can also be useful to solve classification problems.

Before going into the details of RL, it is also important to mention deep learning. Deep learning is a multi-layered artificial neural network. It is inspired by the human brain and generally preferred for sophisticated problems. Its main goal is to make a decision by observing patterns in the input data. Deep learning methods can be used in supervised, unsupervised, and RL problems and they can be very effective for high complexity classification and regression problems. They are also used in object detection, image colorization, etc.

The final subcategory of ML is RL. The fundamentals of the RL rely on the interactions with an environment. An agent interacts with the environment and depending on the observations, it determines the following steps to reach the main goal. There are several fundamental definitions in RL like agent, environment, etc., and they are explained in detail in Section 3.1.

In supervised and unsupervised learning, there is a need for a huge amount of data. These data are used for training purposes. However, RL does not require input data, measurements, or examples. The agent in an environment gathers rewards by observations and in this way it generates its own data throughout the learning process. The theoretical difference between RL and other ML methods is that in those methods, the patterns are learned from instances in the input data. On the other hand, in RL, the learning is done by explorations and experiences.

3.1. Definitions in RL

There are some key elements and terms in an RL system. Their presence defines the system and they play a crucial role in the learning process. Fundamental elements, commonly used terms, and their definitions are given below.

- *Environment:* The world or a medium that an agent acts.
- *Agent:* An operator that moves in an environment.
- *Observation:* Inspection of an agent's action in an environment.
- *Action (a):* Movement of an agent towards a direction.
- *State (s):* The current position or situation of an agent in an environment.
- *Value Function:* Predicting how much reward the agent will get in the forthcoming steps.
- *Policy (π):* Deciding how to choose actions, decision-making process.
- *Reward ($\mathcal{R}$):* Positive or negative feedback that the environment gives to the agent depending on the observation.
- *Goal:* An objective that an agent tries to achieve within an environment with the assistance of observations.
- *Episode:* A sequence of actions that starts from an initial state and ends at a terminal state.

An environment can be a grid world, a chess platform, a street, etc. An agent can be a robot, a chess piece, a self-driving car, etc. An agent can choose an action from the action set, A, which includes the possible actions that an agent can make. In a multi-agent system, the environment can have more than one action set. For example, in a chess game, a knight can move north, south, west, and east, while a pawn can move north, northwest, and northeast directions. When an agent takes an action based on a policy and changes its state, the observer gives feedback, a reward. For instance, a self-driving car receives a negative reward when it hits the sidewalk, other cars, or pedestrians. An RL system aims to take actions over time to maximize the cumulative reward. It can also have further goals like minimizing the number of steps taken in

a grid world to reach the terminal state. Throughout episodes, the agent gains more information about the environment to accomplish the system's goals.

A maze game is depicted in Figure 3.2. Here, the environment is a 4x6 grid world. The game starts from the start point (represented in green) and the goal is to reach the end point (represented in red) by taking the minimum amount of steps. The agent (represented in blue) can move upwards, downwards, left, and right. In the environment there are some obstacles (black boxes), and when the agent collides with an obstacle, it receives a negative reward and when it reaches the end point (the terminal state), it receives a positive reward. Throughout episodes, the agent tries to learn from the rewards and finds the best path from the start point to the end point.

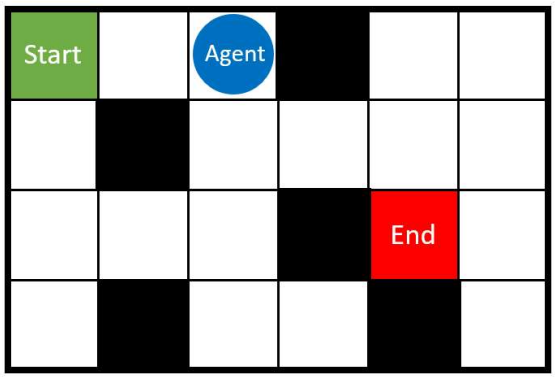

Figure 3.2. A maze game.

3.2. Fundamentals of RL

In RL, the desire is to learn by making mistakes. The agent discovers the environment by taking actions seeking the most reward. Since the agent is interacting with the environment, RL is referred to as active learning. Also, the learning process is sequential, which means that the future interactions may depend on the previous interactions.

The main RL mechanism is shown in Figure 3.3. The agent takes an action in the environment, and the feedback of the action is delivered to the agent by the observer.

Also, the state of the agent is changed depending on the taken action. This process goes on until the goal is achieved.

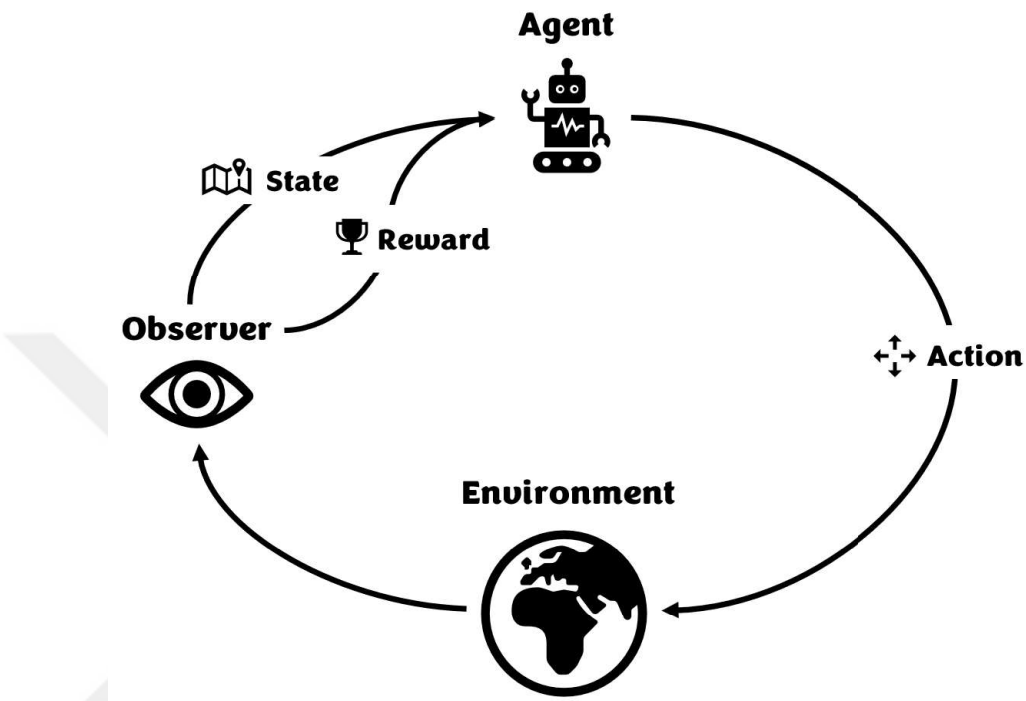

Figure 3.3. RL mechanism.

RL is used when there are sequential decisions. The agent tries to understand the environment by a trial and error approach to achieve the goal. At the beginning of the learning process, it has hardly any information about the environment. So, it begins with exploration. It takes an action, and from the rewards gained from the interactions with the environment, the learning process starts. After many actions and rewards, it reaches its goal and the learning procedure is complete.

In RL, two process models can be used. These are deterministic processes and stochastic processes. In deterministic processes, policy implies that the agent must choose an action with a higher reward [40]. However, in stochastic processes the policy advises the agent to choose the action with the highest reward and there is a possibility to take a different action than suggested to explore the environment [40]. For instance, depending on the application of RL, the agent may also need to minimize the number

of actions. So, if the goal also includes minimizing the number of actions, stochastic processes' policy approach may be more helpful.

In this thesis, RL methods are used to provide an alternative and effective solution for construction problems of polar codes. An RL approach for polar code construction sequentially determines the information channels by maximizing the expected reward. Maximizing the expected reward is equivalent to having a low FER. Therefore, using an RL approach for polar code construction offers promising results.

3.3. Markov Decision Process

Almost every RL problem can be described by using a finite Markov decision process (MDP). MDP is a stochastic process and it is a formalization of sequential decision making. The fundamentals of MDPs and RL depend on the Markov property. A Markov chain has Markov property and it is defined as a sequence of random variables that have values in a finite state set [41]. A state s_t is a Markov if and only if

$$P[s_{t+1}|s_1, \cdots, s_t] = P[s_{t+1}|s_t] \tag{3.1}$$

is satisfied. Markov chain assumes that the probability of taking an action at state s is not affected from previous states, and it only depends on the current state. In a given state, s, the probability of moving to the next state, s', is named as the state transition probability and it is defined as

$$\mathcal{P}_{ss'} = P[s_{t+1} = s'|s_t = s]. \tag{3.2}$$

All state transition probabilities can be represented under a state transition matrix, $\mathcal{P}$, as

$$\mathcal{P} = \begin{bmatrix} \mathcal{P}_{11} & \mathcal{P}_{12} & \mathcal{P}_{13} & \dots & \mathcal{P}_{1m} \\ \mathcal{P}_{21} & \mathcal{P}_{22} & \mathcal{P}_{23} & \dots & \mathcal{P}_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathcal{P}_{m1} & \mathcal{P}_{m2} & \mathcal{P}_{m3} & \dots & \mathcal{P}_{mm} \end{bmatrix}, \tag{3.3}$$

where $s = \{1, 2, \cdots, m\}$ and m is the number of states.

When a state transition occurs, in other words, when an action is taken and therefore the state changes, a reward, $\mathcal{R}$, is received for that transition. In MDPs and RL, the goal is to maximize the cumulative reward. In some cases, the effect of the latest rewards can be more than the earlier rewards. This discount is controlled by a parameter called the discount rate, γ, where $\gamma \in [0, 1]$. If $\gamma = 1$, all rewards have the same amount of effect on the cumulative reward. Total discounted reward, or return, at time step t, G_t, is defined as

$$G_t = \mathcal{R}_{t+1} + \gamma \mathcal{R}_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k \mathcal{R}_{t+k+1}. \tag{3.4}$$

So far, rewards only gave an understanding of how good the process was until the current state. A value function can be used to figure out how good the process will be in the long term. The value function of s, $v(s)$, is defined as

$$\begin{aligned} v(s) &= \mathbb{E}[G_t | s_t = s] \\ &= \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k \mathcal{R}_{t+k+1} | s_t = s \right], \end{aligned} \tag{3.5}$$

where $\mathbb{E}[\cdot]$ is the expected value operator. To decide how to choose an action (a) in a

state (s), a policy π is used. It is defined as

$$\pi(a|s) = P[a_t = a, s_t = s]. \tag{3.6}$$

The value state-action pair under a policy, $q_\pi(s, a)$, can be described as

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[G_t|s_t = s, a_t = a] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k \mathcal{R}_{t+k+1}|s_t = s, a_t = a\right]. \end{aligned} \tag{3.7}$$

In MDPs, exploration is also crucial throughout the learning process. Therefore, being allowed to choose the state that has less reward can help the agent to discover the environment. This allowance of exploration is referred to as greediness. ϵ parameter, where $0 \leq \epsilon \leq 1$, defines how greedy the system is. A greedy policy at state s, $\pi(s)$, can be described by using the estimate of $q_\pi(s, a)$, $Q(s, a)$, as

$$\pi(s) = \begin{cases} \arg\max_a Q(s,a), & \text{with probability } 1-\epsilon \\ \text{random } a \in A, & \text{with probability } \epsilon. \end{cases} \tag{3.8}$$

3.4. RL Algorithms

There are several different algorithms to solve RL problems. Some of these algorithms are considered as offline learning, while some are online learning. In offline learning, the values (e.g., $\pi(s)$) are updated at the end of the episode. However, in online learning, the values are updated after every action. Also, the policy method (off-policy or on-policy) may differ. In the off-policy method, the target policy does not follow the behavior policy (a policy that is used for generating behavior) [40]. On the other hand, in the on-policy method, the target policy and the behavior policy are the same. Furthermore, some algorithms may also use eligibility traces to improve algorithms' performances. In the following subsections, the most commonly used RL

algorithms are explained and their differences are mentioned.

3.4.1. Monte Carlo

Monte Carlo is a method to find the optimal policy and evaluate the value function. This method does not need prior knowledge, it solely needs experience. Taking actions significantly depends on randomness. This method is also called an offline learning method because the reward is only received at the end of an episode. Monte Carlo methods try to estimate the value function, $v(\cdot)$, on a policy π for a given state s, $v_\pi(s)$, by taking sample returns at the end of each episode. It takes the mean of the cumulative reward, G_t, collected in each episode. It is assumed that all the episodes terminate.

Monte Carlo methods require a high amount of episodes. For each state that is visited in an episode, a G_t for that state is calculated. Then it is combined with the average of returns at state s.

3.4.2. Q-Learning

Q-learning algorithm uses the off-policy method. In this algorithm, the aim is to find the most desirable action for a given state. Q-learning algorithm uses the state-action pairs and finds the one with the maximum expected reward.

The update rule for the value function of the state-action pair, $Q(s, a)$, for Q-learning is defined as

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha\big[\mathcal{R}_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)\big], \tag{3.9}$$

where α represents the learning rate [40]. It determines how much the incoming value will affect $Q_t(s, a)$. The value of α, where $\alpha \in [0, 1]$, also affects the convergence of the learning.

The Q-learning algorithm starts with arbitrarily initializing $Q(s, a)$ for every state-action pair. For the terminal state, the corresponding value function is set to 0. At the beginning of every episode, s is initialized and after that, for each step in an episode, a is chosen by using π, the reward is observed, and the next state is located. Then $Q(s, a)$ and s are updated. When the agent reaches the terminal state, an episode is considered as complete. This process goes on until the total number of episodes is achieved.

3.4.3. State Action Reward State Action (SARSA)

In contradiction to Q-learning, the state action reward state action (SARSA) algorithm uses the on-policy method [40]. SARSA contains the current state s_t, the current action a_t, the reward gained from the action $\mathcal{R}_t$, next state s_{t+1}, and the future action a_{t+1}.

The update rule for the value function of state-action pair, $Q(s, a)$, for the SARSA algorithm is calculated as

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha\big[\mathcal{R}_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)\big]. \tag{3.10}$$

SARSA follows a greedy policy and a_{t+1} is selected based on that policy. However, in Q-learning, the maximum value of $Q(s, a)$ is chosen. Therefore, the Q-learning algorithm does not pursue a greedy policy.

The SARSA algorithm starts with arbitrarily initializing $Q(s, a), \forall s \in S, \forall a \in A$. For the terminal state, the corresponding value function is set to 0. At the beginning of every episode, s is initialized and a is chosen by using π. After that, for each step in an episode, a is taken, the reward is observed, and the next state is located. The next action, a', is chosen by using π. Then $Q(s, a)$, s, and a are updated. When the agent reaches the terminal state, an episode is considered as complete. This process goes on

until the total number of episodes is achieved.

3.4.4. Watkins's Q(λ)

Watkins's $Q(\lambda)$ algorithm combines the Q-learning method with eligibility trace. Eligibility traces introduce a short-term memory to the system. The purpose of using eligibility traces is to learn more efficiently [40].

Let $E_t(s,a), \forall s \in S, \forall a \in A$, denote the eligibility trace of a state-action pair. The initial value of $E_0(s,a) = 0$. $E_t(s,a)$, for Watkins's Q(λ) method, is described as

$$E_t(s,a) = \begin{cases} \gamma\lambda E_{t-1}(s,a) + I_{sS_t} I_{aA_t}, & \text{if } Q_{t-1}(s_t,a_t) = \max\limits_a Q_{t-1}(s_t,a) \\ I_{sS_t} I_{aA_t}, & \text{otherwise.} \end{cases}, \tag{3.11}$$

where $\lambda \in [0,1]$ demonstrates the eligibility decay factor and I represents the identity matrix [40]. When λ goes towards zero, only the value function of the most recent states is affected. On the other hand, when λ goes towards one, the value function of much earlier steps is altered.

With the presence of eligibility trace, $Q(s,a)$ is updated as

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha\delta_t E_t(s,a), \tag{3.12}$$

where δ_t is the temporal difference error [40] and it is defined as

$$\delta_t = \mathcal{R}_{t+1} + \gamma \max_{a'} Q_t(s_{t+1},a') - Q_t(s_t,a_t). \tag{3.13}$$

The temporal difference error approximately shows the difference between the utmost reward and the current prediction.

The Watkins's $Q(\lambda)$ algorithm starts with arbitrarily initializing $Q(s,a), \forall s \in S$, $\forall a \in A$. At the beginning of every episode, $E(s,a)$, $\forall s \in S, \forall a \in A$, is set to 0, s is initialized and a is chosen by using π. After that, for each step in an episode, a is taken, the reward is observed, and the next state is located. The next action is chosen by using π. Then δ is calculated and $E(s,a)$ is incremented by one. For all states and actions, $Q(s,a)$, s, and a are updated. If a non-greedy action is taken, $E(s,a)$ is set to zero, otherwise, it is updated as described in (3.11). When the agent reaches the terminal state, an episode is considered as complete. This process goes on until the total number of episodes is achieved.

3.4.5. SARSA(λ)

SARSA(λ) algorithm is the combination of the SARSA method and the eligibility trace. For SARSA(λ) algorithm, the eligibility trace of a state-action pair is updated as

$$E_t(s,a) = \begin{cases} \gamma\lambda E_{t-1}(s,a) + 1, & \text{if } s = s_t \text{ and } a = a_t \\ \gamma\lambda E_{t-1}(s,a), & \text{otherwise.} \end{cases} \quad \text{for all } s \in S, a \in A. \tag{3.14}$$

$Q(s,a)$ is updated as described in (3.12). However, the temporal difference error in (3.12) is calculated differently. For SARSA(λ), δ_t is defined in [40] as

$$\delta_t = \mathcal{R}_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t). \tag{3.15}$$

The SARSA(λ) algorithm starts with arbitrarily initializing $Q(s,a)$ for every state-action pair. At the beginning of every episode, $E(s,a)$, $\forall s \in S, \forall a \in A$, is set to 0, s is initialized and a is chosen by using π. After that, for each step in an episode, a is taken, the reward is observed, and the next state is located. The next action is chosen by using π. Then δ is calculated and $E(s,a)$ is incremented by one. For all

states and actions, $Q(s, a)$, $E(s, a)$, s, and a are updated. When the agent reaches the terminal state, an episode is considered as complete. This process goes on until the total number of episodes is achieved.

The calculation process of the SARSA(λ) algorithm for some steps is given with an illustrative example below. The agent acts in a 3x4 grid environment. The environment and the steps for the first episode are shown in Figure 3.4. It is assumed that the agent receives a reward only in two conditions: a "-1" reward when it goes through an obstacle (black boxes) and a "+1" reward when it reaches the terminal state (red box). The agent starts exploring the environment from the start point, $s = (0, 0)$. When the agent reaches the terminal state, $s = (2, 3)$, an episode is completed. The agent is allowed to move "up (U)", "down (D)", "left (L)", or "right (R)". In other words, $A = \{\text{U, D, L, R}\}$.

To observe the changes in $Q(s, a)$, it is initialized as $Q(s, a) = 0, \forall s \in S, \forall a \in A$. Since it is the first episode, the agent does not have information about the environment. Hence, it takes actions randomly. It is assumed that in the first episode there are five steps and the agent takes the following actions: "R", "D", "R", "R", and "D". Parameters are selected as: $\alpha = 0.5$, $\lambda = 0.4$, and $\gamma = 0.9$. The evaluation of $Q(s, a)$ and $E(s, a)$ can be observed from Tables 3.1 to 3.4.

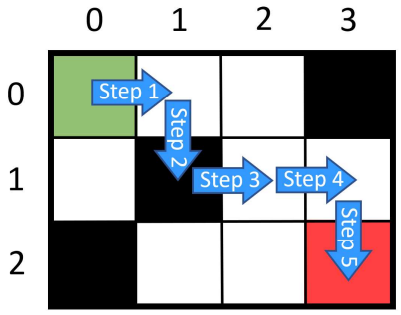

Figure 3.4. An example environment for SARSA(λ) algorithm.

- *Step 1:* $t = 0$, $a =$ "R", $a' =$ "D", $s = (0,0)$, $s' = (0,1)$, and $\mathcal{R}_1 = 0$.

$$\begin{aligned}\delta_0 &= 0 + (0.9) \cdot Q_0((0,1), \text{“D”}) - Q_0((0,0), \text{“R”}) \\ &= 0 + (0.9) \cdot (0) - 0 \\ &= 0.\end{aligned}$$

First, $E_0((0,0), \text{“R”})$ is incremented by one, as in (3.14).

$$\begin{aligned}E_0((0,0), \text{“R”}) &= 0 + 1 \\ &= 1.\end{aligned}$$

Then for all state-action pairs, $Q(s,a)$ and $E(s,a)$ are updated. Since only $((0,0), \text{“R”})$ pair is visited, only $((0,0), \text{“R”})$ pair will be calculated.

$$\begin{aligned}Q_1((0,0), \text{“R”}) &= Q_0((0,0), \text{“R”}) + (0.5) \cdot (0) \cdot (1) \\ &= 0.\end{aligned}$$

and

$$\begin{aligned}E_0((0,0), \text{“R”}) &= (0.9) \cdot (0.4) \cdot (1) \\ &= 0.36.\end{aligned}$$

- *Step 2:* $t = 1$, $a =$ "D", $a' =$ "R", $s = (0,1)$, $s' = (1,1)$, and $\mathcal{R}_2 = -1$.

$$\begin{aligned}\delta_1 &= -1 + (0.9) \cdot Q_1((1,1), \text{“R”}) - Q_1((0,1), \text{“D”}) \\ &= -1 + (0.9) \cdot (0) - 0 \\ &= -1.\end{aligned}$$

Table 3.1. $Q(s,a)$ and $E(s,a)$ after Step 1.

State	$Q(s,a)$				$E(s,a)$			
	Action				Action			
	U	D	L	R	U	D	L	R
(0,0)	0	0	0	0	0	0	0	0.36
(0,1)	0	0	0	0	0	0	0	0
(0,2)	0	0	0	0	0	0	0	0
(0,3)	0	0	0	0	0	0	0	0
(1,0)	0	0	0	0	0	0	0	0
(1,1)	0	0	0	0	0	0	0	0
(1,2)	0	0	0	0	0	0	0	0
(1,3)	0	0	0	0	0	0	0	0
(2,0)	0	0	0	0	0	0	0	0
(2,1)	0	0	0	0	0	0	0	0
(2,2)	0	0	0	0	0	0	0	0
(2,3)	0	0	0	0	0	0	0	0

First, $E_1((0,1), \text{“D”})$ is incremented by one.

$$\begin{aligned} E_1((0,1), \text{“D”}) &= 0 + 1 \\ &= 1. \end{aligned}$$

Then for all state-action pairs, $Q(s,a)$ and $E(s,a)$ are updated. Since only $((0,0), \text{“R”})$ and $((0,1), \text{“D”})$ pairs are visited so far, only those pairs will be calculated.

$$\begin{aligned} Q_2((0,0), \text{“R”}) &= Q_1((0,0), \text{“R”}) + (0.5) \cdot (-1) \cdot E_1((0,0), \text{“R”}) \\ &= 0 + (0.5) \cdot (-1) \cdot (0.36) \\ &= -0.18. \end{aligned}$$

$$\begin{aligned} Q_2((0,1), \text{“D”}) &= Q_1((0,1), \text{“D”}) + (0.5) \cdot (-1) \cdot E_1((0,1), \text{“D”}) \\ &= 0 + (0.5) \cdot (-1) \cdot (1) \\ &= -0.5. \end{aligned}$$

and

$$\begin{aligned} E_1((0,0), \text{“R”}) &= (0.9) \cdot (0.4) \cdot (0.36) \\ &= 0.1296. \end{aligned}$$

$$\begin{aligned} E_1((0,1), \text{“D”}) &= (0.9) \cdot (0.4) \cdot (1) \\ &= 0.36. \end{aligned}$$

Table 3.2. $Q(s,a)$ and $E(s,a)$ after Step 2.

	$Q(s,a)$				$E(s,a)$			
State	Action				Action			
	U	D	L	R	U	D	L	R
(0,0)	0	0	0	-0.18	0	0	0	0.1296
(0,1)	0	-0.5	0	0	0	0.36	0	0
(0,2)	0	0	0	0	0	0	0	0
(0,3)	0	0	0	0	0	0	0	0
(1,0)	0	0	0	0	0	0	0	0
(1,1)	0	0	0	0	0	0	0	0
(1,2)	0	0	0	0	0	0	0	0
(1,3)	0	0	0	0	0	0	0	0
(2,0)	0	0	0	0	0	0	0	0
(2,1)	0	0	0	0	0	0	0	0
(2,2)	0	0	0	0	0	0	0	0
(2,3)	0	0	0	0	0	0	0	0

- *Step 3:* $t = 2$, $a =$ "R", $a' =$ "R", $s = (1,1)$, $s' = (1,2)$, and $\mathcal{R}_3 = 0$.

$$\begin{aligned}\delta_2 &= 0 + (0.9) \cdot Q_2((1,2), \text{“R”}) - Q_2((1,1), \text{“R”}) \\ &= 0 + (0.9) \cdot (0) - 0 \\ &= 0.\end{aligned}$$

First, $E_2((1,1), \text{“R”})$ is incremented by one.

$$\begin{aligned}E_2((1,1), \text{“R”}) &= 0 + 1 \\ &= 1.\end{aligned}$$

Then for all state-action pairs, $Q(s,a)$ and $E(s,a)$ are updated. Only $((0,0), \text{“R”})$, $((0,1), \text{“D”})$, and $((1,1), \text{“R”})$ pairs are visited thus far, therefore, only those pairs will be calculated.

$$\begin{aligned}Q_3((0,0), \text{“R”}) &= Q_2((0,0), \text{“R”}) + (0.5) \cdot (-1) \cdot E_2((0,0), \text{“R”}) \\ &= -0.18 + (0.5) \cdot (0) \cdot (0.1296) \\ &= -0.18.\end{aligned}$$

$$\begin{aligned}Q_3((0,1), \text{“D”}) &= Q_2((0,1), \text{“D”}) + (0.5) \cdot (0) \cdot E_2((0,1), \text{“D”}) \\ &= -0.5 + (0.5) \cdot (0) \cdot (0.36) \\ &= -0.5.\end{aligned}$$

$$\begin{aligned}Q_3((1,1), \text{“D”}) &= Q_2((1,1), \text{“D”}) + (0.5) \cdot (0) \cdot E_2((1,1), \text{“D”}) \\ &= 0 + (0.5) \cdot (0) \cdot (1) \\ &= 0.\end{aligned}$$

and

$$\begin{aligned} E_2((0,0), \text{“R”}) &= (0.9) \cdot (0.4) \cdot (0.1296) \\ &= 0.0467. \end{aligned}$$

$$\begin{aligned} E_2((0,1), \text{“D”}) &= (0.9) \cdot (0.4) \cdot (0.36) \\ &= 0.1296. \end{aligned}$$

$$\begin{aligned} E_2((1,1), \text{“R”}) &= (0.9) \cdot (0.4) \cdot (1) \\ &= 0.36. \end{aligned}$$

Table 3.3. $Q(s,a)$ and $E(s,a)$ after Step 3.

	$Q(s,a)$				$E(s,a)$			
State	Action				Action			
	U	D	L	R	U	D	L	R
(0,0)	0	0	0	-0.18	0	0	0	0.0467
(0,1)	0	-0.5	0	0	0	0.1296	0	0
(0,2)	0	0	0	0	0	0	0	0
(0,3)	0	0	0	0	0	0	0	0
(1,0)	0	0	0	0	0	0	0	0.36
(1,1)	0	0	0	0	0	0	0	0
(1,2)	0	0	0	0	0	0	0	0
(1,3)	0	0	0	0	0	0	0	0
(2,0)	0	0	0	0	0	0	0	0
(2,1)	0	0	0	0	0	0	0	0
(2,2)	0	0	0	0	0	0	0	0
(2,3)	0	0	0	0	0	0	0	0

Then, Step 4 and Step 5 follow. The agent reaches the terminal state, therefore the first episode finishes. The final values of $Q(s,a)$ and $E(s,a)$ at the end of the first episode are given in Table 3.4. For the second episode, the values in Table 3.4 are used in the first step, and then it is updated depending on the sequences. This process goes on until the maximum episode number is reached.

Table 3.4. $Q(s,a)$ and $E(s,a)$ at the end of the first episode (after Step 5).

State	$Q(s,a)$				$E(s,a)$			
	Action				Action			
	U	D	L	R	U	D	L	R
(0,0)	0	0	0	-0.1716	0	0	0	0.006
(0,1)	0	-0.4767	0	0	0	0.0168	0	0
(0,2)	0	0	0	0	0	0	0	0
(0,3)	0	0	0	0	0	0	0	0
(1,0)	0	0	0	0.0648	0	0	0	0.0467
(1,1)	0	0	0	0.18	0	0	0	0.1296
(1,2)	0	0	0	0.5	0	0	0	0.36
(1,3)	0	0	0	0	0	0	0	0
(2,0)	0	0	0	0	0	0	0	0
(2,1)	0	0	0	0	0	0	0	0
(2,2)	0	0	0	0	0	0	0	0
(2,3)	0	0	0	0	0	0	0	0

3.5. Deep RL

Deep RL is the combination of artificial neural networks and RL. In some cases the complexity of the environment can be huge, there can be numerous states, and there may be a need for a large number of sequential decisions. In those cases, the traditional RL methods cannot solve the task or accomplish the goal. The high complexity further makes it hard to set up a system that can solve the problem.

Neural networks can be used for representing the model, the value function, or the policy. When the agent interacts with the environment more, the performance of the deep learning approach gets better. Daily life examples of deep RL are self-driving cars, robot vacuums, some video games, etc.

4. POLAR CODE CONSTRUCTION WITH REINFORCEMENT LEARNING

As mentioned in the previous chapter, RL can offer efficient solutions for various problems. In [29], the polar code construction problem was correlated to a maze game and an RL method was offered to solve the maze game. It was shown that using an RL method for polar code construction gives promising and reliable solutions. In the following subsections, first a model proposed in [29] is introduced, then several new methods are proposed for polar code construction that uses RL methods. Weakness in RL methods for polar code design with higher block lengths are also discussed.

4.1. Maze Approach

There are several ways to construct polar codes. When the performance and feasibility are considered, reinforcement learning gives promising solutions. Considering computational limits, Monte Carlo based polar code construction offers a limited performance due to the high number of simulations needed to sort channels depending on the channel reliability. Unlike a channel sorting based method, the maze game approach proposed in [29] tries to design polar codes by sequentially deciding the frozen and information channels.

The maze game uses the SARSA(λ) method to construct a polar code. The agent sets the channel as information or frozen by taking an action. When the state is changed, the path that the agent take is evaluated by an SCL-genie decoder, and depending on the response of the decoder, the agent gets a reward. By pursuing the path with the highest expected reward, which is equivalent to seeking a low FER, the agent constructs a polar code. The details and the rules defined for the maze game are given below. The environment of the maze game for $\mathbb{P}(16, 8)$ is shown in Figure 4.1.

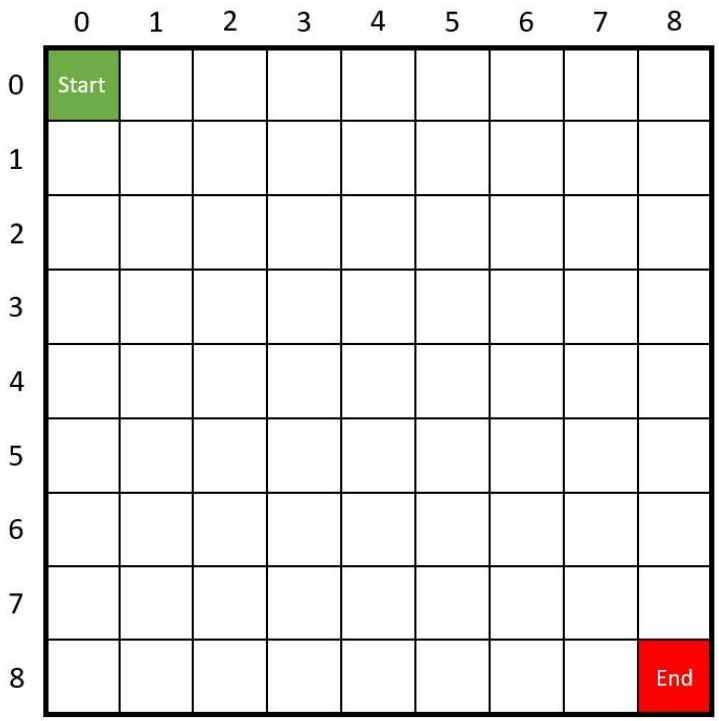

Figure 4.1. Environment of the maze game for $\mathbb{P}(16, 8)$.

- *Environment:* $(N - K + 1)$x$(K + 1)$ grid world. It is not allowed to exceed maze boundaries.
- *Actions:* The action set is defined as $A = \{$"down", "right"$\}$. The agent determines the type of the channel (frozen or information) by taking an action. So, maximum N possible actions are allowed to be taken. Choosing to go "down" is equivalent to setting the channel as frozen and going "right" is equivalent to setting the channel as information.
- *States:* Start state is fixed and located at $(0, 0)$ and the terminal (end) state is fixed and located at $(N - K, K)$. The states in the environment are denoted as (row, column).
- *Reward:* When the codeword is dropped from the list of the SCL-genie decoder, it receives a negative reward, and when it reaches the terminal state it receives a positive reward. Therefore, the reward is determined by the SCL-genie decoder.
- *Goal:* The goal is to reach the "End" state without dropping the codeword from the list and having the maximum expected return.

- *Episode:* Each episode starts with an arbitrary set of LLR values obtained for a specific SNR level. The agent starts exploring the environment from the start state $(0, 0)$. If the agent gets a negative reward or reaches the terminal state, an episode is considered as complete.

The agent follows a greedy approach. However, the level of greediness decreases over episodes. The level of greediness at an episode is calculated using

$$\epsilon_i = 1 - \frac{i}{\Theta}, \tag{4.1}$$

where i is the episode number and Θ denotes the total number of episodes. It is seen from (4.1) that the agent follows a very greedy approach at the beginning of the game and a less greedy approach when the game gets closer to the end.

As the agent explores the environment over episodes, the path that the agent follows converges to the path with the highest expected reward. Therefore, the performance of the constructed polar codes gets better throughout the episodes. Before the game finishes, the agent is expected to build the path with the lowest FER. In Figures 4.2 and 4.3, heat map plots for a $\mathbb{P}(16, 8)$ at 2 dB SNR are shown. Figure 4.2 is obtained after 100 episodes and Figure 4.3 is obtained after 2000 episodes. By comparing the two figures, it is seen that the greediness level decreased and the path that the agent follows converged into a single path.

It is seen that the constructed polar code after 100 episodes is different than the one constructed after 2000 episodes. The constructed polar code after 100 episodes has the set of frozen channel indices as $\mathcal{A}_c = \{1, 2, 4, 5, 6, 7, 9, 10\}$ and information channel indices as $\mathcal{A} = \{3, 8, 11, 12, 13, 14, 15, 16\}$. At the end of the 2000th episode, the set of frozen channel indices becomes $\mathcal{A}_c = \{1, 2, 3, 5, 6, 7, 9, 10\}$ and the set of information channel indices becomes $\mathcal{A} = \{4, 8, 11, 12, 13, 14, 15, 16\}$. Since their information channel indices are different, their performances are different. At 2 dB, the FER value after 100 episodes is 0.1322 and after 2000 episodes it decreases to 0.0945.

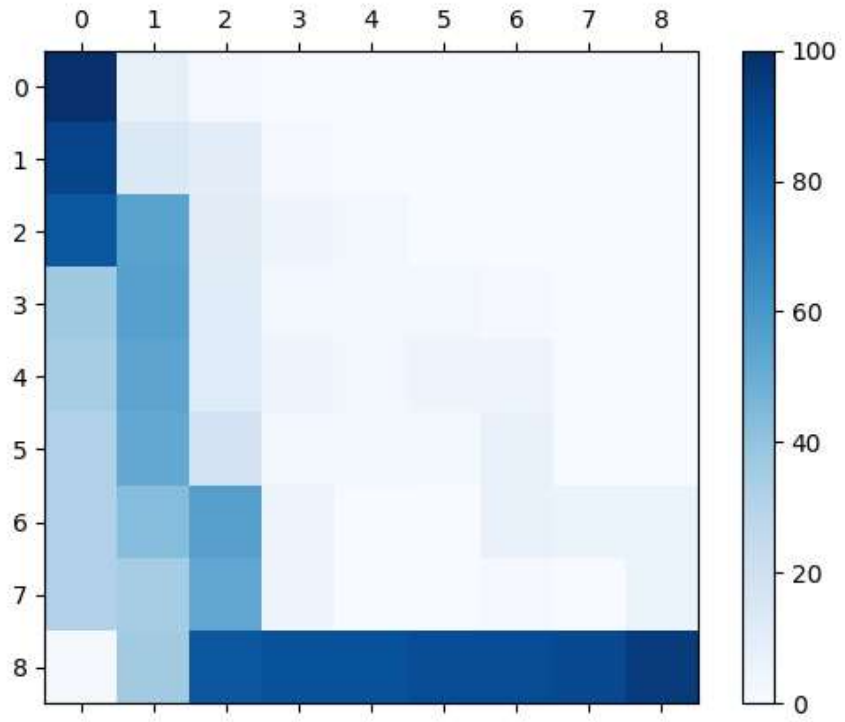

Figure 4.2. Evaluation of learning after 100 episodes.

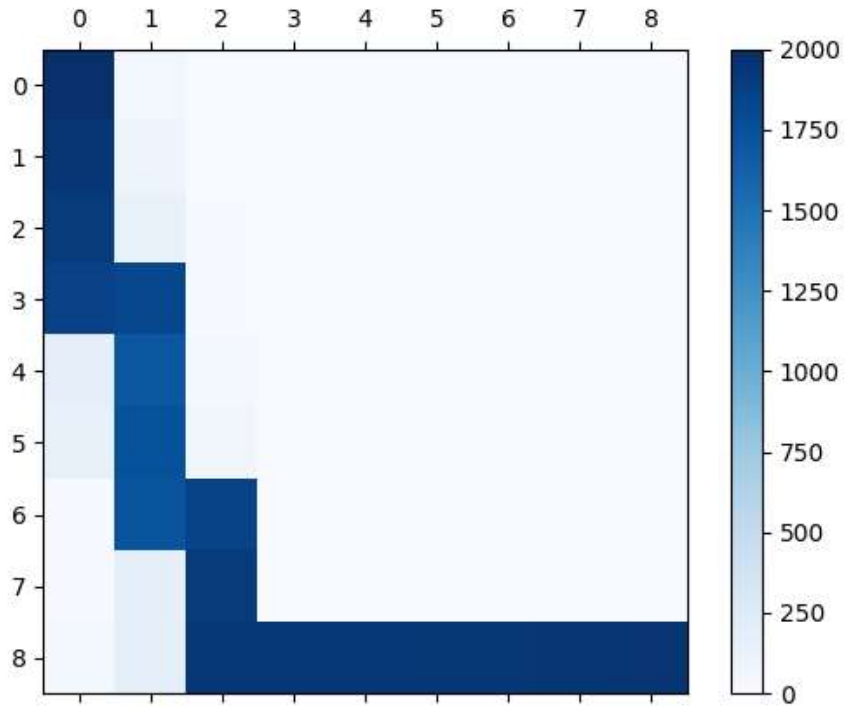

Figure 4.3. Evaluation of learning after 2000 episodes.

In Figure 4.4, the FER performance of the maze game based polar code construction method is plotted for various SNR levels and block lengths. Additionally, for comparison, the FER performance of the Monte Carlo based polar code construction method is also shown. Here, the code rate is $R = 1/2$. Note that both Monte Carlo and maze game methods are applicable for various code rates.

During the simulations, the AWGN channel model with binary phase-shift keying (BPSK) modulation is used. Key parameters (learning rate α, discount rate γ, eligibility decay factor λ, the total number of episodes Θ, and list size L) used during the simulations of the maze game are given in Table 4.1. The CRC polynomial used for 4-bit CRC is $C(x) = x^4 + x + 1$. As mentioned in Chapter 2, the CRC polynomial does not significantly change the FER performance.

Table 4.1. Parameter set of the maze game for various block lengths.

	Block Length (N)			
Parameters	(16)	(64)	(128)	(256)
α	0.05	0.01	0.005	0.001
γ	1	1	1	1
λ	0.3	0.5	0.75	0.8
Θ	2000	100000	200000	250000
L	4	4	8	8

It is seen from Figure 4.4 that the maze game method performs better than the Monte Carlo based polar code construction method for various SNR levels and block lengths ($N \leq 128$). As the SNR level increases, the gap between the two construction methods widens. When $N \geq 256$, the polar code constructed using the maze approach performs poorly. This is due to the increased complexity in the SARSA(λ) method. Recall from Chapter 3 that, as the number of states increases, the systems fail to accomplish the goal. To achieve the complexity problem and construct codes with higher block lengths, new methods that use a similar maze game approach are proposed

in the following subsections.

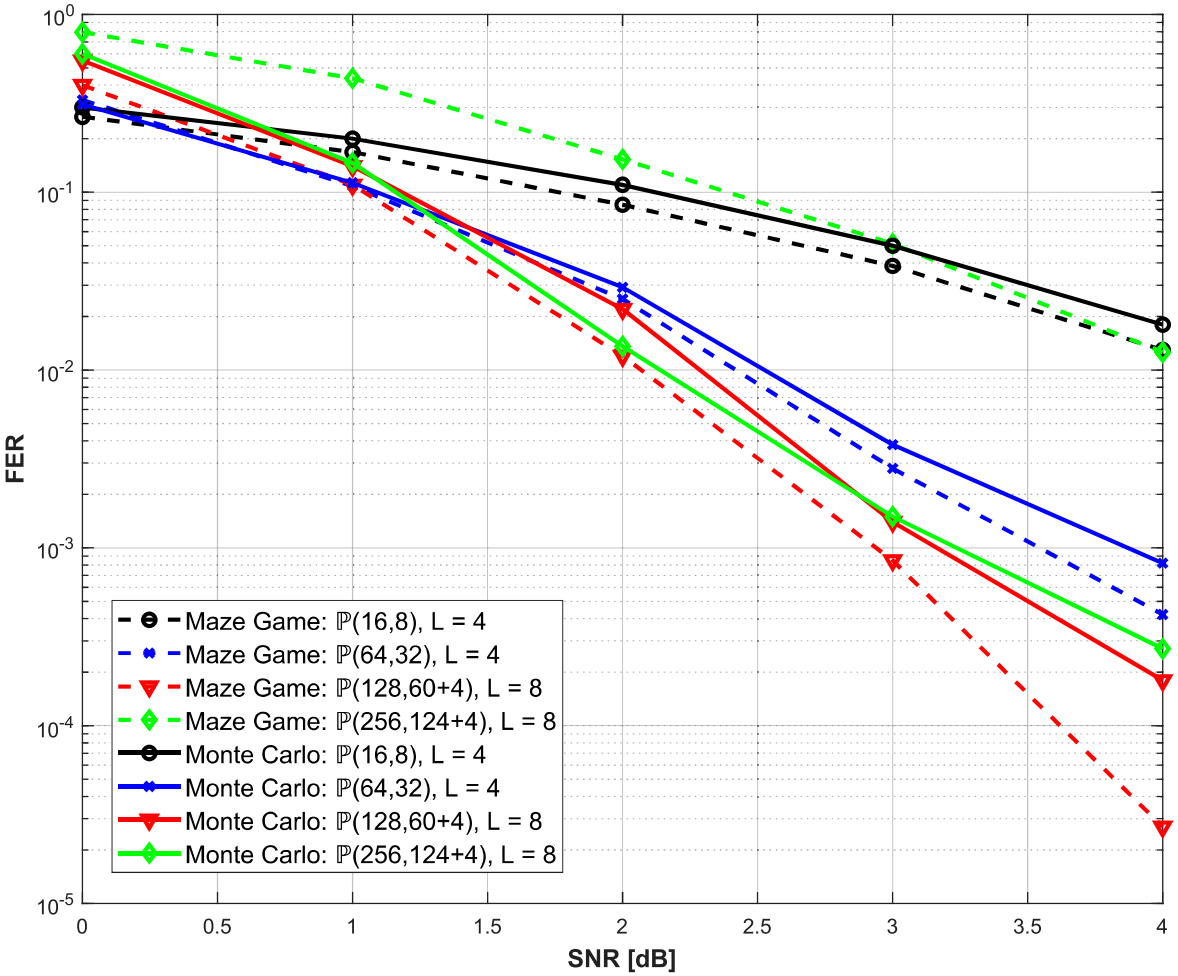

Figure 4.4. FER performance of the maze approach.

4.2. Reduced Cluster-Based Maze Approach

The maze method proposed in [29] fails to construct reliable codes when $N > 128$. The main reason behind this poor performance is the complexity of the game. As the block length increases, the number of states in the maze game increases exponentially. The proposed reduced cluster-based maze approach aims to reduce the complexity and construct polar codes with higher block code lengths.

The reduced cluster-based maze approach is similar to the maze approach proposed in [29]. In the reduced cluster-based method, the environment changes, some restricted areas are defined and new borders are added. The fundamentals of the reduced cluster-based maze approach depend on limiting the actions and the number of states by setting some channels as frozen and information before the game begins. As an illustration, the environment for $R = 8/16$, or in other words, $\mathbb{P}(16, 8)$ is shown in Figure 4.5. Here, the shaded areas are restricted. Thus, the agent is not allowed to enter these areas. Also, the dashed lines define the borders and it is not allowed to cross the borders. Therefore, the agent's actions are limited in some states. This forces the agent to go down or right in some cells, essentially forcing it to declare the corresponding channel frozen or information, respectively.

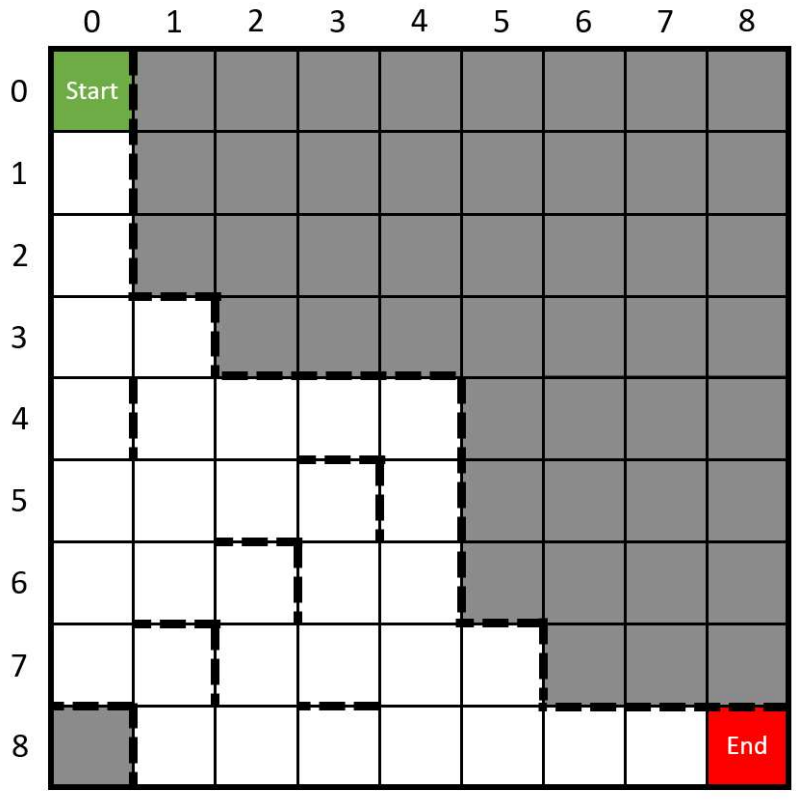

Figure 4.5. Environment of the reduced cluster-based maze game for $\mathbb{P}(16, 8)$.

As mentioned earlier, the maze game is applicable to various code rates. Environment of an $\mathbb{P}(16, 12)$ construction, $R = 12/16$, is presented in Figure 4.6. After the

restrictions are applied to $\mathbb{P}(16, 8)$, there are 20 possible paths left. When the code rate increases to $R = 12/16$, the total number of available paths decreases to 4. This also means that there are 20 distinctive constructions for $\mathbb{P}(16, 8)$ and 4 distinctive constructions for $\mathbb{P}(16, 12)$. As seen from Figures 4.5 and 4.6, the reduced cluster-based construction method decreases the complexity of the maze game.

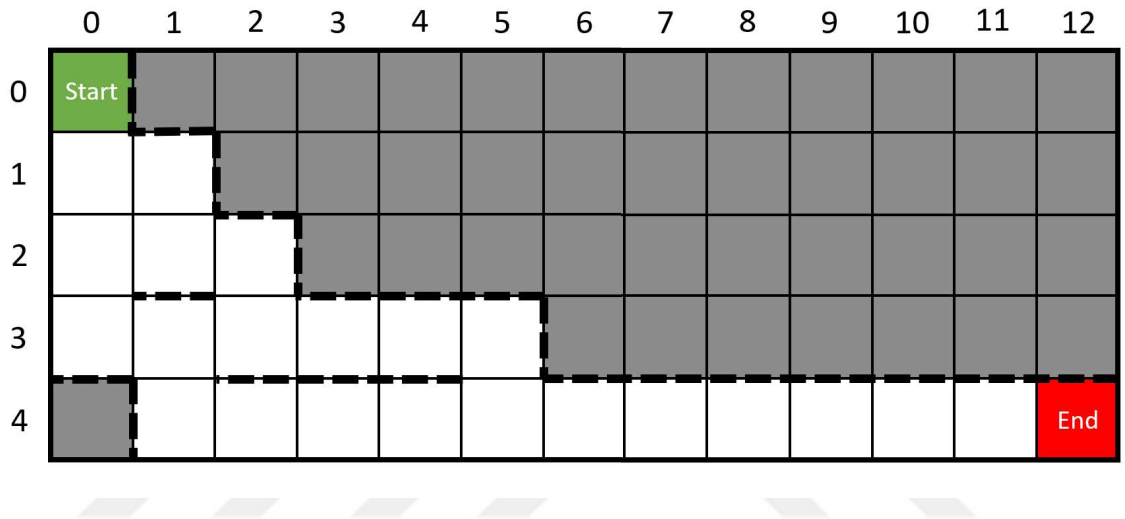

Figure 4.6. Environment of the reduced cluster-based maze game for $\mathbb{P}(16, 12)$.

In the reduced cluster-based construction method, the channels are grouped based on the number of channel combinations they have undergone (see Figure 2.7). This clustering approach relies on the idea that if the channel has undergone more combinations, it will become less reliable for data transmission. The reduced cluster-based maze approach starts with grouping N channels into $\log_2(N)+1$ clusters. Each cluster is denoted as $\mathcal{C}_i$, where i is the number of combinations that the channels have gone through and $0 \leq i \leq \log_2 N$. Tables 4.2 and 4.3 show the channel indices that the clusters contain when $N = 16$ and $N = 64$, respectively. The channel indices per cluster when $N = 128$ and $N = 256$ are given in Appendix A.

After the clusters are formed, depending on the code rate, they are divided into three categories: frozen, information, or the set of interest. Before the maze game starts, the clusters with more channel combinations will be set as frozen and the ones with fewer channel combinations will be set as information. However, there will be at

least one cluster that belongs to the set of interest and the type of the channel (frozen or information) will not be predetermined. Therefore, the maze game will only focus on deciding on the type of the channels found in the set of interest. The rule for setting the channels as frozen or information are explained below.

Table 4.2. Channel indices per cluster when $N = 16$.

Cluster				
$\mathcal{C}_4$	$\mathcal{C}_3$	$\mathcal{C}_2$	$\mathcal{C}_1$	$\mathcal{C}_0$
1	2	4	8	16
	3	6	12	
	5	7	14	
	9	10	15	
		11		
		13		

Let Ω_i denote the number of elements in $\mathcal{C}_i$. The channels in cluster $\mathcal{C}_i$ are set as frozen if

$$i > \left\lfloor \frac{\log_2(N) + 1}{2} \right\rfloor$$

and

$$\Omega_i < N - K - \sum_{j=1}^{\log_2(N)-i} \Omega_{i+j}.$$

The channels in cluster $\mathcal{C}_i$ are predetermined as information if

$$i < \left\lfloor \frac{\log_2(N) + 1}{2} \right\rfloor$$

and

$$\Omega_i < K - \sum_{j=1}^{i} \Omega_{i-j}.$$

Table 4.3. Channel indices per cluster when $N = 64$.

Cluster						
$\mathcal{C}_6$	$\mathcal{C}_5$	$\mathcal{C}_4$	$\mathcal{C}_3$	$\mathcal{C}_2$	$\mathcal{C}_1$	$\mathcal{C}_0$
1	2	4	8	16	32	64
	3	6	12	24	48	
	5	7	14	28	56	
	9	10	15	30	60	
	17	11	20	31	62	
	33	13	22	40	63	
		18	23	44		
		19	26	46		
		21	27	47		
		25	29	52		
		34	36	54		
		35	38	55		
		37	39	58		
		41	42	59		
		49	43	61		
			45			
			50			
			51			
			53			
			57			

If $\mathcal{C}_i$ does not satisfy the conditions above, it is considered as a member of the set of interest. Depending on the code rate and the block length, the set of interest contains at least one and at most two clusters. The number of elements in every cluster for various code lengths is given in Table 4.4. Due to the nature of polar codes, the table forms a Pascal triangle. Thus, the number of channel indices in every cluster for any block length can also be found by looking at the $(\log_2 N)^{th}$ row of the Pascal triangle.

Table 4.4. Number of channel indices in every cluster for several code lengths.

	Cluster										
(N)	$\mathcal{C}_0$	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}_4$	$\mathcal{C}_5$	$\mathcal{C}_6$	$\mathcal{C}_7$	$\mathcal{C}_8$	$\mathcal{C}_9$	$\mathcal{C}_{10}$
(2)	1	1	-	-	-	-	-	-	-	-	-
(4)	1	2	1	-	-	-	-	-	-	-	-
(8)	1	3	3	1	-	-	-	-	-	-	-
(16)	1	4	6	4	1	-	-	-	-	-	-
(32)	1	5	10	10	5	1	-	-	-	-	-
(64)	1	6	15	20	15	6	1	-	-	-	-
(128)	1	7	21	35	35	21	7	1	-	-	-
(256)	1	8	28	56	70	56	28	8	1	-	-
(512)	1	9	36	84	126	126	84	36	9	1	-
(1024)	1	10	45	120	210	252	210	120	45	10	1

For example, for $\mathbb{P}(16, 8)$, $\mathcal{C}_0$ and $\mathcal{C}_1$ satisfy the condition for information, and the channels in those clusters will be set as information channels. $\mathcal{C}_3$ and $\mathcal{C}_4$ meet the condition for being set as frozen. However, $\mathcal{C}_2$ does not satisfy any of the two conditions and therefore it is considered as a member of the set of interest. Therefore, the maze game will only focus on determining the type of those channels. For $\mathbb{P}(16, 12)$, channels found in $\mathcal{C}_0$, $\mathcal{C}_1$, and $\mathcal{C}_2$ will be set as information channels and only the channels in $\mathcal{C}_4$ will be set as a frozen channel. The set of interest will include $\mathcal{C}_3$. When $N = 128$ and $R = 64/128$, channels found in $\mathcal{C}_0$, $\mathcal{C}_1$, and $\mathcal{C}_2$ will be predetermined as information channels. Channels found in $\mathcal{C}_5$, $\mathcal{C}_6$, and $\mathcal{C}_7$ will be determined as frozen channels, and

there will be two clusters ($\mathcal{C}_3$ and $\mathcal{C}_4$) in the set of interest. Notice that the number of clusters found in the set of interest differs for $N = 16$ and $N = 128$.

The FER performance of the constructions obtained by using the reduced cluster-based maze approach is given in Figure 4.7 for several SNR levels. The FER performance of the maze approach in [29] and the polar construction method introduced in [18] are also plotted for comparison. It is important to mention that the method given in [18] has a better performance than both Monte Carlo based construction and the construction method used in [5]. Firstly, when $N = 128$, the polar code constructions obtained by using the maze approach in [29] are very similar to the proposed reduced cluster-based maze approach. When $N = 256$, the maze approach fails to generate a good polar code construction. However, the reduced cluster-based maze approach performs well. Finally, when $N \geq 512$, the performance of the reduced cluster-based maze approach becomes incompetent with [18].

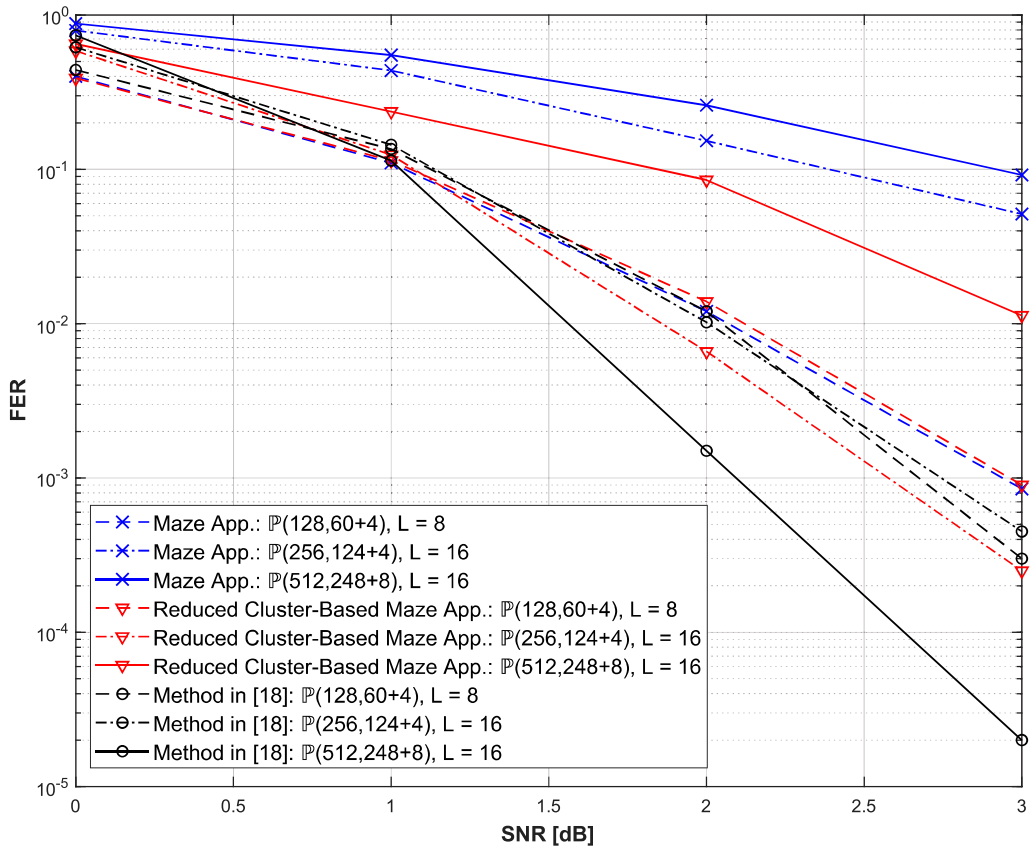

Figure 4.7. FER performance of the reduced cluster-based maze approach.

The reduced cluster-based maze approach not only differs in FER performance (when $N > 128$), but also the total number of episodes, or the number of training samples, needed for convergence varies. The total number of episodes per several block lengths is depicted in Figure 4.8 for both the maze approach and the reduced cluster-based maze approach. As the block length increases, the difference in the total number of episodes also increases. The proposed reduced cluster-based maze approach requires fewer training samples for convergence.

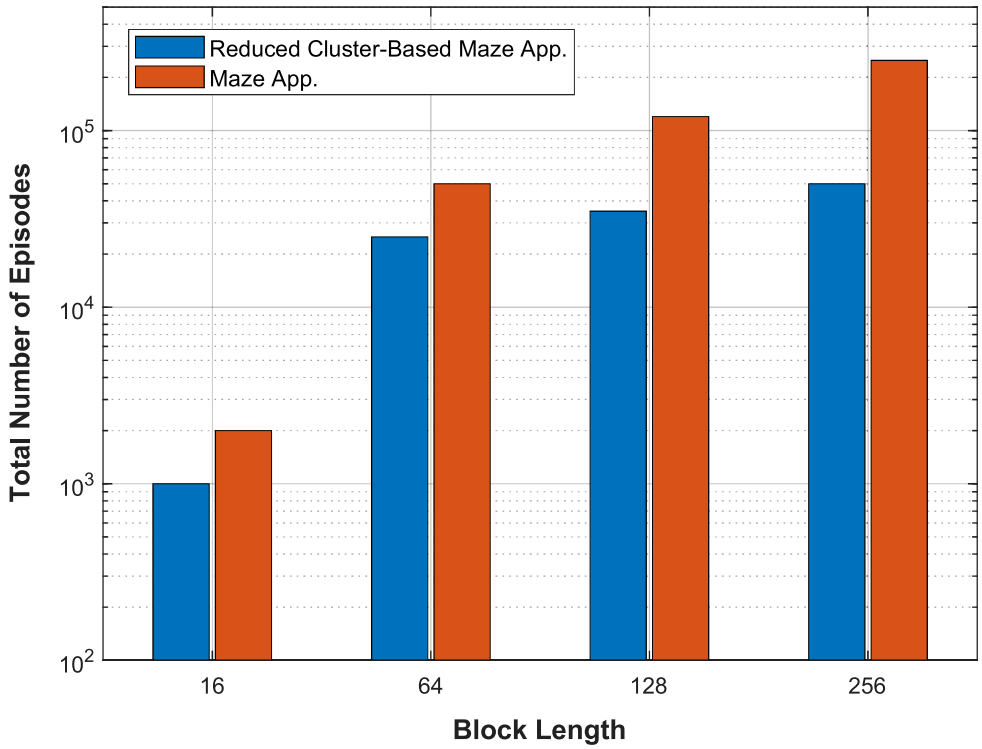

Figure 4.8. Total number of episodes needed for convergence for several block lengths.

4.3. Reduced Cluster-Based Maze Approach with Neighbor Dependency

It is seen that both the maze approach and the reduced cluster-based maze approach fail to generate good performing polar code constructions when $N \geq 512$. Therefore, additional restrictions are introduced to the reduced cluster-based maze approach to decrease the complexity of the maze game. Additional restrictions are applied based on the neighbors of the channels in the set of interest.

Just like in the reduced cluster-based maze approach, the channels are divided into $\log_2(N) + 1$ clusters. Then, clusters are grouped into three categories (frozen, information, or the set of interest) and the set of interest is determined. To reduce the complexity of the maze game, the reduced cluster-based maze approach with neighbor dependency introduces new restrictions and rules. In this approach, the aim is to decrease the number of channels found in the set of interest. To decrease this number, each channel's neighboring channels are examined. If a channel that belongs to the set of interest has a frozen neighbor, it is set as frozen. Likewise, if the channel has a neighbor which is an information channel, it is predetermined as an information channel. However, if one of the neighbors is an information channel and the other one is a frozen channel, or if both of the neighboring channels belong to the set of interest, the channel stays in the set of interest. The implementation of the proposed reduced cluster-based maze approach with neighbor dependency is explained below for several cases.

For example, to construct $\mathbb{P}(512, 256)$, first, 512 channels will be split into 10 clusters. The first 10 rows of the channel indices per clusters table are presented in Table 4.5. Since it is an $R = 256/512$ polar code, $\mathcal{C}_0$, $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ will be considered for information channels, $\mathcal{C}_4$ and $\mathcal{C}_5$ will belong to the set of interest, and the channels in $\mathcal{C}_6$, $\mathcal{C}_7$, $\mathcal{C}_8$, and $\mathcal{C}_9$ will be set as frozen. Thus far, the method given in Section 4.2 is applied. Now, to further reduce the complexity, the channels in the set of interest will be examined. For instance, to determine the type of the first element of $\mathcal{C}_5$ (the 16th channel), the 15th and the 17th channel will be examined. Since they are both frozen channels, the 16th channel will also be set as frozen. Similarly, the 63rd channel will be set as information because its neighbor (the 64th channel) is already predetermined as information. When the 31st channel is considered, it will stay in the set of interest since both of the neighbors (the 30th channel and the 32nd channel) are found in the set of interest. If the neighbor dependent reduction method is applied to all channels found in $\mathcal{C}_4$ and $\mathcal{C}_5$, the elements in the set of interest will decrease from 252 to 112.

Table 4.5. Channel indices per cluster when $N = 512$ (first 10 rows).

Cluster									
$\mathcal{C}_9$	$\mathcal{C}_8$	$\mathcal{C}_7$	$\mathcal{C}_6$	$\mathcal{C}_5$	$\mathcal{C}_4$	$\mathcal{C}_3$	$\mathcal{C}_2$	$\mathcal{C}_1$	$\mathcal{C}_0$
1	2	4	8	16	32	64	128	256	512
	3	6	12	24	48	96	192	384	
	5	7	14	28	56	112	224	448	
	9	10	15	30	60	120	240	480	
	17	11	20	31	62	124	248	496	
	33	13	22	40	63	126	252	504	
	65	18	23	44	80	127	254	508	
	129	19	26	46	88	160	255	510	
	257	21	27	47	92	176	320	511	
		25	29	52	94	184	352		

The performance of the reduced cluster-based maze approach with neighbor dependency for several SNR levels is shown in Figure 4.9. For comparison, the constructed polar codes with the reduced cluster-based maze approach and the method given in [18] are also presented. It is seen that the reduced cluster-based maze approach with neighbor dependency has a close FER performance to the construction method in [18]. When $N = 256$, the reduced cluster-based maze approach and the neighbor dependent maze approach have a very similar FER performance. However, when $N = 512$, the neighbor dependent maze approach outperforms the reduced cluster-based maze approach. As the SNR level increases, the neighbor dependent maze method gets closer to the performance of the construction obtained by using the method in [18] for $N = 512$. After comparing the reduced cluster-based maze approach and the neighbor dependent maze approach, to construct a polar code when $N \leq 256$, the reduced cluster-based maze approach and when $N \geq 512$, the reduced cluster-based maze approach with neighbor dependency can be used.

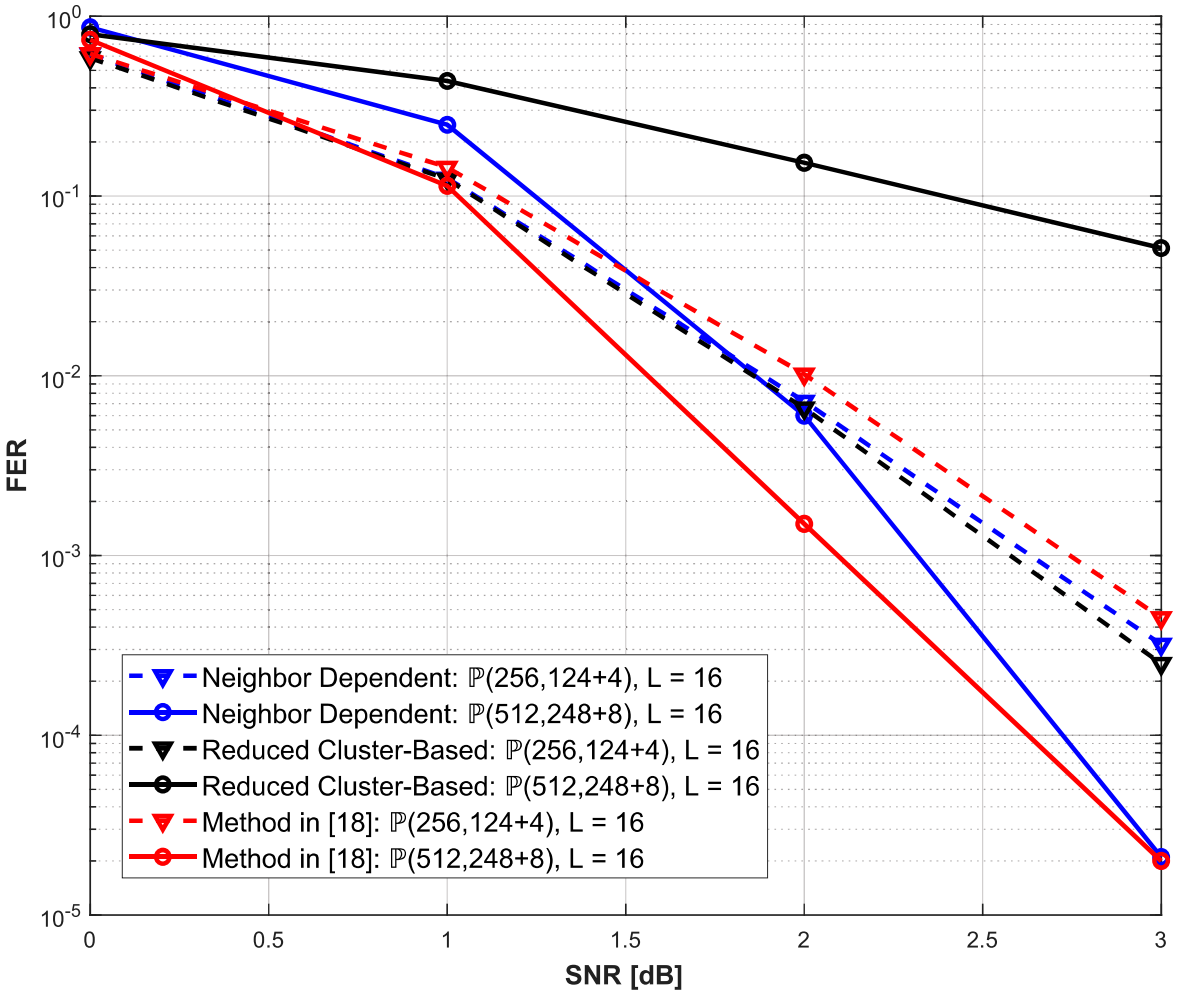

Figure 4.9. FER performance of the reduced cluster-based maze approach with neighbor dependency.

Furthermore, the reduced cluster-based maze approach with neighbor dependency requires fewer training samples than the maze approach. For example, when $N = 512$ for the reduced cluster-based maze approach with neighbor dependency, the total number of episodes, Θ, was chosen as $0.5\text{x}10^5$. However, Θ was chosen as $0.75\text{x}10^5$ and $3\text{x}10^5$, for the reduced cluster-based maze approach and the maze approach, respectively. To sum up, the reduced cluster-based maze approach with neighbor dependency constructs much more reliable codes with fewer training samples when compared to other maze approaches.

5. CONCLUSION

Polar codes gained popularity over time due to their promising error performance. In the literature, many construction and decoding methods are proposed to improve the error performance of the polar codes. A good performing polar code construction technique with an RL based maze game approach is proposed for short block lengths. However, it performs poorly for longer block lengths and cannot construct reliable codes due to the high complexity. In this thesis, the error performance of a reinforcement learning-based polar code construction method is improved for longer block lengths ($N > 128$). Several methods are proposed to increase the performance of the constructed codes while decreasing the complexity.

First, a reduced cluster-based maze approach is proposed to reduce the complexity of the maze game and to construct reliable codes for $N > 128$. When $N \leq 128$, the performance of the proposed reduced cluster-based maze approach is very similar to the performance of both the maze approach and other polar code construction methods found in the literature. It is shown that the reduced cluster-based approach outperformed the maze approach method when $N = 256$ and also had a similar FER performance to other construction methods. However, the reduced cluster-based maze approach failed to construct reliable codes when $N > 512$ due to the increased complexity. To achieve the complexity problem, further restrictions are applied to the maze game and the reduced cluster-based maze approach with neighbor dependency is proposed. It is demonstrated that the reduced cluster-based maze approach with neighbor dependency constructs reliable codes when $N \leq 512$. Moreover, it is shown that it has a close FER performance to the construction methods given in the literature. Therefore, it can be said that the reduced cluster-based maze approach with neighbor dependency is an efficient polar code construction method for block lengths smaller than 1024.

As the block length increases, the complexity of the system and the size of the Q-table and the eligibility trace table increase. For polar codes with higher block lengths ($N > 512$), the traditional RL algorithms perform weakly. They cannot achieve the goal despite increasing the number of episodes in the learning process. Therefore, as future work, the polar code construction problem can be modeled with deep reinforcement learning methods such as deep Q-learning.

In addition to the deep learning-based polar code construction approaches, the position of the CRC bits in the sequence can be changed. Since RL methods require feedback to find the best path, separation of CRC bits along the sequence may be beneficial for polar code construction, especially for higher block lengths. This separation will let the agent get rewards earlier and therefore the convergence to a single path will be faster. Also, changing the position of CRC bits will increase the frequency of the rewards and thus the FER performance may also be improved.

REFERENCES

1. Arıkan, E., "Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels", *IEEE Transactions on Information Theory*, Vol. 55, No. 7, pp. 3051–3073, 2009.

2. Arıkan, E., "A Performance Comparison of Polar Codes and Reed-Muller Codes", *IEEE Communications Letters*, Vol. 12, No. 6, pp. 447–449, 2008.

3. Mori, R. and T. Tanaka, "Performance of Polar Codes with the Construction Using Density Evolution", *IEEE Communications Letters*, Vol. 13, No. 7, pp. 519–521, 2009.

4. Pedarsani, R., S. H. Hassani, I. Tal and E. Telatar, "On the Construction of Polar Codes", *IEEE International Symposium on Information Theory Proceedings*, pp. 11–15, 2011.

5. Tal, I. and A. Vardy, "How to Construct Polar Codes", *IEEE Transactions on Information Theory*, Vol. 59, No. 10, pp. 6562–6582, 2013.

6. Trifonov, P., "Efficient Design and Decoding of Polar Codes", *IEEE Transactions on Communications*, Vol. 60, No. 11, pp. 3221–3227, 2012.

7. Wu, D., Y. Li and Y. Sun, "Construction and Block Error Rate Analysis of Polar Codes over AWGN Channel Based on Gaussian Approximation", *IEEE Communications Letters*, Vol. 18, No. 7, pp. 1099–1102, 2014.

8. Şaşoğlu, E., E. Telatar and E. Arıkan, "Polarization for Arbitrary Discrete Memoryless Channels", *IEEE Information Theory Workshop*, pp. 144–148, 2009.

9. Gulcu, T. C., M. Ye and A. Barg, "Construction of Polar Codes for Arbitrary Discrete Memoryless Channels", *IEEE Transactions on Information Theory*, Vol. 64,

No. 1, pp. 309–321, 2017.

10. Hussami, N., S. B. Korada and R. Urbanke, "Performance of Polar Codes for Channel and Source Coding", *IEEE International Symposium on Information Theory*, pp. 1488–1492, 2009.

11. Yuan, B. and K. K. Parhi, "Architecture Optimizations for BP Polar Decoders", *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 2654–2658, 2013.

12. Goela, N., S. B. Korada and M. Gastpar, "On LP Decoding of Polar Codes", *IEEE Information Theory Workshop*, pp. 1–5, 2010.

13. Tal, I. and A. Vardy, "List Decoding of Polar Codes", *IEEE Transactions on Information Theory*, Vol. 61, No. 5, pp. 2213–2226, 2015.

14. Niu, K. and K. Chen, "CRC-Aided Decoding of Polar Codes", *IEEE Communications Letters*, Vol. 16, No. 10, pp. 1668–1671, 2012.

15. Balatsoukas-Stimming, A., M. B. Parizi and A. Burg, "LLR-Based Successive Cancellation List Decoding of Polar Codes", *IEEE Transactions on Signal Processing*, Vol. 63, No. 19, pp. 5165–5179, 2015.

16. Niu, K. and K. Chen, "Stack Decoding of Polar Codes", *Electronics Letters*, Vol. 48, No. 12, pp. 695–697, 2012.

17. Jia, X., F. Wang, Y. Sun and S. Zhang, "A Novel Modulation Scheme of Polar Codes", *14th International Wireless Communications & Mobile Computing Conference*, pp. 1385–1390, 2018.

18. Chiu, M.-C., "Analysis and Design of Polar-Coded Modulation", *IEEE Transactions on Communications*, Vol. 70, No. 3, pp. 1508–1521, 2022.

19. Bravo-Santos, A., "Polar Codes for the Rayleigh Fading Channel", *IEEE Communications Letters*, Vol. 17, No. 12, pp. 2352–2355, 2013.

20. Niu, K. and Y. Li, "Polar Codes for Fast Fading Channel: Design Based on Polar Spectrum", *IEEE Transactions on Vehicular Technology*, Vol. 69, No. 9, pp. 10103–10114, 2020.

21. Aydoğan, O., *Polar Codes and Their Performance in Satellite Communication*, Master's Thesis, Istanbul Technical University, 2022.

22. Zhou, D., K. Niu and C. Dong, "Construction of Polar Codes in Rayleigh Fading Channel", *IEEE Communications Letters*, Vol. 23, No. 3, pp. 402–405, 2019.

23. Gruber, T., S. Cammerer, J. Hoydis and S. ten Brink, "On Deep Learning-Based Channel Decoding", *51st Annual Conference on Information Sciences and Systems*, pp. 1–6, 2017.

24. Cammerer, S., T. Gruber, J. Hoydis and S. Ten Brink, "Scaling Deep Learning-Based Decoding of Polar Codes via Partitioning", *IEEE Global Communications Conference*, pp. 1–6, 2017.

25. Doan, N., S. A. Hashemi and W. J. Gross, "Decoding Polar Codes with Reinforcement Learning", *IEEE Global Communications Conference*, pp. 1–6, 2020.

26. Habib, S., A. Beemer and J. Kliewer, "Belief Propagation Decoding of Short Graph-Based Channel Codes via Reinforcement Learning", *IEEE Journal on Selected Areas in Information Theory*, Vol. 2, No. 2, pp. 627–640, 2021.

27. Elkelesh, A., M. Ebada, S. Cammerer and S. Ten Brink, "Decoder-Tailored Polar Code Design Using the Genetic Algorithm", *IEEE Transactions on Communications*, Vol. 67, No. 7, pp. 4521–4534, 2019.

28. Huang, L., H. Zhang, R. Li, Y. Ge and J. Wang, "AI Coding: Learning to Construct

Error Correction Codes", *IEEE Transactions on Communications*, Vol. 68, No. 1, pp. 26–39, 2019.

29. Liao, Y., S. A. Hashemi, J. Cioffi and A. Goldsmith, "Construction of Polar Codes with Reinforcement Learning", *IEEE Global Communications Conference*, pp. 1–6, 2020.

30. Irawan, A., G. Witjaksono and W. K. Wibowo, "Deep Learning for Polar Codes over Flat Fading Channels", *International Conference on Artificial Intelligence in Information and Communication*, pp. 488–491, 2019.

31. Lin, S. and D. J. Costello, *Error Control Coding*, Prentice Hall New York, 2001.

32. ETSI, *5G; NR; Multiplexing and Channel Coding (V15.2.0 Release 15)*, TS 138 212, 2018.

33. Shannon, C. E., "A Mathematical Theory of Communication", *The Bell System Technical Journal*, Vol. 27, No. 3, pp. 379–423, 1948.

34. Ash, R. B., *Information Theory*, Courier Corporation, 2012.

35. Goldsmith, A., *Wireless Communications*, Cambridge University Press, 2005.

36. Vangala, H., E. Viterbo and Y. Hong, "A Comparative Study of Polar Code Constructions for the AWGN Channel", *ArXiv Preprint ArXiv:1501.02473*, 2015.

37. Dizdar, O. and E. Arıkan, "A High-Throughput Energy-Efficient Implementation of Successive Cancellation Decoder for Polar Codes Using Combinational Logic", *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 63, No. 3, pp. 436–447, 2016.

38. Condo, C., V. Bioglio and I. Land, "Generalized Fast Decoding of Polar Codes", *IEEE Global Communications Conference*, pp. 1–6, 2018.

39. Moor, J., "The Dartmouth College Artificial Intelligence Conference: The Next Fifty Years", *AI Magazine*, Vol. 27, No. 4, pp. 87–91, 2006.

40. Sutton, R. S. and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 2018.

41. Puterman, M. L., *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, John Wiley & Sons, 2014.

APPENDIX A: CHANNEL INDICES PER CLUSTER WHEN $N = 128$ AND $N = 256$

Table A.1. Channel indices per cluster when $N = 128$.

Cluster							
$\mathcal{C}_7$	$\mathcal{C}_6$	$\mathcal{C}_5$	$\mathcal{C}_4$	$\mathcal{C}_3$	$\mathcal{C}_2$	$\mathcal{C}_1$	$\mathcal{C}_0$
1	2	4	8	16	32	64	128
	3	6	12	24	48	96	
	5	7	14	28	56	112	
	9	10	15	30	60	120	
	17	11	20	31	62	124	
	33	13	22	40	63	126	
	65	18	23	44	80	127	
		19	26	46	88		
		21	27	47	92		
		25	29	52	94		
		34	36	54	95		
		35	38	55	104		
		37	39	58	108		
		41	42	59	110		
		49	43	61	111		
		66	45	72	116		
		67	50	76	118		
		69	51	78	119		
		73	53	79	122		
		81	57	84	123		
		97	68	86	125		
			70	87			
			71	90			

Table A.1. Channel indices per cluster when $N = 128$. (cont.)

Cluster							
$\mathcal{C}_7$	$\mathcal{C}_6$	$\mathcal{C}_5$	$\mathcal{C}_4$	$\mathcal{C}_3$	$\mathcal{C}_2$	$\mathcal{C}_1$	$\mathcal{C}_0$
			74	91			
			75	93			
			77	100			
			82	102			
			83	103			
			85	106			
			89	107			
			98	109			
			99	114			
			101	115			
			105	117			
			113	121			

Table A.2. Channel indices per cluster when $N = 256$.

Cluster								
$\mathcal{C}_8$	$\mathcal{C}_7$	$\mathcal{C}_6$	$\mathcal{C}_5$	$\mathcal{C}_4$	$\mathcal{C}_3$	$\mathcal{C}_2$	$\mathcal{C}_1$	$\mathcal{C}_0$
1	2	4	8	16	32	64	128	256
	3	6	12	24	48	96	192	
	5	7	14	28	56	112	224	
	9	10	15	30	60	120	240	
	17	11	20	31	62	124	248	
	33	13	22	40	63	126	252	
	65	18	23	44	80	127	254	
	129	19	26	46	88	160	255	
		21	27	47	92	176		
		25	29	52	94	184		
		34	36	54	95	188		
		35	38	55	104	190		
		37	39	58	108	191		
		41	42	59	110	208		
		49	43	61	111	216		
		66	45	72	116	220		
		67	50	76	118	222		
		69	51	78	119	223		
		73	53	79	122	232		
		81	57	84	123	236		
		97	68	86	125	238		
		130	70	87	144	239		
		131	71	90	152	244		
		133	74	91	156	246		
		137	75	93	158	247		
		145	77	100	159	250		

Table A.2. Channel indices per cluster when $N = 256$. (cont.)

Cluster								
$\mathcal{C}_8$	$\mathcal{C}_7$	$\mathcal{C}_6$	$\mathcal{C}_5$	$\mathcal{C}_4$	$\mathcal{C}_3$	$\mathcal{C}_2$	$\mathcal{C}_1$	$\mathcal{C}_0$
		161	82	102	168	251		
		193	83	103	172	253		
			85	106	174			
			89	107	175			
			98	109	180			
			99	114	182			
			101	115	183			
			105	117	186			
			113	121	187			
			132	136	189			
			134	140	200			
			135	142	204			
			138	143	206			
			139	148	207			
			141	150	212			
			146	151	214			
			147	154	215			
			149	155	218			
			153	157	219			
			162	164	221			
			163	166	228			
			165	167	230			
			169	170	231			
			177	171	234			
			194	173	235			
			195	178	237			

Table A.2. Channel indices per cluster when $N = 256$. (cont.)

Cluster								
$\mathcal{C}_8$	$\mathcal{C}_7$	$\mathcal{C}_6$	$\mathcal{C}_5$	$\mathcal{C}_4$	$\mathcal{C}_3$	$\mathcal{C}_2$	$\mathcal{C}_1$	$\mathcal{C}_0$
			197	179	242			
			201	181	243			
			209	185	245			
			225	196	249			
				198				
				199				
				202				
				203				
				205				
				210				
				211				
				213				
				217				
				226				
				227				
				229				
				233				
				241				