Universidad de Alcalá

Escuela Politécnica Superior

Grado en Ingeniería en Sistemas de Telecomunicación

Trabajo Fin de Grado

Evaluation of Channel Coding Methods for Next Generation Mobile Communication Standards

Autor: Xiaoshen Li

Tutor/es: Francisco Javier Escribano Aparicio

2021

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería en Sistemas de Telecomunicación

Trabajo Fin de Grado

Evaluation of Channel Coding Methods for Next Generation Mobile Communication Standards

Autor: Xiaoshen Li

Tutor: Francisco Javier Escribano Aparicio

TRIBUNAL:

Presidente: José Antonio Portilla Figueras

Vocal 1º: Fernando Cruz Roldán

Vocal 2º: Francisco Javier Escribano Aparicio

FECHA: 21 / 09 / 2021

Contents

Resumen

La codificación de canales es crucial para los sistemas de comunicación móvil, y los sistemas de comunicación inalámbrica 5G han decidido utilizar los códigos LDPC como esquema de codificación para sus canales de datos y los códigos Polares como esquema de codificación para sus canales de control. Este estudio se centra en los fundamentos de los códigos LDPC y los códigos Polares, especialmente los nuevos códigos polares, explicando en detalle sus características de polarización y las técnicas de decodificación recursiva. También se estudia las especificaciones de diseño relacionadas con estos dos esquemas de codificación de canales en 5G. Mediante simulaciones, se compara el rendimiento del nuevo esquema de codificación de canales inalámbricos 5G con el de los códigos Turbo a diferentes longitudes de bloque y tasas de código, y se extraen conclusiones relevantes para demostrar la aplicabilidad del esquema de codificación de canales 5G NR.

Palabra Clave

5G NR; Codificación de canal; Códigos LDPC; Códigos Polares; Códigos Trubo.

Abstract

Channel coding is essential for mobile communication systems, and the 5G wireless standardization committees decided to use LDPC codes as the coding scheme of its data channel and Polar codes as the coding scheme of its control channel. This study focuses on the fundamentals of LDPC codes and Polar codes, especially the emerging Polar codes, with detailed explanations of their polarization characteristics and recursive decoding techniques. It is also focused on the design specification related to these two channel coding schemes in 5G. The performance of the 5G New Radio channel coding scheme is compared with that of LTE Turbo codes at different block lengths and code rates through simulations, and relevant conclusions are drawn to demonstrate the suitability of the 5G NR channel coding scheme.

Index Terms

5G NR; channel coding; LPDC codes; Polar codes; Turbo codes.

List of Figures and Tables

Figures

Tables

Abbreviations and acronyms

3GPP ..... *3rd Generation Partnership Project*
3GPP2 ..... *3rd Generation Partnership Project 2*
AMPS..... *Advanced Mobile Phone System*
APP..... *a Posteriori Probability*
BCH ..... *Bose–Chaudhuri–Hocquenghem codes*
B-DMC..... *binary-input discrete memoryless channel*
BEC..... *Binary Erasure Channel*
BF ..... *Bit-Flipping*
BG ..... *Base graph*
BI-AWGN..... *Binary Input-Additive White Gaussian Noise*
BSC..... *Binary Symmetric Channel*
CC..... *Convolutional code*
DE..... *Decision Element*
DECT..... *Digital Enhanced Cordless Telecommunications*
EDGE ..... *Enhanced Data rates for GSM Evolution*
eMBB ..... *Enhanced Mobile Broadband*
FDMA ..... *Frequency-Division Multiple Access*
FEC ..... *Forward Error Correction*
GSM ..... *Global System for Mobile Communications*
i.i.d ..... *independently identically distributed*
IDBP ..... *Iterative Decoding based on Belief Propagation*
IMT..... *International Mobile Telecommunications*
IoT ..... *Internet of Things*
IRA..... *Irregular Repeat Accumulate Codes*
IR-HARQ..... *Incremental Redundancy-Hybrid Automatic Repeat Request*
ITU..... *International Telecommunication Union*
ITU-R ..... *ITU- Radiocommunication Sector*
LDPC..... *low-density parity-check*
LLR..... *Log-Likelihood Ratio*
MAP ..... *Maximum a Posteriori Probability*
ML ..... *Maximum Likelihood*
MLG..... *Majority-Logic*
mMTC ..... *Massive Machine Type Communications*
NOMA ..... *Non-Orthogonal Multiple Access*
NR ..... *New Radio*
OFDMA ..... *Orthogonal Frequency Division Multiple Access*
PDC ..... *Personal Digital Cellular*
SC ..... *Successive Cancellation*
SC-FDMA..... *Single-Carrier Frequency Division Multiple Access*
SCL..... *Successive Cancellation List*

SCMA ..... *Sparse Code Multiple Access*
SPA ..... *Sum-Product Algorithm*
SPC ..... *Single Parity-Check*
TBCC ..... *Tail-Biting Convolutional Code*
UCI ..... *Uplink Control Information*
UMTS ..... *Universal Mobile Telecommunications System*
URLLC ..... *Ultra-Reliable and Low-Latency Communications*
WARC ..... *World Administrative Radio Conference*
WBF ..... *Weight BF*
WCDMA ..... *Wideband Code Division Multiple Access*
WiMax ..... *Worldwide Interoperability for Microwave Access*

1. Introduction

With the 4G era, the experience of high-speed data and streaming media has inspired people to imagine more possibilities for mobile communication networks. With the development of the cloud, big data, artificial intelligence, blockchain, and a host of other technologies, people want to create a network system that digitizes the real world, but the old mobile communication system is not able to meet these needs, people need a new network with a higher speed, lower latency, high reliability, so 5G was born. Like the glue that holds these different technologies together so that they can be used in different scenarios. Nowadays, the 5G era has arrived and is gaining popularity, and is subtly changing our lives.

To this end, the ITU-Radiocommunication Sector (ITU-R) and 3GPP have defined a series of new specifications for 5G New Radio (5G NR) that will enable 5G NR to meet the needs of different scenarios. Although 5G NR builds on the technology of previous generations of mobile networks, in terms of channel coding, 5G NR uses a completely new channel coding scheme, i.e., LDPC codes are designated as the coding scheme for the data channel and Polar codes are designated as the coding scheme for the control channel.

Therefore, this report aims to clarify why these two channel codes stand out from other channel codes. We focus on [1-4] to learn about the low-density parity-check codes (LDPC), their basic definitions, decoding algorithms, and the QC-LDPC codes constructed by their regular structure. As for Polar codes, we focus on the studies of their inventor Erdal Arikan [5-7] to understand their polarization characteristics and recursive decoding algorithms. Finally, according to [8-9], we can understand the definitions and specifications of these two channel codes in the 5G NR.

In order to accomplish this goal, it is first necessary to understand the history of mobile communication systems. The evolution of mobile communication systems from 2G to 5G can also be described as the evolution of channel coding, i.e., the constant search for a channel coding that can reach the Shannon limit. Therefore, Chapter 2 will present the standers of different generations of communication systems and the history of these related channel codes. After understanding the relevant background, Chapter 3 will focus on learning the theoretical knowledge of LDPC codes and Polar codes, their structural design, coding and decoding algorithms, etc., and their specifications by 3GPP in 5G NR. Finally, in Chapters 4 and 5, the simulation results obtained using Matlab will be evaluated to demonstrate the advantages or disadvantages of these two channel codes, i.e., why 3GPP chose them in 5G NR.

2. Historical background of mobile communication systems and channel coding

2.1 The evolution of mobile communication systems

Mobile communication technology means that people can accomplish the exchanging information with each other (mobile terminal to mobile terminal or mobile terminal to fixed terminal) through a variety of different technologies without relying on a physical connection (fiber or cable). As shown in Figure 2-1, it can be seen that mobile communication systems have evolved rapidly over four decades, from the initial analog voice to today's Internet of Things (IoT).

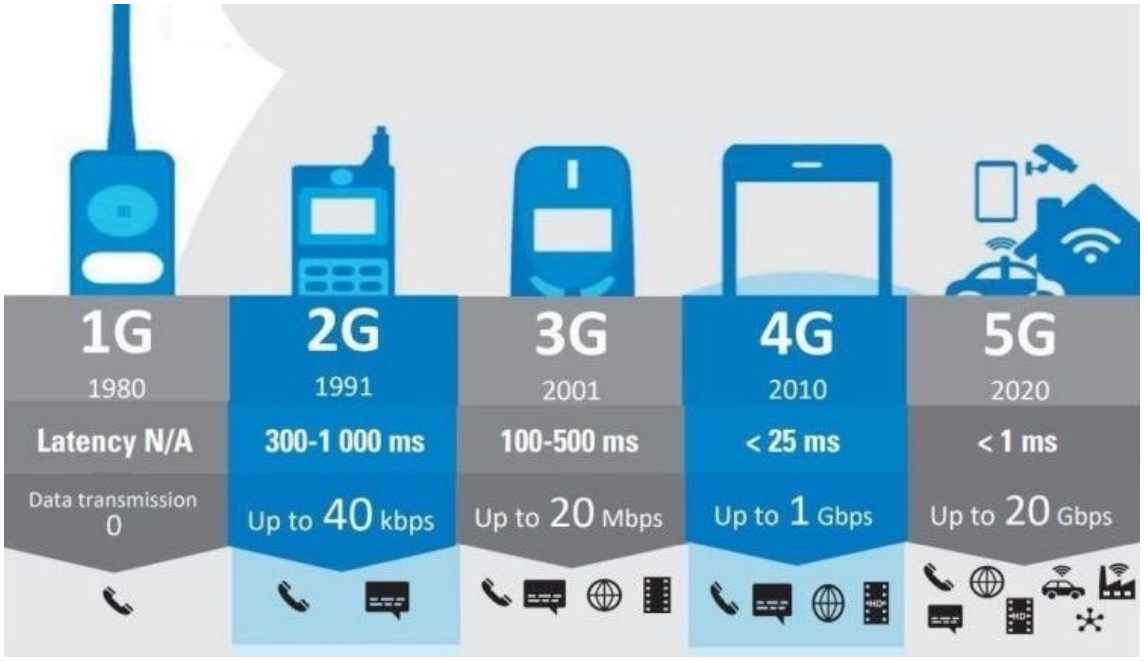


Figure 2-1 Evaluation of mobile telecommunication systems[1]

2.1.1 The First Generation (1G) of Mobile Telecommunications

In 1976, the World Administrative Radio Conference (WARC) approved frequency allocations for cellular phones in the 800/900 MHz band. In 1979, Nippon Telephone and Telegraph (NTT) went into operation and was the first analog cellular system to be launched commercially. In 1981, Nordic Mobile Telephony (NMT) went into service in Sweden and Norway. In 1983, the Advanced Mobile Phone System (AMPS) was first deployed in the United States [10].

[1] retrieved from: https://cutt.ly/zQyFiEd

	NTT	NMT	AMPS
Frequency band UL/DL[2] (MHz)	925–940/870–885	453-457.5/463-467.5	824–849/869–894
	915–918.5/860–863.5	890–915/917–950	
	922–925/867–870		
Channel Bandwidth (kHz)	25/6.25	25	30
	6.25	25	
	6.25		
Modulation	Analog FM	Analog FM	Analog FM

Table 2-1 The main characteristics of 1G cellular standards [10]

The characteristics of each standard are shown in Table 2-1. Among them, 1G used analog signals, and the multiple access technology was Frequency Division Multiple Access (FDMA), which only provided voice services. Thus, it can be seen that the channel bandwidth of these three standards was very large by design, with a minimum of 6.25KHz and a maximum of 30KHz. All three standards operated in the frequency band specified by WARC. However, the analog voice signal was not compressed, and the voice information was not protected by error detection and correction. Therefore, the resource utilization rate was low, the system capacity was small, and it was characterized by poor communication quality. At the same time, because the devices were difficult to integrate, the hardware cost of the terminal was high, the price was high, and the terminals were large and heavy. Today, all 1G cellular systems have become history.

2.1.2 The Second Generation (2G) of Mobile Telecommunications

The second-generation (2G) digital cellular systems were developed in the 1980s and early 1990s, which included the Global System for Mobile Communications (GSM) standard in Europe, the Japanese Personal Digital Cellular (PDC) standard, and the American IS-54 / 136, IS-95 standards [10].

The characteristics of each standard are summarized in Table 2-2, and, unlike 1G, 2G used digital signals. As mentioned in the previous paragraph, since a different

[2] UL: uplink; DL: downlink.

country or region developed each standard, different frequency bands, access methods, etc., were used.

The GSM standard had several different frequency bands, of which DCS1800 and PCS1900 were also denominated GSM1800 and GSM1900. The bandwidth of each channel had been increased to 200 KHz, and, by using TDMA technology, eight simultaneous calls were allowed on the same frequency [11].

While the GSM standard was being developed in Europe, the United States was developing its own IS standard. IS-54 standard and IS-136 standard were similar. The only difference was the control channel: the IS-54 standard used an analog control channel while the other used a digital control channel [10]. It should be noted that the IS-95 standard, based on CDMA technology, was proposed by Qualcomm [10]. In addition, the bandwidth of each channel was six times greater than that of GSM. Based on these advantages, the IS-95 standard could offer higher call quality and higher network capacity.

Examining the typical standard characteristics, it can be seen that the transmission efficiency, system capacity, and communication quality had been improved thanks to the new multiple access techniques, modulation, larger bandwidth, and channel coding techniques. Although these standards used BCH (Bose-Chaudhuri-Hocquenghem codes) and CC (Convolutional codes) coding, the performance is not very high, as shown in Figure 2-2.

	GSM	PDC	IS-54/136	IS-95
Frequency band UL/DL (MHz)	GSM: 890-915 / 935-960	810–826/ 940–956	824–829/ 869/894	824–829/ 869–894
	DCS1800: 1710-1785/ 1805-1880	1429–1453/ 1477–1501	1930–1990/ 1850–1910	1930–1990/ 1850–1910
	PCS1900: 1930-1990/ 1850-1910			
Multiple Access	F/TDMA	F/TDMA	F/TDMA	F/CDMA
Channel Bandwidth (kHz)	200	25	30	1250
Modulation	GMSK	π/4-DQPSK	π/4-DQPSK	QPSK
Channel coding	Rate= 1/2 CC	Rate= 1/2 BCH	Rate= 1/2 CC	UL: rate= 1/2 CC
				DL: rate= 1/3 CC

Table 2-2 The main characteristics of 2G cellular standards [10]

2.1.3 The Third Generation (3G) of Mobile Telecommunications

In March 1992, WARC defined the worldwide spectrum allocation for the 1885-2200 MHz band to support International Mobile Telecommunications-2000 (IMT-2000). The IMT-2000 standard contains a variety of different standards, two of which are based on TDMA technology: *Enhanced Data rates for GSM Evolution* (EDGE) and *Digital Enhanced Cordless Telecommunications* (DECT). However, it is widely considered that EDGE is a 2.5G network, not a 3G network. There are two widely used standards in 3G: cdma2000, developed by 3GPP2; the other is the Universal Mobile Telecommunications System (UMTS), a series of standards developed by 3GPP. Since Wideband Code Division Multiple Access (WCDMA) is the primary interface technology used in UMTS, it was generally for operators to use the denomination WCDMA in their announcements rather than UMTS. The Worldwide Interoperability for Microwave Access (WiMax), developed by the IEEE 802.16 working group, was also one of the IMT-2000 standards. However, for commercial purposes, this standard is considered a 4G standard rather than a 3.5G standard [10].

The characteristics of each standard are shown in Table 2-3. As in the case of previous generations, there are only slight differences between them. However, WCDMA is more efficient than cdma2000 in terms of transmission speed since the channel bandwidth of WCDMA is 5MHz compared to 1.25MHz for cdma2000 [10].

The rapid development of 3G networks is mainly due to CDMA technology, power control techniques, and Turbo coding. In particular, the breakthrough in channel coding in 1993 allowed the throughput of wireless links to approach the Shannon capacity limit [1]. Therefore, in both standards, CC codes and Turbo codes were used.

	WCDMA (UMTS)	cdma2000
Multiple Access	DS-CDMA	DS-CDMA
Channel Bandwidth (MHz)	5	1.25
Modulation	UL: QPSK	UL: QPSK / BPSK
	DL: BPSK	DL: BPSK
Channel coding [12], [13] (K: constraint length)	Rate= 1/2, 1/3 K=9 CC	Rate= 1/2, 1/3, 1/4, 1/6 K=9 CC
	Rate= 1/3 K[3] =4 Turbo Codes	Rate= 1/3, 1/4, 1/5 K=4 Turbo Codes

Table 2-3 The main characteristics of 3G cellular standards [10]

[3] It is a value of one constituent encoder.

2.1.4 The Fourth Generation (4G) of Mobile Telecommunications

As with the other generations, the International Telecommunication Union (ITU) had developed a series of standards for 4G, named IMT-Advanced. Although LTE and WiMax do not meet the technical requirements of the IMT-Advanced standard, they are the basis for later versions of LTE-A and WiMax Release 2 and, therefore, are categorized as 4G standards. From some points of view, it could be considered that 4G mobile telecommunication was a further development of 3G.

As presented in Table 2-4, unlike 3G, 4G used orthogonal frequency division multiple access (OFDMA) and single-carrier frequency division multiple access (SC-FDMA) technologies. Because of these new multiple access technologies, 3GPP specified a subcarrier spacing of 15KHz in the 4G standard [14] to balance system performance and anti-interference capability [15]. In addition, 3GPP had higher requirements for the maximum data rate [10], so the channel bandwidth of LTE and LTE-A standards is more extensive and is used with more flexibility.

In terms of channel coding, both LTE and LTE-A used the Turbo codes already employed by 3G as the FEC (Forward Error Correction) system for the data channel. Both standards specified the use of Turbo codes in the data channel and Tail-Biting Convolutional Codes (TBCC) in the control channel [16].

	LTE	LTE-A
Multiple Access	UL: OFDMA	UL: OFDMA
	DL: SC-FDMA	DL: SC-FDMA
Channel Bandwidth (MHz)	1.4, 3, 5, 10, 15, 20	In addition, it supports downlinks up to 100 MHz and uplinks up to 40 MHz with carrier aggregation.
Subcarrier spacing (kHz)	15	
Modulation	UL: QPSK, 16QAM, 64QAM	UL: QPSK, 16QAM, 64QAM, 256QAM
	DL: QPSK, 16QAM, 64QAM (optional)	DL: QPSK, 16QAM, 64QAM (optional)
Channel coding	Turbo Codes	Turbo Codes
	TBCC	TBCC

Table 2-4 The main characteristics of 4G cellular standards [1]

2.1.5 The Fifth Generation (5G) of Mobile Telecommunications

In early 2012, the ITU-R started developing new IMT standards for the next generation of mobile telecommunications. In 2015, it published the IMT-2020 standard. It proposed three main application areas: enhanced mobile broadband (eMBB), massive machine-type communications (MMTC), and ultra-reliable and low-latency communication (URLLC) [17-18].

As shown in Table 2-5, unlike the previous four generations, the application areas of 5G are very diverse, so significant changes have been made in 5G cellular systems to meet those needs, such as using Polar Codes in the control channel and LDPC codes in the data channel. In addition to OFDMA, other new access technologies such as Non-Orthogonal Multiple Access (NOMA) and Sparse Code Multiple Access (SCMA) are also used [19].

Another unique feature of 5G NR is spectrum management. It can be found that 5G NR uses two different frequency ranges (FR1 and FR2). The frequency range FR1 is designed to match and improve 4G cellular networks, as it contains almost all the frequency bands and channel bandwidth of LTE and LTE-A. On the other hand, the frequency range FR2 contains the extremely high-frequency range (24.25-52.6 GHz), so it is sometimes referred to as Millimeter Wave (mmWave), and it will mainly provide high-speed data transmission over short distances because high frequencies always have a higher loss in wireless link propagation and require line-of-sight (LOS) links [20].

The scalable OFDM numerology is designed to vary with the channel bandwidth, so, unlike 4G, which specified a subcarrier spacing of 15kHz, 5G NR specifies several different subcarrier spacings (15KHz, 30 kHz, 60 kHz, 120 kHz, 240 kHz) to accommodate different scenarios and applications of 5G NR [21].

	5G NR	
Frequency range (FR) (MHz)	FR1: 410-7125	FR2: 24 250-52 600
Channel bandwidth (MHz)	5, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90, 100	50, 100, 200, 400
Subcarrier spacing (kHz)	15, 30, 60	60, 120, 240
Multiple Access	OFDM, NOMA (optional), SCMA (optional)	
Modulation	QPSK, 16QAM, 64QAM, 256QAM	
Channel coding	LDPC codes	
	Polar codes	

Table 2-5 The main characteristics of 5G NR cellular standards [19,22]

2.2 Channel coding

As shown in Figure 2-2, it can be seen that these channel coding systems, from right to left, are constantly approaching the Shannon limit. According to [23], the BCH codes used in 2G were 5.7dB away from the Shannon limit, the CC codes used in the 2G and 3G era have a minimum distance of 1.75dB from the Shannon limit, and the Turbo codes used in the 3G and 4G era are only 0.7dB away from the Shannon limit. The LDPC codes since then have reduced the distance to 0.0045dB [24], while the Polar codes have been proven to completely touch the Shannon limit [16].

It is the continuous improvement in the performance of these channel coding systems that allow mobile communication systems to meet the increasing demands of people. With the discovery of Polar codes, a new chapter in mobile communication systems has been opened.

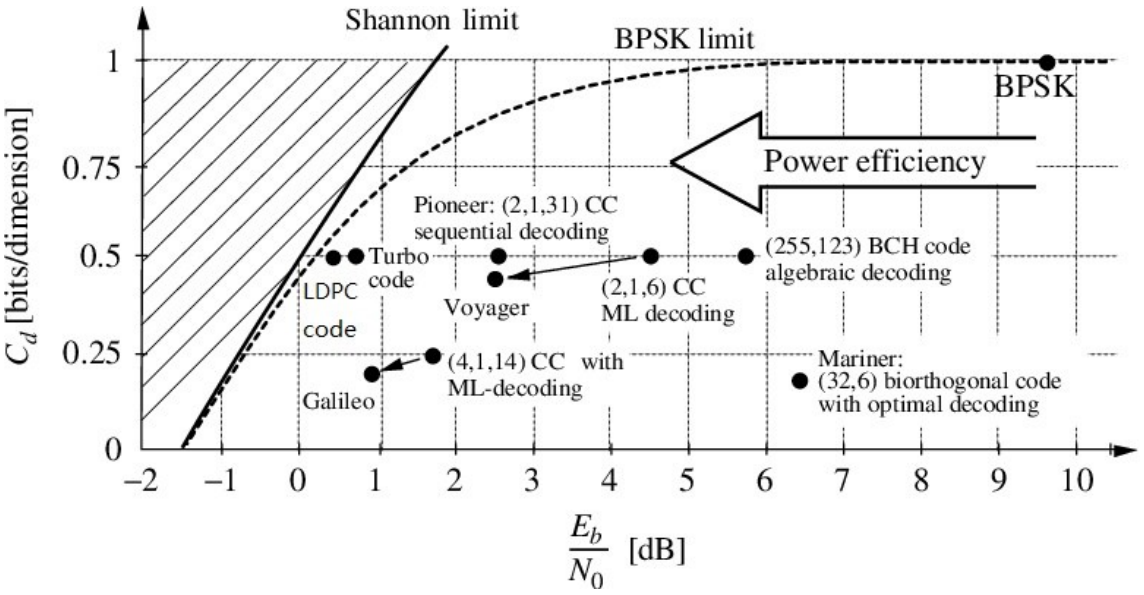


Figure 2-2 The evolution of channel coding [23]

2.2.1 Convolutional Codes

In 1955, Peter Elias invented CC codes. In 1967, Andrew Viterbi proposed a maximum likelihood (ML) decoding algorithm, known as the Viterbi Algorithm, which finds the optimal path through the Trellis structure [1]. Another decoding method, the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm, was proposed in 1976, which solves one of the Viterbi algorithm's limitations in that it no longer assumes that the probabilities of all the information bits are the same. The BCJR algorithm can be considered as the basis of the Maximum a Posteriori Probability (MAP) algorithm. Thus in the 2G era, some standards used the convolutional codes as their channel coding. In the following decades, surprisingly, it was found that CC codes still have a good performance and acceptable

decoding complexity at the same code rate when the message length is short, so it was also widely used in control channels in the 3G and 4G era [25].

2.2.2 Turbo codes

In 1993, C. Berrou, A. Glavieux and P. Thitimajshima proposed the Turbo codes [1]. Turbo codes have significantly impacted channel coding: as shown in Figure 2-2, they are only 1dB away from the Shannon limit, making Turbo codes the preferred channel coding for 3G and 4G standards. Meanwhile, the two fundamental ideas of the Turbo code, namely random encoding and iterative decoding, have led to further research on LDPC codes [1]. However, due to the better performance of LDPC codes and the discovery of Polar codes, Turbo codes were discouraged in 5G NR because of their comparatively low performance [16].

2.2.3 LDPC codes

In 1962, *Gallager* proposed LDPC codes [1]. Although LDPC codes have been studied for a long time, they did not receive much attention until the 1990s. With the discovery of Turbo codes, researchers focused on LDPC codes due to the development of the belief propagation algorithm (sum-product algorithm) and iterative decoding techniques. In 1996, MacKay and Neal constructed new LDPC codes, similar to Turbo codes, about 1dB away from the Shannon limit [24]. In 2001, S.Y. Chung constructed another LDPC codes, only 0.0045 dB away from the Shannon limit [24]. This has led to widespread use of LDPC codes in the 4G era and they have eventually replaced Turbo codes as the data channel coding method in 5G NR.

2.2.4 Polar codes

The Polar Codes are a new type of linear block code initially designed to improve the cutoff rate R_0. In random coding and ML decoding, this parameter determines the maximum error probability of the corresponding code block: while in sequential decoding, R_0 can be considered the 'computational cutoff rate', when the actual rate is higher, the code decoding algorithm will not give the expected performance [5]. In 2009, Erdal Arikan used channel combining and splitting to improve the cutoff rate [6]. He found that as the code length (or the number of channels) increases, the capacity of these subchannels[4] will converge to two extremes: one is a channel with an approximate capacity of 1, and the other is with the approximate capacity of 0. This phenomenon is

[4] Subchannels: Channels after channel splitting operations, see Section 3.2.1 for details.

known as channel polarization. Therefore, this coding method is called Polar code, where 'Polar' is an abbreviation for polarization. It is agreed that Polar codes are the first channel coding method that can provably achieve the Shannon limit in a binary-input discrete memoryless channel (B-DMC). Polar codes developed rapidly and have been considered for use in the 5G NR control channel in 2016 [26].

3. Study of the channel coding of 5G standards

As mentioned in Section 2.1.5, there are three main application areas of 5G NR: eMBB, MMTC, and URLLC [17]. Convolutional codes and Turbo codes can no longer fully satisfy their requirements. So, 3GPP has specified to use of two different types of channel coding in the next generation of mobile communication systems, where LDPC codes will be used in the data channels, and Polar codes will be used in the control channels.

3.1 LDPC codes

3.1.1 Basic Theory

3.1.1.1 Definition of LDPC codes

LDPC codes belong to the class of linear block codes. Like all linear block codes, a matrix form can describe it. As shown in Figure 3-1: a message row vector *u* of *k* bits, adding a redundancy part of *m* bits, can generate the codeword *c* of *n* bits. In this case, the code rate is k/n.

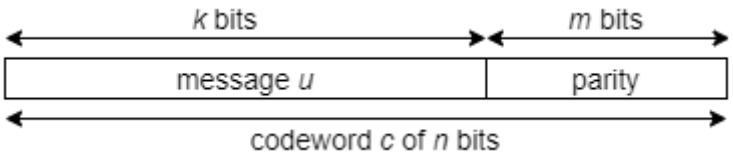


Figure 3-1 Structure of codeword

Since they are linear codes, the codeword *c* can be produced by a generator matrix ***G***:

$$c = \boldsymbol{G}^T u\,, \tag{3-1}$$

where the matrix ***G*** contains two parts ***I*** and ***P***:

$$\boldsymbol{G} = [\boldsymbol{P} \quad \boldsymbol{I}_K]\,. \tag{3-2}$$

The parity check matrix ***H*** can be presented as:

$$\boldsymbol{H} = [\boldsymbol{I}_{n-k} \quad \boldsymbol{P}]\,. \tag{3-3}$$

The characteristic of the LDPC codes is that matrix ***H*** has a low density (the matrix contains many 'zeros'). Thus, based on [1], if the matrix ***H*** of any code satisfies the following conditions:

1) Each column consists of s 1's.
2) Each row consists of v 1's.

3) In any two rows or columns of the matrix, the number of 1's in the same position cannot be greater than 1.
4) Compared to the word length, the parameters *s* y *v* should be as small as possible.

Then these codes can be referred to as regular LDPC codes.

The matrix ***H*** should be sparse. Since LDPC codes use the sum-product algorithm for decoding, a large number of calculations are generated during the decoding process, and a sparse matrix can effectively reduce the number of these calculations and reduce the decoding time, on the other hand, sparsity means that fewer nodes need to be processed and the algorithm can make full use of its operations to reduce the complexity of the calculations.

3.1.1.2 Regular e irregular

Depending on whether condition 3) is satisfied, the LDPC codes can be classified into regular and irregular codes. If this is satisfied, then LDPC codes can be called regular LDPC codes. Otherwise, they are called irregular LDPC codes.

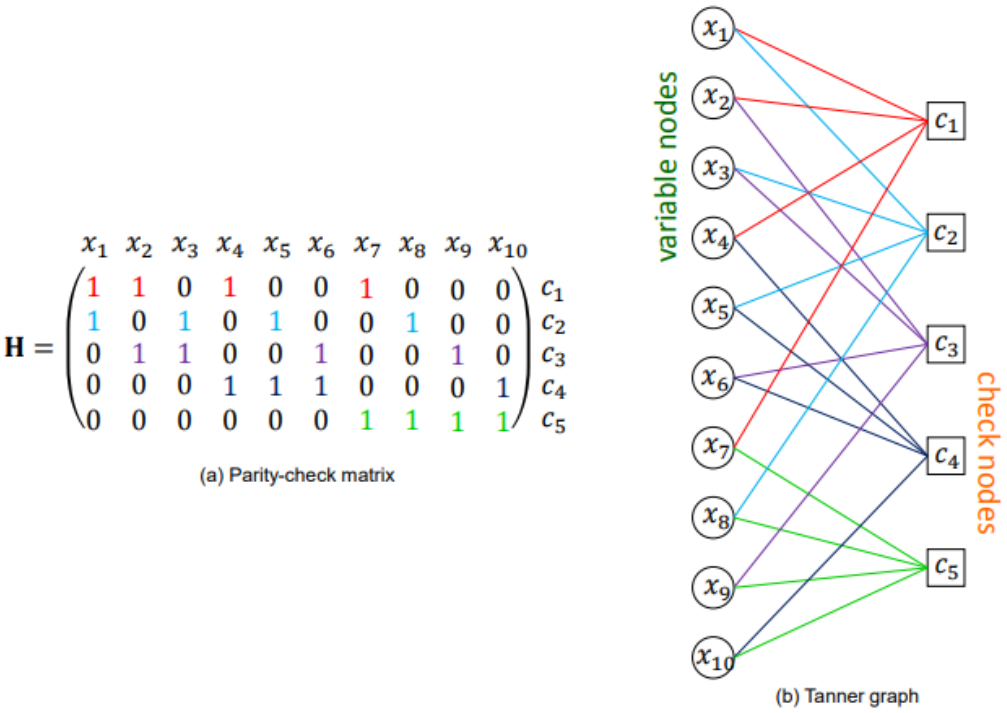


Figure 3-2 Regular LDPC codes[5]

[5] Nguyen LY Thien Truong, "Efficient Hardware Implementations of LDPC Decoders, through Exploiting Impreciseness in Message-Passing Decoding Algorithms," recuperado de: https://cutt.ly/8Qxgddc

As shown in Figure 3-2 (b), each variable node represents a column of the matrix ***H*** and each check node represents a row of the matrix ***H***. The line between a variable node (x_i) and a check node (c_j) is named as an edge, which represents the corresponding element h_{ij} of the matrix ***H*** as an non-zero element. The nodes of the same type are not connected to each other, which means they do not directly transmit any information. The information is only transmitted through the edge between the variable node and the check node [24].

Moreover, based on [2,9,27], the edge degree distribution of variable nodes and check nodes are usually used to represent the regular irregular LDPC codes. The number of non-zero elements in each column of the matrix ***H*** is denoted as d_v and the number of non-zero elements in each row is denoted as d_c, then the parameter λ_i and ρ_i represent the fraction of edges connected to the variable node and check node of degree i, respectively. Communly, the regular LDPC codes have the same edge degree distribution for each node of the same type, while the irregular LDPC codes may have different edge degrade distributions, two edge degree distribution polynomials are shown as follows:

$$\lambda(x) = \sum_{i=1}^{d_v} \lambda_i x^{i-1}, \qquad \rho(x) = \sum_{i=1}^{d_c} \rho_i x^{i-1} . \tag{3-4}$$

The equation (3-4) show the degree distribution of variable nodes and check nodes, respectively. Compared with regular LDPC codes, the irregular LDPC codes are more flexible and may yield better performance [2].

3.1.1.3 Decoding of LDPC codes

There are many different decoding algorithms for LDPC codes, and some of them are shown in the following Table 3-1:

Types	Algorithms	Features
Hard-Decision	Majority-Logic decoding (MLG)	Low complexity and latency with fairly good performance.
	Bit-Flipping decoding (BF)	
Soft-Decision	A posterior probability (APP)	High complexity and latency, but the performance is much better than those algorithms of *Hard-Decision*.
	Weighted BF (WBF)	
	Iterative Decoding based on Belief Propagation (IDBP)	

Table 3-1 Decoding algorithms [1,28]

The decoding algorithms of LDPC codes can be classified into two categories: Hard-Decision and Soft-Decision. The difference between them is that the message transmitted in those decoding algorithms of Hard-Decision contains the actual value of each bit. In contrast, the Soft-Decision is based on a probabilistic decoding algorithm, i.e., the transmitted message is a probability value of a particular bit [29].

The IDBP algorithm has the best performance among these algorithms [1], so it has been widely used in various fields. Therefore, we will focus on this algorithm in this section.

The IDBP algorithm is commonly known as the sum-product algorithm (SPA). Because there are a large number of multiplications calculations in the decoding process, the Log-Likelihood Ratio (LLR) is used in practice to reduce the computational complexity by converting many multiplication operations into summation operations, and this improved method is called the LLR-SPA algorithm.

Before explaining the LDPC decoding and the LLR-SPA algorithm in detail, as can be seen in Figure 3-3, based on [2,30], assume that an LDPC code is characterized by some variable nodes ($\mathbf{v_{0,1,2\ldots}}$) and some check nodes ($\mathbf{c_{0,1,2\ldots}}$). For the LDPC codes transmitted in a binary input additive white Gaussian noise channel (BI-AWGN) with BPSK modulation, the mean value is null and the variance equal to $\boldsymbol{\sigma^2}$.

According to [2], the following definitions apply:

- $\mathrm{M(j)}$ is a set of check nodes connected with the variable node v_j. $\mathrm{M(j) \setminus i}$ is a set of check nodes without the check node $\mathrm{c_i}$.
- $\mathrm{N(i)}$ is a set of variable nodes connected with the check node $\mathrm{c_i}$. $\mathrm{N(i) \setminus j}$ is a set of variable nodes without the variable node v_j.
- $\mathrm{q_{ij}(x)}, x \in \{0,1\}$ is the information transmitted from the variable node v_i to the check node c_j, which indicates the probability of the bit associated with v_i being equal to x. This information is calculated by combining the information of all those check nodes connected with v_i, except c_j.
- $r_{ji}(x), x \in \{0,1\}$ is the information transmitted from the check node c_j to the variable node v_i, which indicates the probability of the bit associated with v_i being equal to x. This information is calculated by combining the information of all those variable nodes connected with c_j, except v_i.

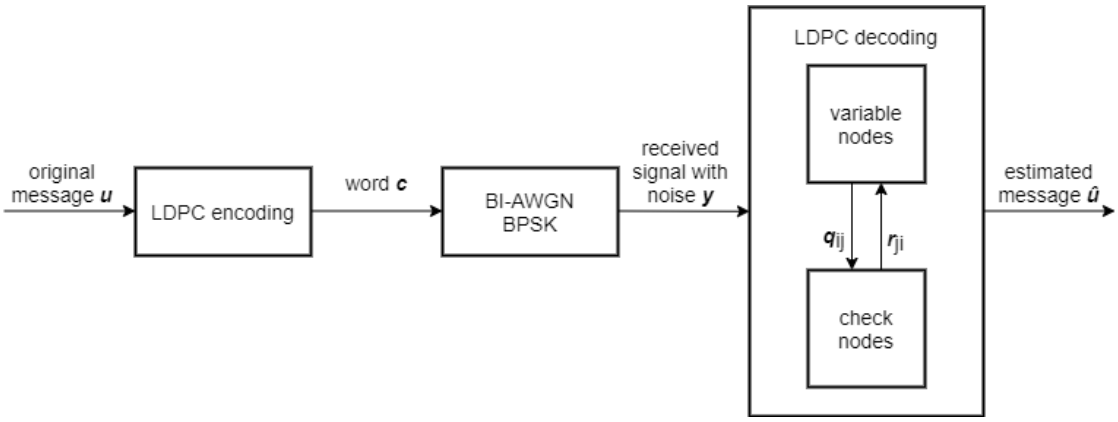


Figure 3-3 Model of encoding and decoding process of LDPC codes

Then, the information of the received value y is defined in LLR form:

$$LLR(y_i) = \ln\left(\frac{P(y_i|x_j = 0)}{P(y_i|x_j = 1)}\right). \tag{3-5}$$

The message sent from the variable node i to the check node j can be described as:

$$LLR(q_{ij}) = \ln\left(\frac{q_{ij}(0)}{q_{ij}(1)}\right). \tag{3-6}$$

The message sent from the check node j to the variable node i can be described as:

$$LLR(r_{ji}) = \ln\left(\frac{r_{ji}(0)}{r_{ji}(1)}\right). \tag{3-7}$$

Based on [2,24,30], the decoding steps are as follows:

1) Initialization

Assign the initial values to the nodes on both sides.

$$LLR(q_{ij}) = LLR(y_i) = \frac{2y_i}{\sigma^2}. \tag{3-8}$$

2) Update the check nodes

The message is transmitted from the check node to the variable node.

$$LLR(r_{ji}) = 2\tanh^{-1}\left[\prod_{j\in N(i)\backslash \mathrm{y}} \tanh\left(\frac{1}{2}LLR(q_{ij})\right)\right], \tag{3-9}$$

let,

$$\begin{cases} \alpha_{ij} = sign\left(LLR(q_{ij})\right) \\ \beta_{ij} = \left|LLR(q_{ij})\right| \end{cases}. \tag{3-10}$$

then, it can be written as:

$$LLR(r_{ji}) = \left(\prod_{j \in N(i) \setminus j} \alpha_{ij} \right) f\left(\sum_{j \in N(i) \setminus j} f(\beta_{ij}) \right), \tag{3-11}$$

Where,

$$f(x) = f^{-1}(x) = -\ln\left(\frac{x}{2}\right) = \ln\left(\frac{e^x + 1}{e^x - 1}\right). \tag{3-12}$$

3) Update the variable nodes

The message is transmitted from the variable node to the check node.

$$LLR(q_{ij}) = LLR(y_i) + \sum_{i \in M(j) \setminus i} LLR(r_{ji}). \tag{3-13}$$

4) Update the states of $LLR(y_i)$ and make an estimate.

$$LLR(y_i)_{total} = LLR(y_i) + \prod_{i \in M(j)} r_{ji}. \tag{3-14}$$

The estimated value is:

$$\hat{x}_i = \begin{cases} 1, & LLR(y_i)_{total} < 0 \\ 0, & LLR(y_i)_{total} \geq 0 \end{cases}. \tag{3-15}$$

The above steps are repeated until the whole codeword is decoded, i.e., $\hat{x}_i \boldsymbol{H}^T = 0$. It can be seen from those steps above that there are still many multiplication calculations in LLR-SPA. In order to ease the calculation and increase the decoding speed, there is a similar algorithm called the Min-Sum algorithm [24]. In this algorithm, only the approximate value of (3-11) is requested, i.e.:

$$f\left(\sum_{j \in N(i) \setminus y} f(\beta_{ij}) \right) \approx \min_{j \in N(i) \setminus y} (\beta_{ij}), \tag{3-16}$$

then (3-11) is replaced by:

$$LLR(r_{ji}) = \left(\prod_{j \in N(i) \setminus y} \alpha_{ij} \right) \min_{j \in N(i) \setminus y} (\beta_{ij}). \tag{3-17}$$

Since the Min-Sum algorithm calculates the approximate value in step 2), it can reduce the computational complexity, but its performance will be reduced.

3.1.2 LDPC codes in 5G NR

LDPC codes can be classified into two categories based on how they are built: based on random structures or based on regular structures [31], such as irregular LDPC codes and regular LDPC codes, respectively, as mentioned in Section 3.1.1.2. Although random structures perform better than regular structures because of their higher flexibility when the code length is long, they are also very complex to encode and decode because the

variable nodes and check nodes are irregularly connected, making them challenging to implement in hardware. However, regular structures can have lower complexity while maintaining a relatively high performance [31]. Quasi-Cyclic LDPC (QC-LDPC) codes are codes based on regular structures. They have been widely used in different standards, e.g., IEEE 802.11n, IEEE 802.16e, etc. Therefore, 3GPP has chosen to continue to use QC-LDPC codes as channel coding for 5G NR. In the following, we will focus on the characteristics of QC-LDPC codes under the 5G NR standard.

3.1.2.1 Protograph construction

In general, the matrix $\boldsymbol{H}$ of the QC-LDPC codes can be defined by an exponent matrix $\boldsymbol{H_B}$ and the shift coefficients $\mathrm{P_{i,j}}$ [3-4].

The form of the exponent matrix $\boldsymbol{H_B}$ is shown in (3-18):

$$\boldsymbol{H_B} = \begin{bmatrix} P_{1,1} & \cdots & P_{1,n} \\ \vdots & \ddots & \vdots \\ P_{m,1} & \cdots & P_{m,n} \end{bmatrix}. \tag{3-18}$$

The dimension of the matrix $\boldsymbol{H_B}$ is $(\mathrm{m} \times \mathrm{n})$, where $\mathrm{m} < \mathrm{n}$. For any integer value $\mathrm{P_{i,j}}$, has $0 < \mathrm{P_{i,j}} < \mathrm{z}$, where $i \in \{1,2,3,\cdots,m\},\ j \in \{1,2,3,\cdots,n\}$.

The matrix obtained by shifting the identity matrix $\boldsymbol{I}$ by $\mathrm{P_{i,j}}$ units to the right is called the cyclic permutation matrix $\boldsymbol{Q}(\mathrm{P_{i,j}})$ whose size is $(\mathrm{z} \times \mathrm{z})$ [3]. When $\mathrm{P_{i,j}}$ is a negative integer, the matrix $\boldsymbol{Q}(\mathrm{P_{i,j}})$ will be a zero matrix. For example, when $z = 3$:

$$\boldsymbol{Q}(0) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{Q}(2) = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \boldsymbol{Q}(-1) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{3-19}$$

Finally, as can be seen in (3-20), the parity check matrix $\boldsymbol{H}$ can be obtained by replacing the elements in the matrix $\boldsymbol{H_B}$ with the matrix $\boldsymbol{Q}$. This method is called protograph construction [3-4]. It can reduce the required memory in hardware and allow encoding and decoding using a simple switching network [9]. In addition, the protograph codes enable nature parallelism [9], support parallel operations in both encoding and decoding, greatly reducing the latency of operations.

An example of protograph construction:

$$\boldsymbol{H_B} = \begin{bmatrix} 1 & -1 & 1 \\ 0 & 2 & 0 \end{bmatrix}$$
$$\Downarrow \quad z = 3$$
$$\boldsymbol{H} = \left[\frac{Q(1)|Q(-1)|Q(1)}{Q(0)|Q(2)|Q(0)}\right] = \left[\begin{array}{ccc|ccc|ccc} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]. \tag{3-20}$$

In 5G NR, the parameter z is referred to as lifting size. Considering the parallelism and complexity of the switch networks [9,26], 3GPP has specified the lifting size $z = a \times 2^j$, where $a \in \{2,3,5,7,9,11,13,15\}$ and $j \in \{0,1,2,3,4,5,6,7\}$. As shown in Table 3-2, 3GPP has specified eight sets ($i_{LS} = 0,1,\cdots,7$) of 51 different values, where the minimum value of z is 2, and the maximum value is 384. In addition, the base graph of QC-LDPC codes should support all these values.

Set index (i_{LS})	a	Lifting size $z = a \times 2^j$
0	2	{2, 4, 8, 16, 32, 64, 128, 256}, $j \in \{0,1,2,3,4,5,6,7\}$
1	3	{3, 6, 12, 24, 48, 96, 192, 384}, $j \in \{0,1,2,3,4,5,6,7\}$
2	5	{5, 10, 20, 40, 80, 160, 320}, $j \in \{0,1,2,3,4,5,6\}$
3	7	{7, 14, 28, 56, 112, 224}, $j \in \{0,1,2,3,4,5\}$
4	9	{9, 18, 36, 72, 144, 288}, $j \in \{0,1,2,3,4,5\}$
5	11	{11, 22, 44, 88, 176, 352}, $j \in \{0,1,2,3,4,5\}$
6	13	{13, 26, 52, 104, 208}, $j \in \{0,1,2,3,4\}$
7	15	{15, 30, 60, 120, 240}, $j \in \{0,1,2,3,4\}$

Table 3-2 Lifting size set [8]

3.1.2.2 Base graphs

In 5G NR, the exponent matrix is often referred to as Base Graph (BG). The structure of the base graph is shown in Figure 3-4. The dimension of the BG is $(m_b \times n_b)$. The columns are divided into two parts: the information part (total k_b columns) and the parity part (total m_b columns), also the rows are divided into two parts: core checks part and extension checks part [3,32]. According to [3-4], the base graph comprises five different submatrices: ***A, B, O, C,*** and ***I***. Submatrix ***A*** contains systematic bits. Submatrix ***B*** corresponds to the parity bits. It is a square matrix with a dual-diagonal structure whose first column weights 3, and other columns have a dual-diagonal structure. Submatrix ***O*** is a null matrix. Submatrix ***C*** corresponds to the single parity check (SPC) rows. Submatrix ***I*** is an identity matrix, which is an extension part of SPC. The extension checks rows can be orthogonal or quasi-orthogonal, as shown in Figure 3-6 and Figure 3-8. This design can reduce the decoding latency and improve decoding reliability [9,32]. The submatrices ***A*** and ***B*** combine as the kernel matrix, which has a structure of the

Irregular Repeat Accumulate (IRA) code type, in order to be encoded quickly and efficiently [9]. The other submatrices ***O***, ***C,*** and ***I*** are called extensions and are mainly used to support the incremental redundancy hybrid automatic repeat request (IR-HARQ) to improve the reliability [9] and the transmission efficiency [32] of the communication. This structure type is similar to a Raptor-like extension [3], with a high-rates kernel matrix and other extensions are used to support low-rates.

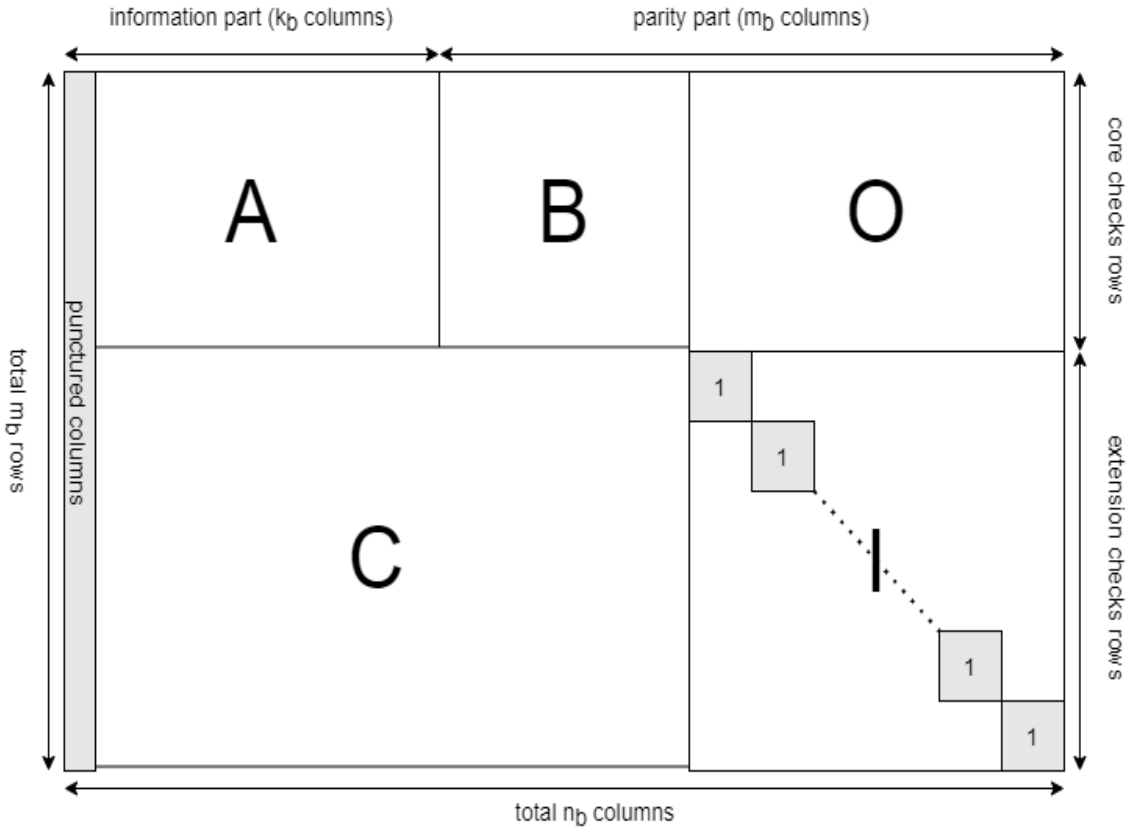


Figure 3-4 The structure of the Base Graph (BG)

When designing the base graph, it is necessary to consider some factors: performance, coding and decoding complexity, hardware cost, etc. Therefore, considering all these factors, 3GPP has specified two types of base graph structures: Base Graph 1 (BG1) and Base Graph 2 (BG2), and how they are chosen depends on the code rate (R) and transport block size (or payload length A). In general, the BG1 is designed for a high code rate and a long information block, while the BG2 is used for a low code rate and a small information block. As shown in Figure 3-5, the BG2 is selected when $A < 292$, or $292 < \mathrm{A} < 3824 \wedge \mathrm{R} < 0.67$, or $R < 0.25$. Otherwise, BG1 is selected.

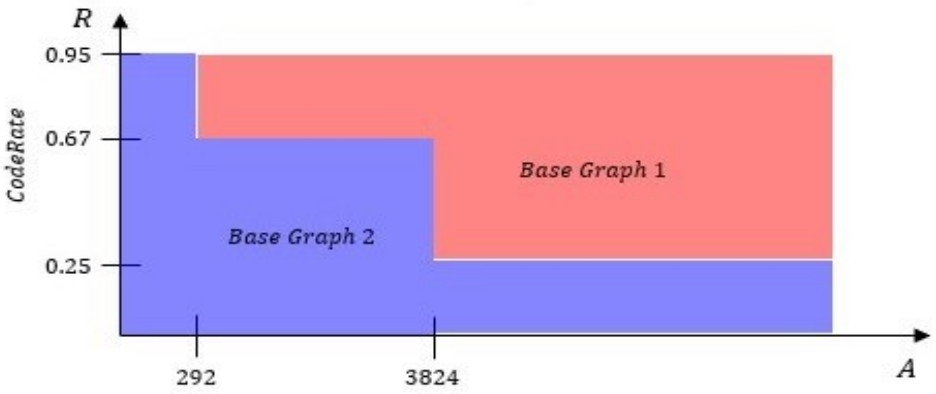


Figure 3-5 Base Graph selection[6]

Moreover, as shown in Figure 3-4, it can be seen that the first two columns of both BGs are always punctured to improve their performance. Consequently, these first two columns are not transmitted at any time [32]. However, the decoder must recover them because they contain information bits [9].

3GPP specifies two different BGs so that LDPC codes can ensure good performance and low decoding latency at different code rates and code block sizes.

	Base Graph	
	BG1	BG2
Dimension ($m_b \times n_b$)	46x68	42x52
Number of the information column	$k_b = 22$	$k_b = 10$
Maximum code block size K_{cb} (bits)	8448	3840
Minimum code rate	1/3	1/5

Table 3-3 Principal features of two BGs [8,16]

3.1.2.2.1 Base graph 1 (BG1)

As shown in Table 3-3, $\boldsymbol{H_{BG1}}$ has a total of 46 rows and 68 columns. As shown in Figure 3-6, the submatrix ***A*** of BG1 consists of the first 4 rows and the first 22 columns ($k_{b,máx} = 22$). The green dots in Figure 3-7 represent the non-zero elements of the BG1. As mentioned in Section 3.1.2.2, submatrix ***B*** has a dual-diagonal structure, and its first column weights 3. The maximum block code size supported by BG1 is:

$$K_{cb} = k_{b,máx} \times z_{máx} = 22 \times 384 = 8448\ bits\ , \qquad (3\text{-}21)$$

where the maximum value of lifting size $z_{máx}$ can be obtained from Table 3-2. The minimum code rate is 1/3.

[6] retrieved from: https://www.sharetechnote.com/html/5G/5G_PDSCH.html

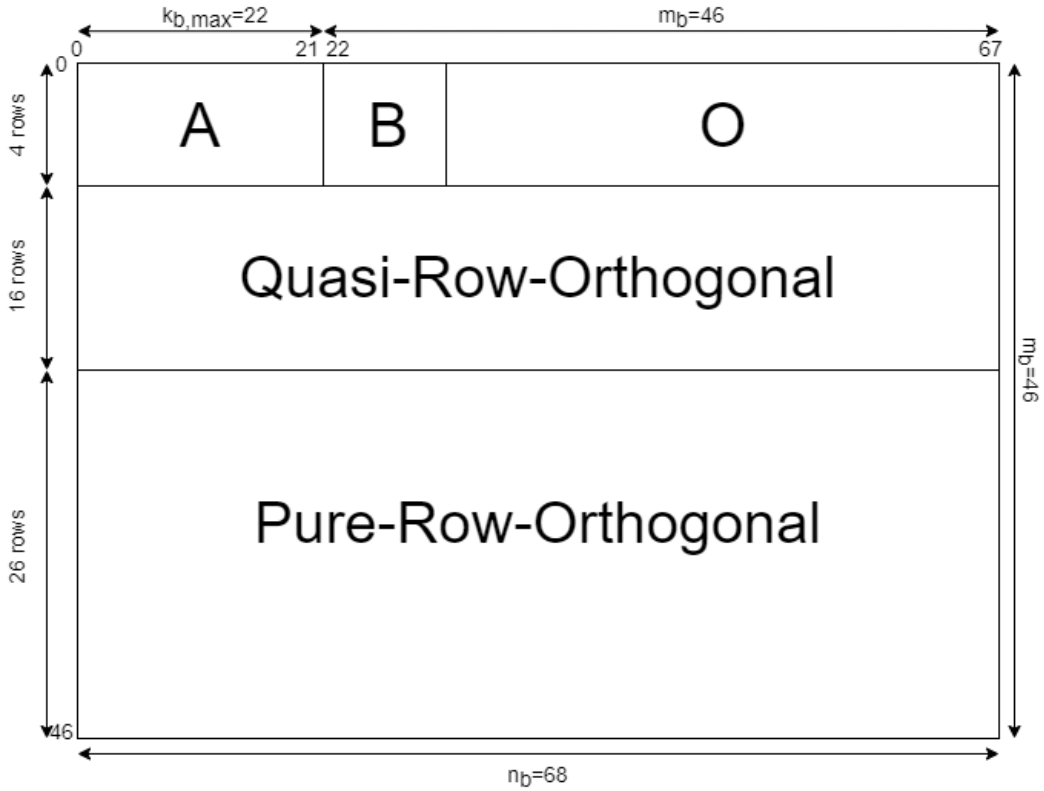


Figure 3-6 Structure of BG1[7]

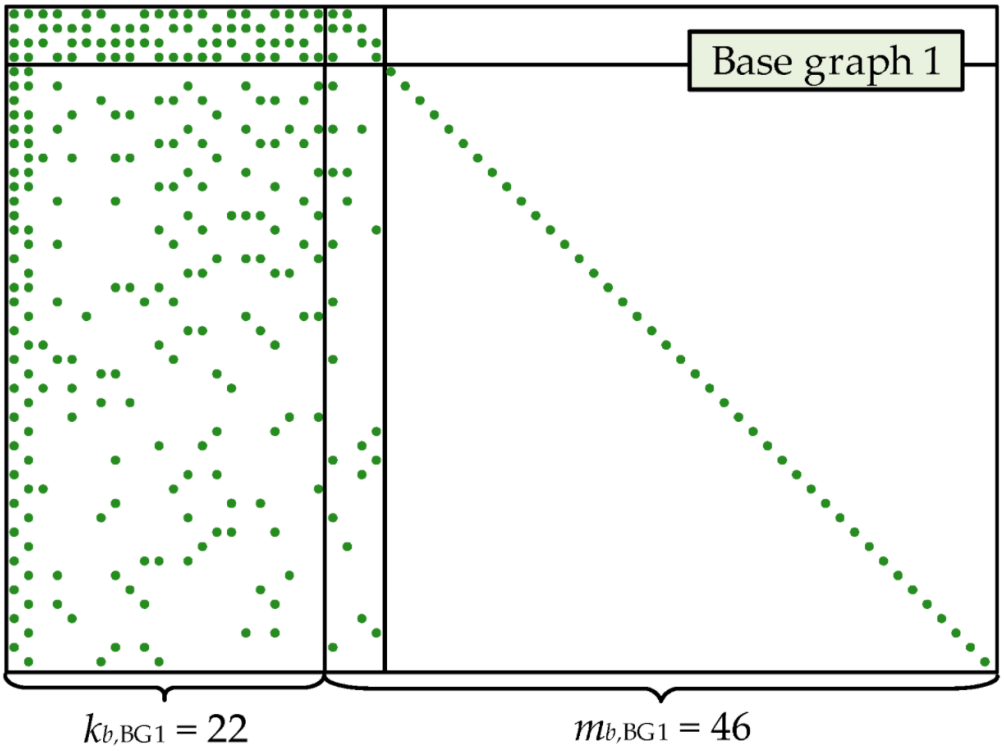


Figure 3-7 Non-zero elements of BG1 [33]

[7] retrieved from: https://developer.aliyun.com/article/739879, figure 2-43

3.1.2.2.2 Base graph 2 (BG2)

As shown in Table 3-3, $\boldsymbol{H_{BG2}}$ has a total of 42 rows and 52 columns. As shown in Figure 3-8, the submatrix ***A*** of BG2 consists of the first four rows and the first ten columns ($k_{b,máx} = 10$). The green dots in Figure 3-9 represent the non-zero elements of the BG2. As mentioned in Section 3.1.2.2, submatrix ***B*** has a dual-diagonal structure, and it should be noted that although the weight of the first column is 3, the zero elements of BG1 are in the third row while BG2's is in the second row. The maximum block code size supported by BG2 is:

$$K_{cb} = k_{b,máx} \times z_{máx} = 10 \times 384 = 3840\ bits\ , \tag{3-22}$$

where the maximum value of lifting size $z_{máx}$ can be obtained from Table 3-2. The minimum code rate is 1/5.

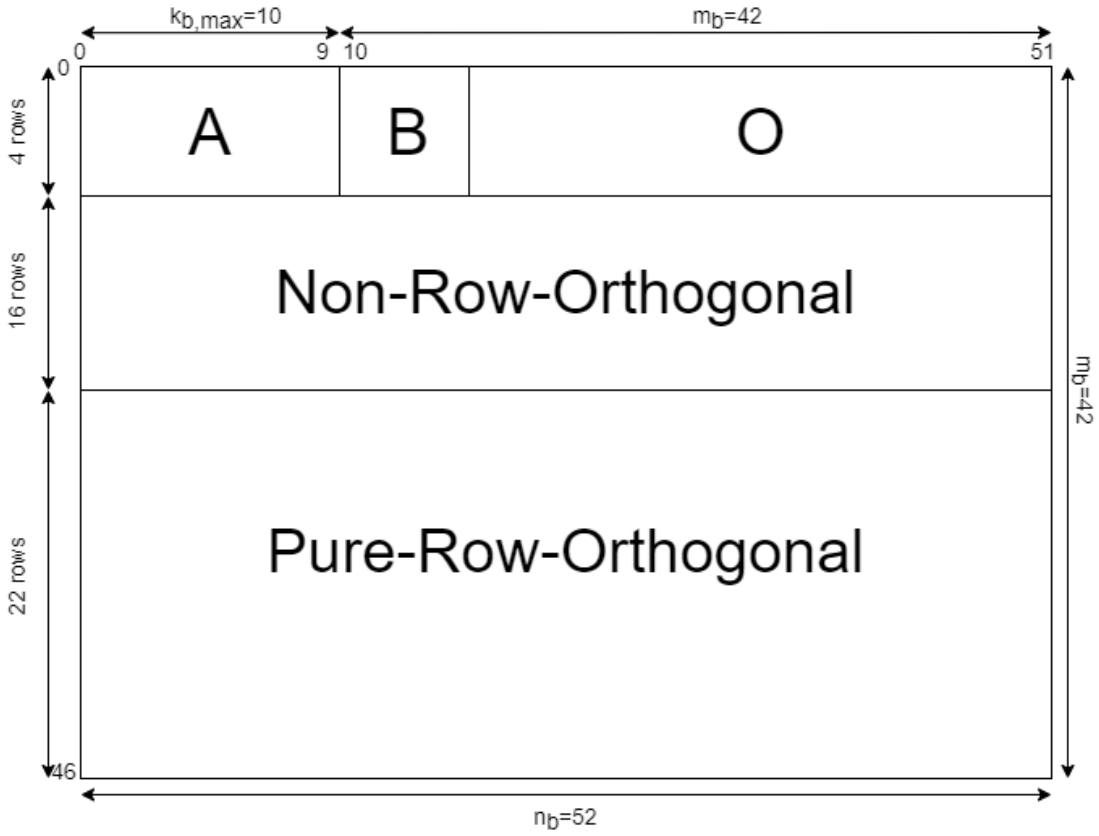


Figure 3-8 Structure of BG2[8]

[8] retrieved from: https://developer.aliyun.com/article/739879, figure 2-45.

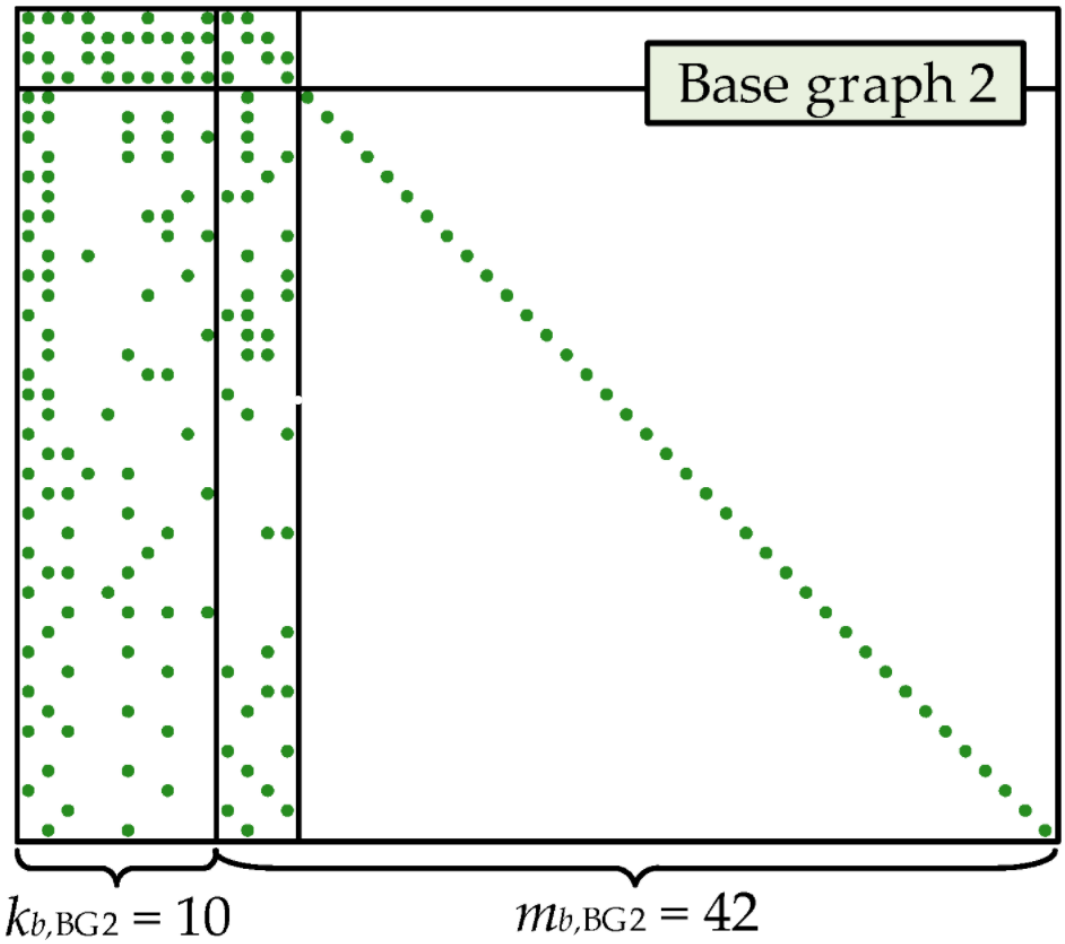


Figure 3-9 Non-zero elements of BG2 [33]

3.1.2.2.3 Example of the construction of the matrices $\boldsymbol{H_{BG}}$ and $\boldsymbol{H}$ in 5G NR

As shown in Figure 3-5, it is already known how to select the type of BGs. As shown in Figure 3-13, we assume that the uncoded code B is 15 bits and the code rate R is 0.5. Based on [3], the steps to construct the matrix ***H*** are as follows:

1) As shown in Figure 3-5, $B = 15 < 292$ bits, and choose BG2.
2) Determine the value of k_b.

According to [8], it can be summarized as (3-23):

$$\begin{cases} BG1 \rightarrow k_b = 22 \\ BG2 \rightarrow \begin{cases} B > 640 \rightarrow k_b = 10 \\ B > 560 \rightarrow k_b = 9 \\ B > 192 \rightarrow k_b = 8 \\ otherwise \rightarrow k_b = 6 \end{cases} \end{cases}, \tag{3-23}$$

since BG2 has been selected in steps one and *B=15*, then we can obtain $k_b = 6$.

3) Determine the minimum value of z.

$$k_b \times z \geq B\,. \tag{3-24}$$

Based on (3-24) and Table 3-2, we can obtain the following results $z_{min} = 3$ $(i_{LS} = 1,\ a = 3)$.

4) Determine the value of $\mathrm{P_{i,j}}$.

$$\mathrm{P_{i,j}} = \begin{cases} -1, & \mathrm{v_{i,j}} = \mathrm{Null}^{9} \\ \mathrm{mod}(\mathrm{v_{i,j}}, \mathrm{z}), & \text{otherwise} \end{cases}, \tag{3-25}$$

In reference [8], there are relevant tables that can be found to determine the values of $\mathrm{v_{i,j}}$[10]. As can be seen in Figure 3-12, when $i_{LS} = 1$, the value of $\mathrm{v_{i,j}}$ in row 0, column 0 is 174, then based on (3-25), it has:

$$\mathrm{P_{i,j}} = \mathrm{mod}\left(\mathrm{v_{i,j}}, \mathrm{z}\right) = \mathrm{mod}(174, 3) = 0\,, \tag{3-26}$$

moreover, for row 0 column 1, it has:

$$\mathrm{P_{i,j}} = \mathrm{mod}\left(\mathrm{v_{i,j}}, \mathrm{z}\right) = \mathrm{mod}(97, 3) = 1\,. \tag{3-27}$$

Similarly like (3-26) and (3-27), the matrix $\boldsymbol{H_{BG2}}$ can be obtained by calculating each row and column in turn. For example, the values of submatrix ***A*** are shown in Figure 3-10.

	0	1	2	3	4	5	6	7	8	9	
0	174	97	166	66	-1	-1	71	-1	-1	172	$v_{i,j}$
1	27	-1	-1	36	48	92	31	187	185	3	
2	25	114	-1	117	110	-1	-1	-1	114	-1	
3	-1	136	175	-1	113	72	123	118	28	186	

	0	1	2	3	4	5	6	7	8	9	
0	0	1	1	0	-1	-1	2	-1	-1	1	***A***
1	0	-1	-1	0	0	2	1	1	2	0	
2	1	0	-1	0	2	-1	-1	-1	0	-1	
3	-1	1	1	-1	2	0	0	1	1	0	

*Figure 3-10 The values of $v_{i,j}$ and submatrix **A***

5) As mentioned in Section 3.1.2.1, once the values of the matrix $\boldsymbol{H_{BG2}}$ and z are obtained, a parity check matrix ***H*** of size $(m_b \times z) \times (n_b \times z)$ can be obtained by extending the matrix $\boldsymbol{H_{BG2}}$ with the matrix $\boldsymbol{Q}(\mathrm{P_{i,j}})$ using the protograph construction method. As a result, submatrix ***A*** of matrix ***H*** is extended to a matrix of size 12×30, as shown in Figure 3-11.

[9] There is no corresponding value in the tables.

[10] There are two tables in [8], corresponding to BG1 and BG2, respectively. See [8] pp. 21-25 for details.

	0			1			2			3			4			5			6			7			8			9		
0	1	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0
	0	1	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	1	0	0	1	0	0	0	1	1	0	0
	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	1	0	1	0
	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	0	0	0	1	0	0	0	1
2	0	1	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
	0	0	1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
3	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	1	0	0	1	0	0	0	1	0	0	1	0	1	0	0
	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	1	0
	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	1	0	1	0	0	1	0	0	1	0	1

*Figure 3-11 The extended submatrix **A** of matrix **H***

As described in steps 1 to 5, although only the submatrices ***A*** are shown due to the large computational requirements, the base graph is not fully computed and presented. However, it is also described in detail the steps and methods how to select the base graph and construct a parity check matrix ***H***. Repeating the same steps, the submatrices ***B***, ***O***, ***I***, and ***C*** are computed sequentially, and then extended to yield the complete parity check matrix ***H***.

$\mathbf{H}_{\mathrm{BG}}$		$V_{i,j}$							
Row index i	Column index j	Set index i_{LS}							
		0	1	2	3	4	5	6	7
0	0	9	174	0	72	3	156	143	145
	1	117	97	0	110	26	143	19	131
	2	204	166	0	23	53	14	176	71
	3	26	66	0	181	35	3	165	21
	6	189	71	0	95	115	40	196	23
	9	205	172	0	8	127	123	13	112
	10	0	0	0	1	0	0	0	1
	11	0	0	0	0	0	0	0	0
1	0	167	27	137	53	19	17	18	142
	3	166	36	124	156	94	65	27	174
	4	253	48	0	115	104	63	3	183
	5	125	92	0	156	66	1	102	27
	6	226	31	88	115	84	55	185	96
	7	156	187	0	200	98	37	17	23
	8	224	185	0	29	69	171	14	9
	9	252	3	55	31	50	133	180	167
	11	0	0	0	0	0	0	0	0
	12	0	0	0	0	0	0	0	0
2	0	81	25	20	152	95	98	126	74
	1	114	114	94	131	106	168	163	31
	3	44	117	99	46	92	107	47	3
	4	52	110	9	191	110	82	183	53
	8	240	114	108	91	111	142	132	155
	10	1	1	1	0	1	1	1	0
	12	0	0	0	0	0	0	0	0
	13	0	0	0	0	0	0	0	0
3	1	8	136	38	185	120	53	36	239
	2	58	175	15	6	121	174	48	171
	4	158	113	102	36	22	174	18	95
	5	104	72	146	124	4	127	111	110
	6	209	123	12	124	73	17	203	159
	7	54	118	57	110	49	89	3	199
	8	18	28	53	156	128	17	191	43
	9	128	186	46	133	79	105	160	75
	10	0	0	0	1	0	0	0	1
	13	0	0	0	0	0	0	0	0

Figure 3-12 BG2 ($\boldsymbol{H}_{BG2}$) and its parity check matrices $\boldsymbol{V}_{i,j}$[11]

[11] The complete table are shown in [8], pp. 25.

3.1.3 NR LDPC coding chain

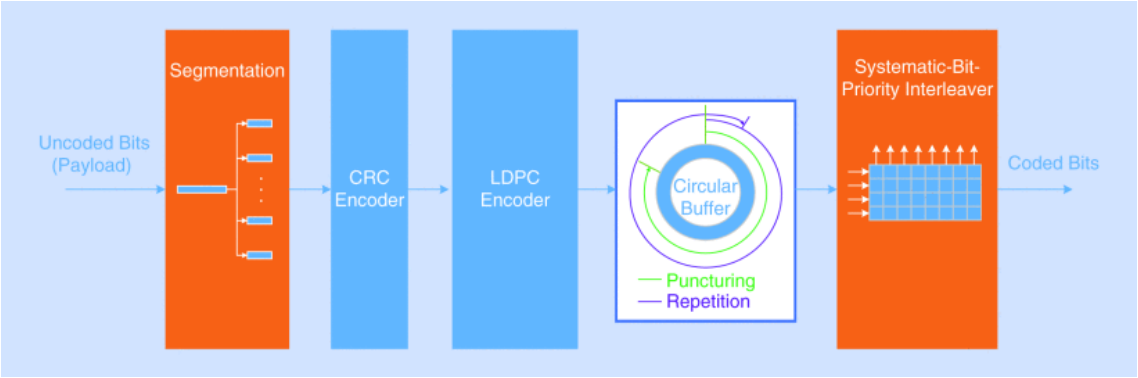


Figure 3-13 The NR LDPC coding chain [16]

3.1.3.1 Segmentation

As shown in Table 3-3, for each BG, there is a maximum code block size (K_{cb}) it supports. As presented in Figure 3-13, according to [8], when the length of the uncoded code B is greater than K_{cb}, this code B will be split into several new segments B'. The number of segments C is defined as :

$$C = \left\lceil \frac{B}{K_{cb} - L} \right\rceil , \tag{3-28}$$

where L is an additional CRC sequence of 24 bits, thus, the length of the new segment is equal to:

$$B' = B + CL . \tag{3-29}$$

3.1.3.2 Rate matching

Since the amount of resources transmitted in mobile communication systems is dynamically changing, 5G NR LDPC codes need to support a rate matching feature to select any number of bits for transmission [9]. Moreover, this feature is applied to each code block. The rate matching operation is controlled by several redundancy versions (RVs) in the circular buffer. Each RV corresponds to a column position in a BG. The length of each RV is different, depending on the number of transmission resources available [9]. The starting position of RV is a multiple of z, where z is the lifting size. The initial transmission always starts with RV0. As can be seen in Table 3-4, the position of its corresponding column is 0. However, as mentioned in Section 3.1.2.2, the first two columns of the base graph are always punctured and not transferred. Therefore, the actual corresponding column is the third column of the base graph [9,34]. 3GPP has not specified the transmission order, but for both BGs, better performance can usually be obtained by RV0 → RV2 → RV3 → RV1 [34].

RVs	Corresponding column index	
	BG1	BG2
RV0	0	0
RV1	17z	13z
RV2	33z	25z
RV3	56z	43z

Table 3-4 The version of RVs and the corresponding initial position [34]

3.1.3.3 Interleaving

After rate matching, bit interleaving is required. This operation can increase the reliability of the systematic bits and improve the performance of the QC-LDPC codes [32]. According to [8], input bits are defined as $e_0, \cdots, e_{E-1}$and output bits are defined as $f_0, \cdots, f_{E-1}$, where E is the rate matching output sequence length. As shown in Figure 3-14, in the interleaver, the bits are written row-by-row, read column-by-column, and their number of rows is the same as the modulation order Q_m [8,32]. For example, for 16-QAM, 64-QAM, 256-QAM, the order is 4, 6, and 8, respectively. According to the definition of [8][12], the relationship between e and f is:

$$f_{i+jQ_m} = e_{iE/Q_m+j}\ , \tag{3-30}$$

where $j \in \{0, \cdots, E/Q_m - 1\}$, y $i \in \{0, \cdots, Q_m - 1\}$.

As shown in Figure 3-13 and Figure 3-15, the operation of interleaving comes after rate matching, thus the rate matching output E is also the same a the interleaving input. when the input length $E = 16$, and the modulation order $Q_m = 2$, in the interleaver, the input bits e are written in two rows of eight columns, starting from the first column, read out sequentially by columns, and then recombined into a row of 16 columns as the output bits f.

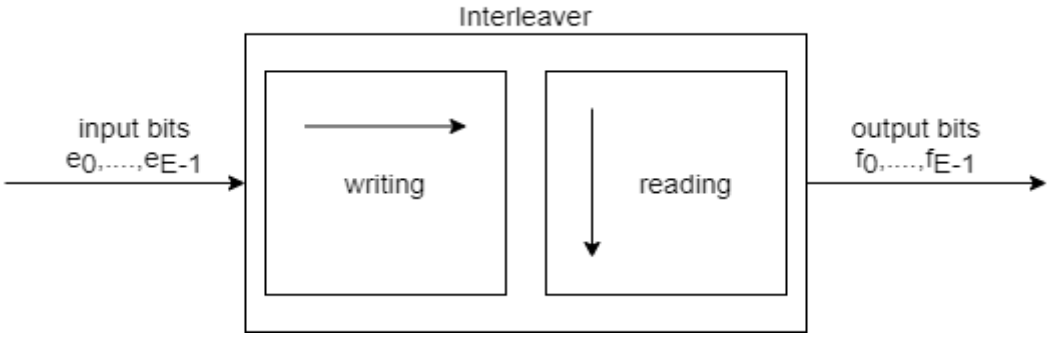


Figure 3-14 the interleaver model

[12] In clause 5.4.2.2 of [8], pp. 33.

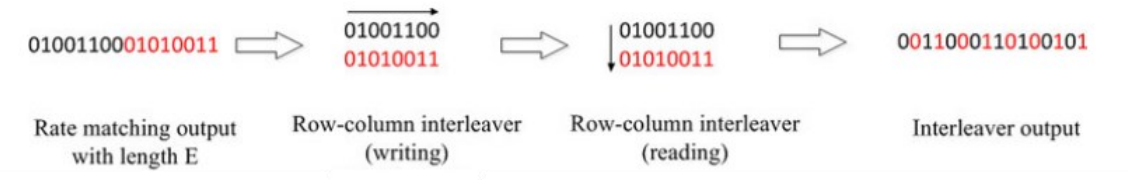


Figure 3-15 An interleaving example [9]

3.1.4 Conclusion

Section 3.1 focuses on some basic definitions, theories, and the most widely used decoding algorithm of LDPC codes. In addition, it introduces the structure and design requirements of its regular construction form, QC-LDPC codes, in 5G NR. Finally, the processes that need to be carried out in the 5G NR coding chain are presented, as well as the specification of 3GPP for each process.

3.2 Polar Codes

3.2.1 Basic theory

Before detailing the Polar codes, the following definition of channel capacity is made according to [6,7]. For a B-DMC, as shown in Figure 3-16, define its input as $X = \{0,1\}$, and the transition probability as $W(y|x)$, where $x \in X, y \in Y$.

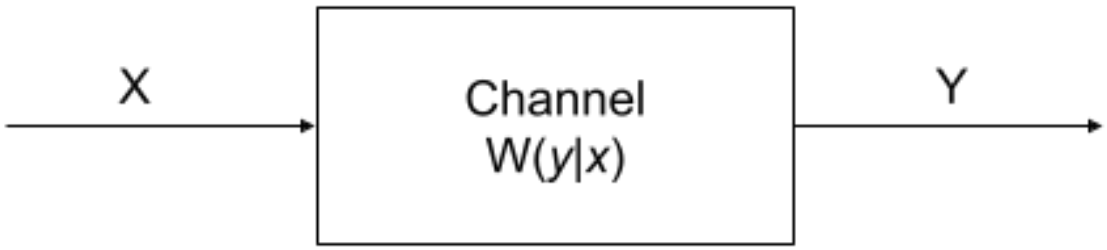


Figure 3-16 B-DMC model

Therefore, when the B-DMC is symmetric with equal probability of the input x, the channel capacity is the same as the mutual information $I(X;Y)$:

$$C(W) \triangleq I(X;Y) = \sum_{y\in Y}\sum_{x\in X}\frac{1}{2}W(y|x)\log\frac{W(y|x)}{\frac{1}{2}W(y|0)+\frac{1}{2}W(y|1)}. \qquad (3\text{-}31)$$

In general, the Bhattacharyya parameter is used to measure transmission reliability [6]. It is defined as:

$$Z(W) \triangleq \sum_{y \in Y} \sqrt{W(y|0)W(y|1)} \ . \tag{3-32}$$

According to [6-35], $Z(W)$ refers to the upper bound on the error probability of the ML decision when the channel W transmits only a single bit. Thus, this parameter is also considered as the maximum probability of transmission error.

It can likewise be found that $Z(W) \in [0,1]$. By using the base-2 logarithm, similar as $Z(W)$, $C(W)$ will also belong to $[0,1]$. At this point, $C(W)$ is inversely related to $Z(W)$, i.e., $C(W) \approx 1$ iff $Z(W) \approx 0$, and vice versa [6]. In addition, for any B-DMC W, $C(W)$ is bounded as:

$$\log \frac{2}{1+Z(W)} \leq C(W) \leq \sqrt{1-Z(W)^2} \, . \tag{3-33}$$

3.2.1.1 Channel combining

A basic model of the channel combination is shown in Figure 3-17, where two ($N = 2$) independent individual channels $W: X \to Y$ are combined into a new channel $W_2: X^2 \to Y^2$ by a one-to-one bit mapping [5]. The mapping f_2 is defined as:

$$f_2(u_1, u_2) \triangleq (u_1 \oplus u_2, u_2) \, , \tag{3-34}$$

where $\oplus$ is the modulo-2 addition in the binary field.

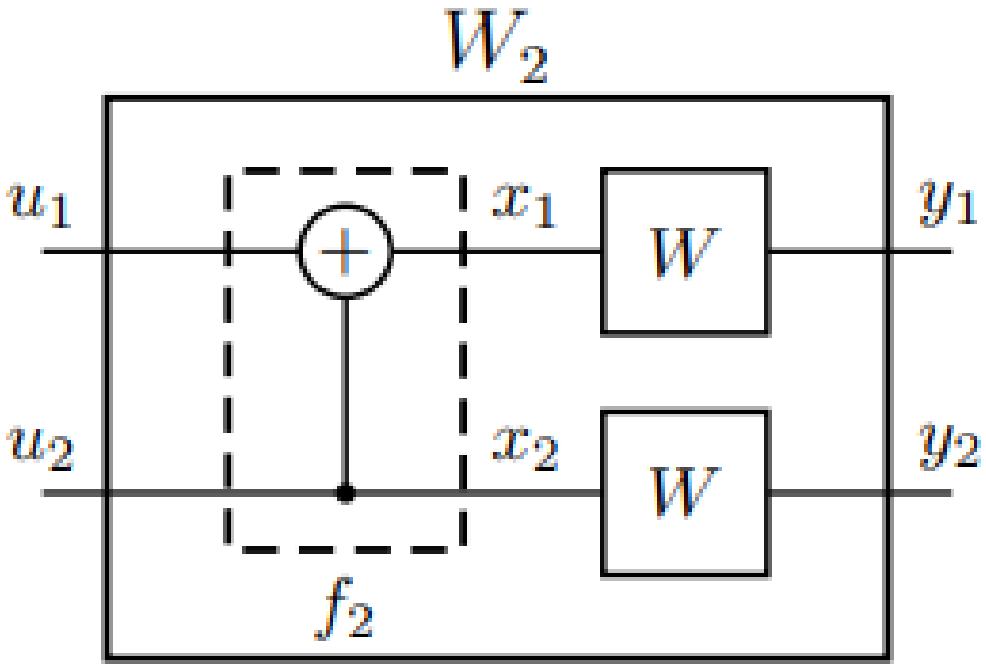


Figure 3-17 Basic model of Polar codes [5]

According to [5,7], u_1 y u_2 are i.i.d (independently identically distributed) uniform over $\{0,1\}$, and u and x are related to each other by a one-to-one mapping, thus the mutual information $I(U^2;Y^2)$ is equal to $I(X^2;Y^2)$. Moreover, since each individual channel W is independent, the total capacity of the channel W_2 is the sum of the capacity of each individual channel. It can be expressed as follows:

$$C(W_2) = I(U^2;Y^2) = I(X^2;Y^2) = 2\mathrm{C}(\mathrm{W}), \tag{3-35}$$

where $U^2 = [u_1, u_2]$, $X^2 = [x_1, x_2]$, $Y^2 = [y_1, y_2]$. This combined operation is lossless and is known as the conservation of capacity [5].

Further, if there are N independent individual channels, they can be combined into a vector channel $W_N: X^N \to Y^N$, where $N = 2^n, n \geq 0$ [6]. Then, the capacity of the vector channel is N times the capacity of the individual channels :

$$C(W_N) = N \cdot C(W)\,. \tag{3-36}$$

3.2.1.2 Channel splitting

According to [6,7], channel splitting can be considered an inverse operation with respect to channel combining. As shown in Figure 3-18, a channel W_N may be re-split to a set of N new binary-input coordinate channels: $W_N^{(i)}: X \to Y^N \times X^{i-1}, 1 \leq i \leq N$. The corresponding transition probability is defined as:

$$W_N^{(i)}\left(y_1^N, u_1^{i-1} \middle| u_i\right) \triangleq \sum_{u_{i+1}^N \in X^{N-i}} \frac{1}{2^{N-1}} W_N(y_1^N | u_1^N)\,, \tag{3-37}$$

where u_i is the input of $W_N^{(i)}$, and $\left(y_1^N, u_1^{i-1}\right)$ is the output.

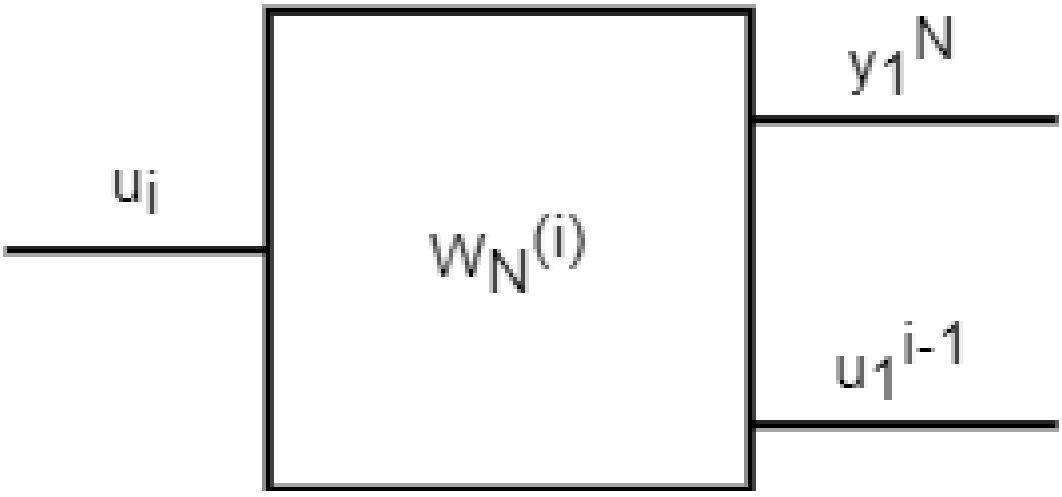


Figure 3-18 the model of a subchannel $W_N^{(i)}$

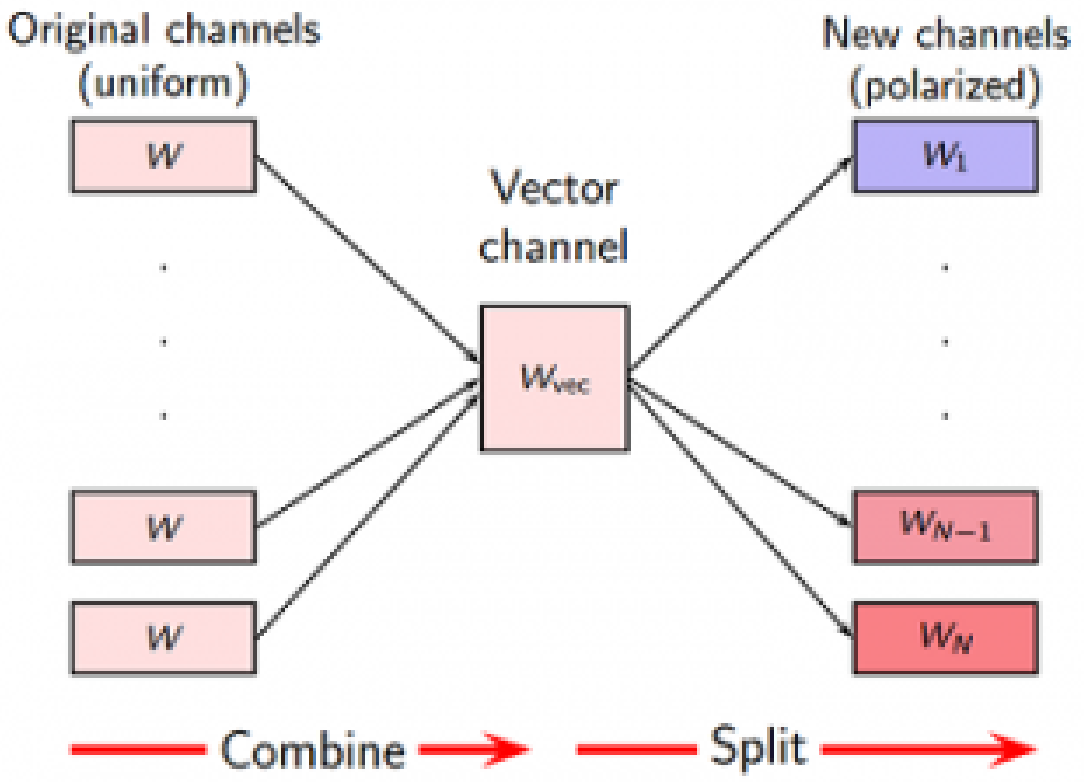


Figure 3-19 channel splitting [7]

As shown in Figure 3-19, the individual channels W are uniform and independent and are first combined into a vector channel W_{vec}, and then they are split into N new individual channels ($W_1 \cdots W_N$), where we find the phenomenon of polarization. These new individual channels are not uniform anymore, and the transition probabilities of the odd-order split subchannels and the even-order split subchannels can be obtained by two recursive equations [6]:

$$W_{2N}^{(2i-1)}\left(y_1^{2N}, u_1^{2i-1} \middle| u_{2i-1}\right) =$$

$$\sum_{u_{2i}} \frac{1}{2} W_N^{(i)}\left(y_1^N, u_{1,odd}^{2i-2} \oplus u_{1,even}^{2i-2} \middle| u_{2i-1} \oplus u_{2i}\right) \cdot W_N^{(i)}\left(y_{N+1}^{2N}, u_{1,e}^{2i-2} \middle| u_{2i}\right),$$

$$W_{2N}^{(2i)}\left(y_1^{2N}, u_1^{2i-1} \middle| u_{2i}\right) =$$

$$\frac{1}{2} W_N^{(i)}\left(y_1^N, u_{1,odd}^{2i-2} \oplus u_{1,even}^{2i-2} \middle| u_{2i-1} \oplus u_{2i}\right) \cdot W_N^{(i)}\left(y_{N+1}^{2N}, u_{1,e}^{2i-2} \middle| u_{2i}\right), \quad (3\text{-}38)$$

where $n \geq 0$, $N = 2^n$, $1 \leq i \leq N$.

3.2.1.3 Channel polarization

As mentioned in Section 3.2.1, the channel type is B-DMC, and there are two typical examples of B-DMC that satisfy the symmetry condition: binary erasure channel (BEC) and binary symmetric channel (BSC).

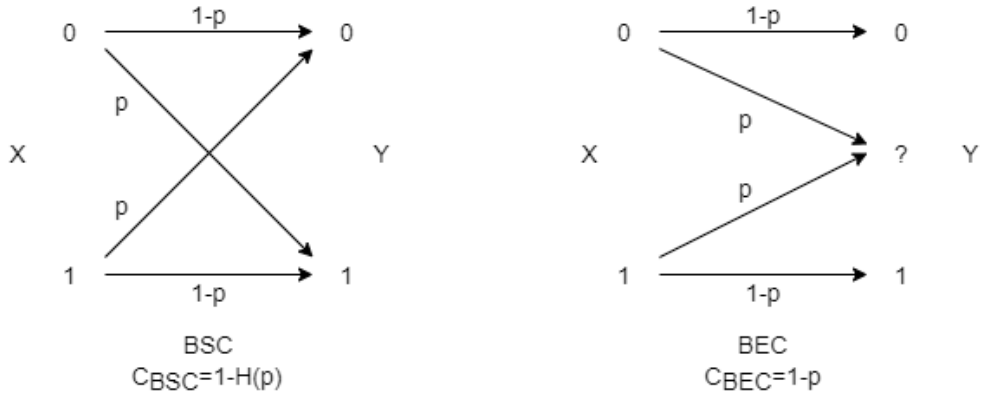


Figure 3-20 The BSC and BEC models

As shown in Figure 3-20, it can be found that both BSC and BEC have a probability of $1-p$ to keep them in their original states. However, the difference is that the BSC has a probability of p to change to another state, while the BEC has a probability of p to change to an erased state, i.e., a state that does not provide any information to the receiver. In addition, it can be observed that it will be simpler to calculate the capacity of the BEC, which only needs the value of the probability of erroneous transition p.

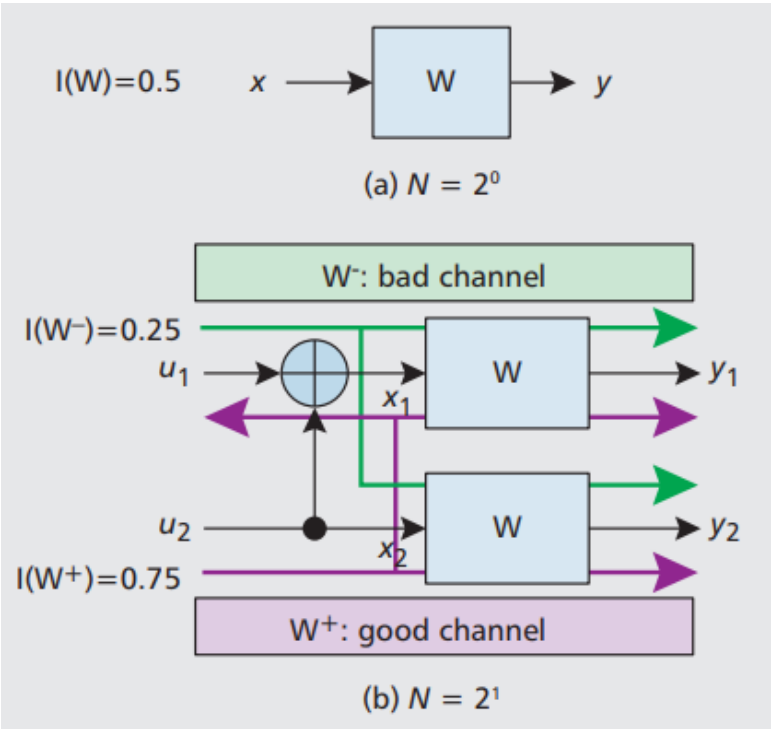


Figure 3-21 An example of channel polarization in the BEC channel [36]

As shown in Figure 3-21, based on [6], assuming that the channel W is BEC with an erasure probability $p = 0.5$, then the capacity can be calculated by using the recursive relationships:

$$C\left(W_N^{(2i-1)}\right) = C\left(W_{N/2}^{(i)}\right)^2,$$
$$C\left(W_N^{(2i)}\right) = 2C\left(W_{N/2}^{(i)}\right) - C(W_{N/2}^{(i)})^2. \quad (3\text{-}39)$$

where $i \in \{1, \cdots, N\}, N = 2^n$.

If n=0, as shown in Figure 3-21 (a), there is only one channel, the capacity is

$$C(W) = 1 - p = 0.5\,. \quad (3\text{-}40)$$

If n=1, as shown in Figure 3-21 (b), there are two individual channels, the capacity of the two channels can be calculated based on (3-29) and (3-39):

$$C\left(W_2^{(1)}\right) = C(W)^2 = 0.25,$$
$$C\left(W_2^{(2)}\right) = 2C(W) - C(W)^2 = 0.75. \quad (3\text{-}41)$$

Based on (3-36), the total capacity is:

$$C(W_2) = 2 \times C(W) = C\left(W_2^{(1)}\right) + C\left(W_2^{(2)}\right). \quad (3\text{-}42)$$

As mentioned in Section 3.2.1.2, it can be found that two independent individual channels W are combined into a vector channel W_2 and then split into two new channels $W_2^{(1)}$and $W_2^{(2)}$. However, the two newly split channels and the previous channel have different channel capacities. The relationship is shown in (3-43).

$$C\left(W_2^{(1)}\right) \le C(W) \le C\left(W_2^{(2)}\right). \quad (3\text{-}43)$$

According to [5,36], The channel $W_2^{(1)}$ with a lower capacity is referred to as the 'bad' channel. Correspondingly, the channel $W_2^{(2)}$ with a higher capacity is referred to as the 'good' channel. Typically, the notation W^- and W^+ is used for the 'bad' and 'good' channels, respectively. The equation (3-43) is also written as:

$$C(W^-) \le C(W) \le C(W^+)\,. \quad (3\text{-}44)$$

This phenomenon is called channel polarization and is the origin of the term 'polar'. Moreover, according to [6,36], polarization will become more extreme as N approaches infinity, and the polarized channel capacity will converge to two limits: one end is as an ideal channel with a capacity approximately equal to 1, and the other end will have a capacity approximately equal to 0, like pure noise.

Further, in [5], for a group of input variables $u_i, i \in \{1, \cdots, N\}$, the following notation is used: A is used to denote the set of 'active' variables, while A^c is used to

denote the set of 'frozen' variables. The 'active' variables are defined as $U_A \triangleq (U_i : i \in A)$ and the 'frozen' variables are defined as $U_{A^c} \triangleq (U_i : i \in cA^c)$, each vector is a subvector of the vector U^N. In simpler terms, 'active' and 'frozen' are the inputs to the 'good' and 'bad' channels, respectively.

3.2.2 Encoding and Decoding of Polar Codes

3.2.2.1 Polar encoding

In general, there are three Polar encoding methods: non-systematic coding, systematic coding, and generalized concatenated coding [36]. Since Erdal Arikan used non-systematic coding in [6], we will also focus on that method.

According to [6,35-36], it can be found that, like any linear code, Polar codes can be expressed in the form of a generator matrix, as shown in (3-45):

$$x_1^N = u_1^N \cdot \boldsymbol{G_N}\,, \tag{3-45}$$

$$\boldsymbol{G_N} = \boldsymbol{B_N}\boldsymbol{F}^{\otimes n}\,, \tag{3-46}$$

where,$u_1^N = \{u_1, u_2, \cdots, u_N\}$ is the data bit vector, $x_1^N = \{x_1, x_2, \cdots, x_N\}$ is the encoding vector, and $\boldsymbol{G_N}$ is the generator matrix, $N = 2^n, n \geq 0$. Moreover, 3GPP has specified the maximum value of N in [8][13]: for the uplink, $N_{max} = 2^{10} = 1024$, and for the downlink, $N_{max} = 2^9 = 512$.

The operation $\otimes$ is the Kronecker product, and it is defined as:

$$\left(\boldsymbol{A}_{m\times n} \otimes \boldsymbol{B}_{p\times q}\right)_{mp\times nq} = \begin{bmatrix} a_{11}\boldsymbol{B} & \cdots & a_{1n}\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\boldsymbol{B} & \cdots & a_{mn}\boldsymbol{B} \end{bmatrix}. \tag{3-47}$$

The matrix $\boldsymbol{F}^{\otimes n}$ is defined as:

$$F^{\otimes n} = F \cdot F^{\otimes(n-1)}\,, \tag{3-48}$$

where the matrix $\boldsymbol{F}$ is a kernel matrix [36], equal to $\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$. The matrix $\boldsymbol{B_N}$ is calculated by the following recursion:

$$\boldsymbol{B_N} = \boldsymbol{R_N}\left(I_2 \otimes \boldsymbol{B}_{N/2}\right)\,, \tag{3-49}$$

where the initial value of $\boldsymbol{B}_N$ is $\boldsymbol{B}_2 = I_2$. $\boldsymbol{R}_N$ is called the reverse shuffle permutation matrix (or odd-even permutation matrix) [35], defined by:

$$(s_1, s_2, s_3, \cdots, s_N)\boldsymbol{R}_N = (s_1, s_3, \cdots, s_{N-1}, s_2, s_4, \cdots, s_N)\,. \tag{3-50}$$

[13] In clause 7.1.4 of [8], pp. 84.

Thus, the matrix form of $\boldsymbol{R_N}$ is:

$$\boldsymbol{R_N} = \begin{bmatrix} 1 & 0 & & 0 & 0 & & 0 & 0 \\ 0 & 0 & \cdots & 0 & 1 & \cdots & 0 & 0 \\ 0 & 1 & & 0 & 0 & & 0 & 0 \\ \vdots & & \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & & 0 & 0 & & 1 & 0 \\ 0 & 0 & \cdots & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & & 0 & 0 & & 0 & 1 \end{bmatrix}. \tag{3-51}$$

For example, as shown in Figure 3-22, we can consider a setup where the input data bit vector belongs to an i.i.d uniform distribution over {0,1}, the channel is BEC with an erasure probability $p = 0.5$, the number of individual channels is $N = 8$, and the code rate is $R = 1/2$. In this case, the coding process is as follows:

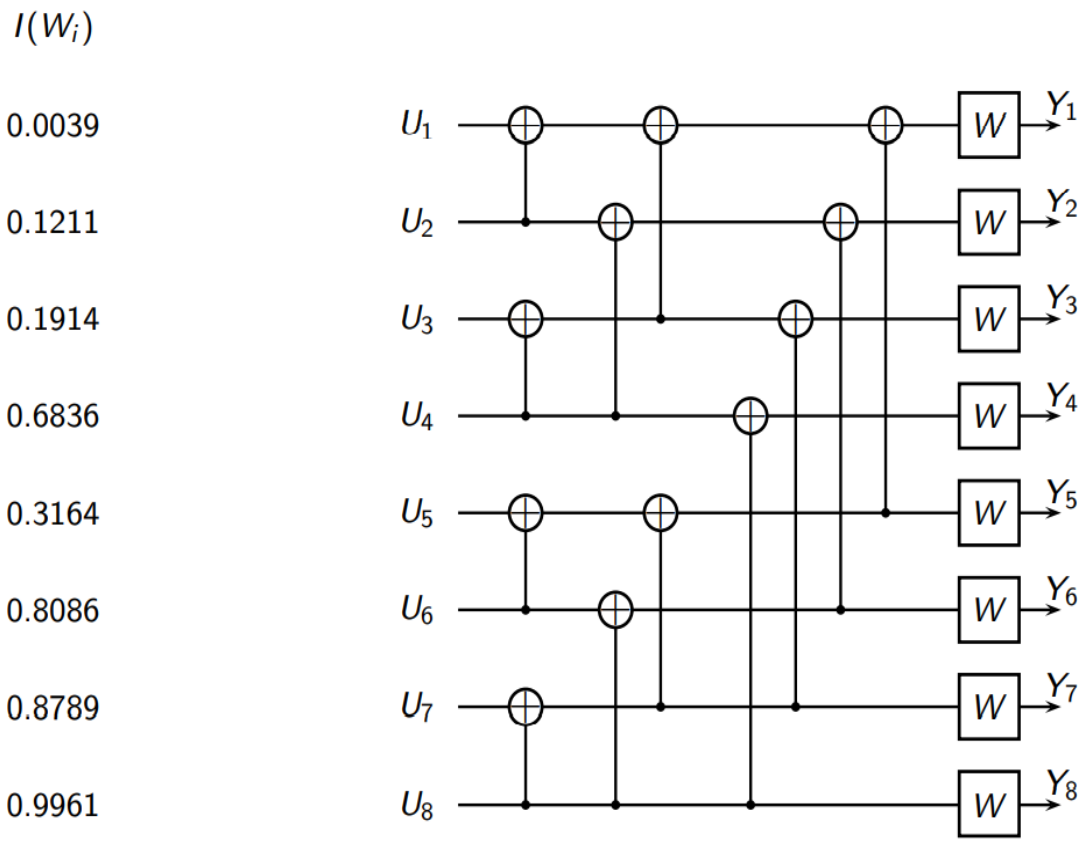


Figure 3-22 Model of the Polar codes with $N = 8$ [7]

1) Determine the 'active' variables and 'frozen' variables:

As shown in Figure 3-22, these capacities of eight individual channels have been calculated and according to (3-31), $I(W_i) = C(W_i)$ for these individual channels. As described in Section 3.2.1, the Bhattacharyya parameter is used to

evaluate the reliability of a channel transmission, i.e., the smaller $Z(W)$, the more reliable the channel transmission. Since $Z(W)$ is inversely proportional to $C(W)$, it can be found that the channels 1, 2, 3, and 5 correspond to the large values of the Bhattacharyya parameters and, as mentioned in Section 3.2.1.3, the 'active' variables and 'frozen' variables can be expressed in terms of U_A and U_{A^c}:

$$U_A = \{u_4, u_6, u_7, u_8\},$$
$$U_{A^c} = \{u_1, u_2, u_3, u_5\}.$$

In addition, assume that the 'active' bit is 1, and the 'frozen' bit is 0, then the input bits are:

$$\begin{aligned} u_1^8 &= [u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8] \\ &= [0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1]. \end{aligned} \tag{3-52}$$

2) Calculate $\boldsymbol{B_8}$:

Based on (3-49), we have:

$$\boldsymbol{B_8} = \boldsymbol{R_8}(\boldsymbol{I_2} \otimes \boldsymbol{B_4}), \tag{3-53}$$

$$\boldsymbol{B_4} = \boldsymbol{R_4}(\boldsymbol{I_2} \otimes \boldsymbol{B_2}), \tag{3-54}$$

and we can see that to calculate B_8, we first have to calculate $\boldsymbol{B_4}$ and $\boldsymbol{R_4}$. Since $\boldsymbol{B_2} = \boldsymbol{I_2}$ and $\boldsymbol{R_4}$ can be obtained from (3-50) and (3-51), we can thus write:

$$\boldsymbol{I_2} \otimes \boldsymbol{B_2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3-55}$$

$$\boldsymbol{R_4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3-56}$$

Once the value of $\boldsymbol{R_4}$ and $\boldsymbol{I_2} \otimes \boldsymbol{B_2}$ are obtained, $\boldsymbol{B_4}$ can be calculated according to (3-54):

$$\boldsymbol{B_4} = \boldsymbol{R_4}(\boldsymbol{I_2} \otimes \boldsymbol{B_2}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3-57}$$

We can calculate $\boldsymbol{I_2} \otimes \boldsymbol{B_4}$, $\boldsymbol{R_8}$, and $\boldsymbol{B_8}$ sequentially in the same way:

$$\boldsymbol{I_2} \otimes \boldsymbol{B_4} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3-58}$$

$$\boldsymbol{R_8} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3-59}$$

$$\boldsymbol{B_8} = \boldsymbol{R_8}(\boldsymbol{I_2} \otimes \boldsymbol{B_4}) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3-60}$$

3) Based on (3-48), calculate $\boldsymbol{F}^{\otimes 3}$:

$$\boldsymbol{F}^{\otimes 2} = \boldsymbol{F} \otimes \boldsymbol{F} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix},$$

$$\boldsymbol{F}^{\otimes 3} = \boldsymbol{F} \otimes \boldsymbol{F}^{\otimes 2} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{3-61}$$

4) Calculate the generator matrix $\boldsymbol{G_8}$:

Once the value of $\boldsymbol{F}^{\otimes 3}$ and R_8 are obtained, according to (3-46), we can calculate $\boldsymbol{G}_8$:

$$\boldsymbol{G_8} = \boldsymbol{B_8}\boldsymbol{F}^{\otimes 3} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{3-62}$$

5) Get the codeword:

Finally, substituting the value of (3-52) and (3-62) in (3-45), we can get the codeword x_1^8:

$$x_1^8 = u_1^8 \cdot \boldsymbol{G_8} = [0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1]. \tag{3-63}$$

As shown in Figure 3-23, a clumsier method, calculated node by node, the final result obtained is the same as that of (3-63) to prove the correctness of the previous method.

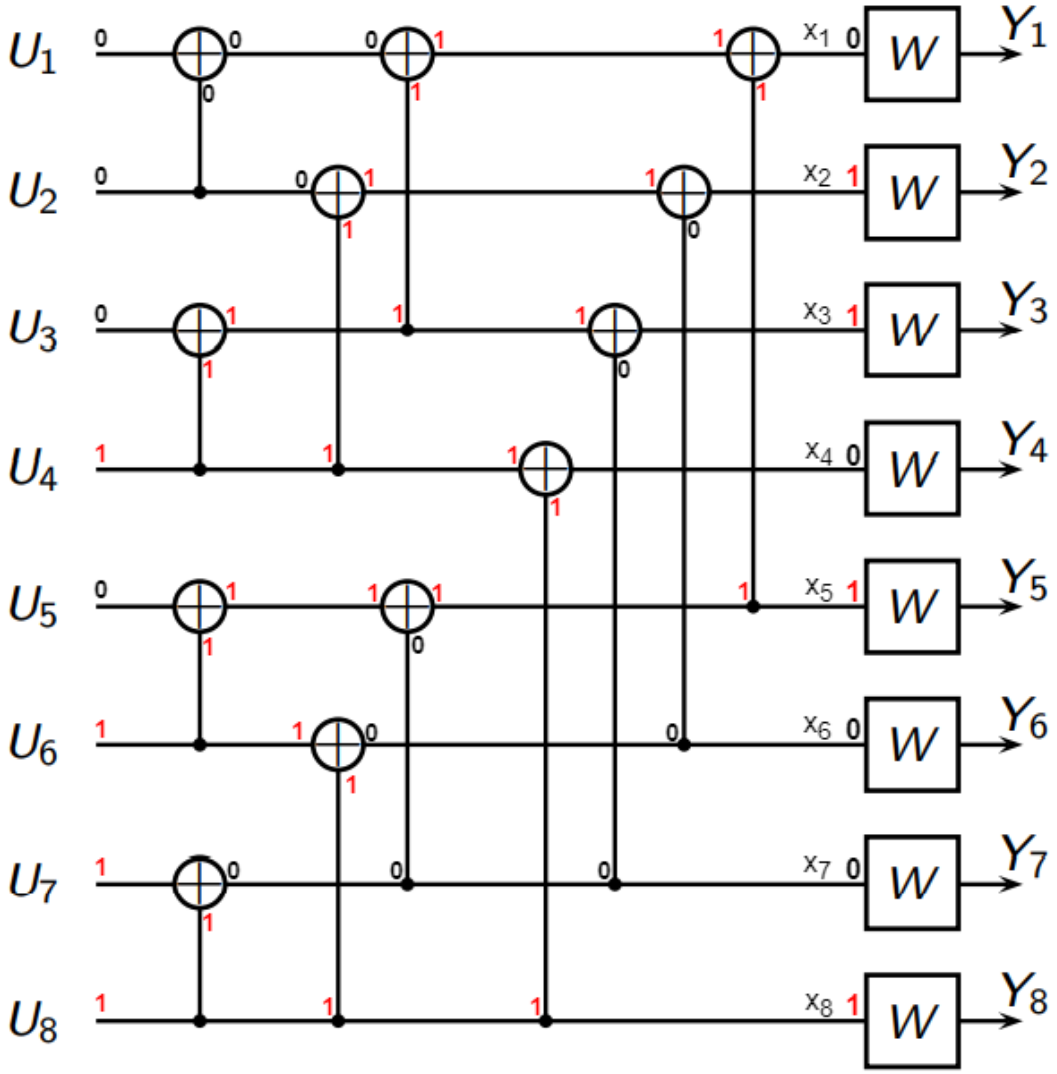


Figure 3-23 Example of the coding of Polar codes with $N = 8$

3.2.2.2 Polar decoding

Since Erdal Arikan proposed the use of the successive cancellation (SC) algorithm as a decoding algorithm for Polar codes in his paper [6], this algorithm has become one of the most basic and classical algorithms. Therefore, we will focus on this algorithm in the sequel.

According to [5-6,37-38], the estimated message can be written as:

$$\hat{u}_i = \begin{cases} u_i, & i \in A^c \\ h_i\left(y_1^N, \hat{u}_1^{i-1}\right), & i \in A \end{cases}, \tag{3-64}$$

where $i \in \{1, \cdots, N\}$ and $h_i: Y^N \times X^{i-1} \to X$ are the decision functions [37], defined as follows:

$$h_i\left(y_1^N, \hat{u}_1^{i-1}\right) = \begin{cases} 0, & \dfrac{W_N^{(i)}\left(y_1^N, \hat{u}_1^{i-1} \middle| 0\right)}{W_N^{(i)}\left(y_1^N, \hat{u}_1^{i-1} \middle| 1\right)} \geq 1 \\ 1, & otherwise \end{cases}, \tag{3-65}$$

where for all the $y_1^N \in Y^N, \hat{u}_1^{i-1} \in X^{i-1}$.

As explained in Section 3.1.1.3, the decision function can be written in the form of LLR as follows:

$$LLR\left(y_1^N, \hat{u}_1^{i-1}\right) = \ln\left(\frac{W_N^{(i)}\left(y_1^N, \hat{u}_1^{i-1} \middle| 0\right)}{W_N^{(i)}\left(y_1^N, \hat{u}_1^{i-1} \middle| 1\right)}\right). \tag{3-66}$$

Then, the equation (3-64) in [5] can be rewritten as:

$$\hat{u}_i = \begin{cases} u_i, & i \in A^c \\ 0, & i \in A \wedge LLR\left(y_1^N, \hat{u}_1^{i-1}\right) \geq 0 \\ 1, & i \in A \wedge LLR\left(y_1^N, \hat{u}_1^{i-1}\right) < 0 \end{cases}. \tag{3-67}$$

The equation (3-68) is then obtained by calculating (3-38) using the recursive formulas [6]:

$$L_N^{2i-1}\left(y_1^N, \hat{u}_1^{2\mathrm{i}-2}\right) = \frac{L_{\frac{N}{2}}^{(i)}\left(y_1^{\frac{N}{2}}, \hat{u}_{1,odd}^{2\mathrm{i}-2} \oplus \hat{U}_{1,even}^{2i-2}\right) L_{\frac{N}{2}}^{(i)}\left(y_{\frac{N}{2}+1}^N, \hat{u}_{1,even}^{2\mathrm{i}-2}\right) + 1}{L_{\frac{N}{2}}^{(i)}\left(y_1^{\frac{N}{2}}, \hat{u}_{1,odd}^{2\mathrm{i}-2} \oplus \hat{U}_{1,even}^{2i-2}\right) + L_{\frac{N}{2}}^{(i)}\left(y_{\frac{N}{2}+1}^N, \hat{u}_{1,even}^{2\mathrm{i}-2}\right)},$$

$$L_N^{2i}\left(y_1^N, \hat{u}_1^{2\mathrm{i}-1}\right) = \left[L_{\frac{N}{2}}^{(i)}\left(y_1^{\frac{N}{2}}, \hat{u}_{1,odd}^{2\mathrm{i}-2} \oplus \hat{U}_{1,even}^{2i-2}\right)\right]^{1-2\hat{u}_{2i-1}} L_{\frac{N}{2}}^{(i)}\left(y_{\frac{N}{2}+1}^N, \hat{u}_{1,even}^{2\mathrm{i}-2}\right). \tag{3-68}$$

With these two recursive formulas, the computations for a block of length N can be reduced to two computation blocks of length $N/2$. The simplification continues until $L_1^{(1)}(y_i) = W(y_i|0)/W(y_i|1)$, which can be computed directly [6].

Moreover, we can denote:

$$L_{\frac{N}{2}}^{(i)}\left(y_1^{\frac{N}{2}}, \hat{u}_{1,odd}^{2\mathrm{i}-2} \oplus \hat{U}_{1,even}^{2i-2}\right) = e^{\alpha}, \tag{3-69}$$

$$L_{\frac{N}{2}}^{(i)}\left(y_{\frac{N}{2}+1}^N, \hat{u}_{1,even}^{2\mathrm{i}-2}\right) = e^{\beta}, \tag{3-70}$$

Performing logarithmic operations on both sides of (3-68) and substituting (3-69) and (3-70) into the previous result, we have:

$$LLR_N^{2i-1}\left(y_1^N, \hat{u}_1^{2i-2}\right) = \frac{e^{\alpha+\beta}+1}{e^{\alpha}+e^{\beta}},$$

$$LLR_N^{2i}\left(y_1^N, \hat{u}_1^{2i-2}\right) = \left(1-2\hat{u}_1^{2i-2}\right)\alpha+\beta, \tag{3-71}$$

where $\alpha, \beta \in R, \hat{u}_1^{2i-2} \epsilon \{0,1\}$ [38].

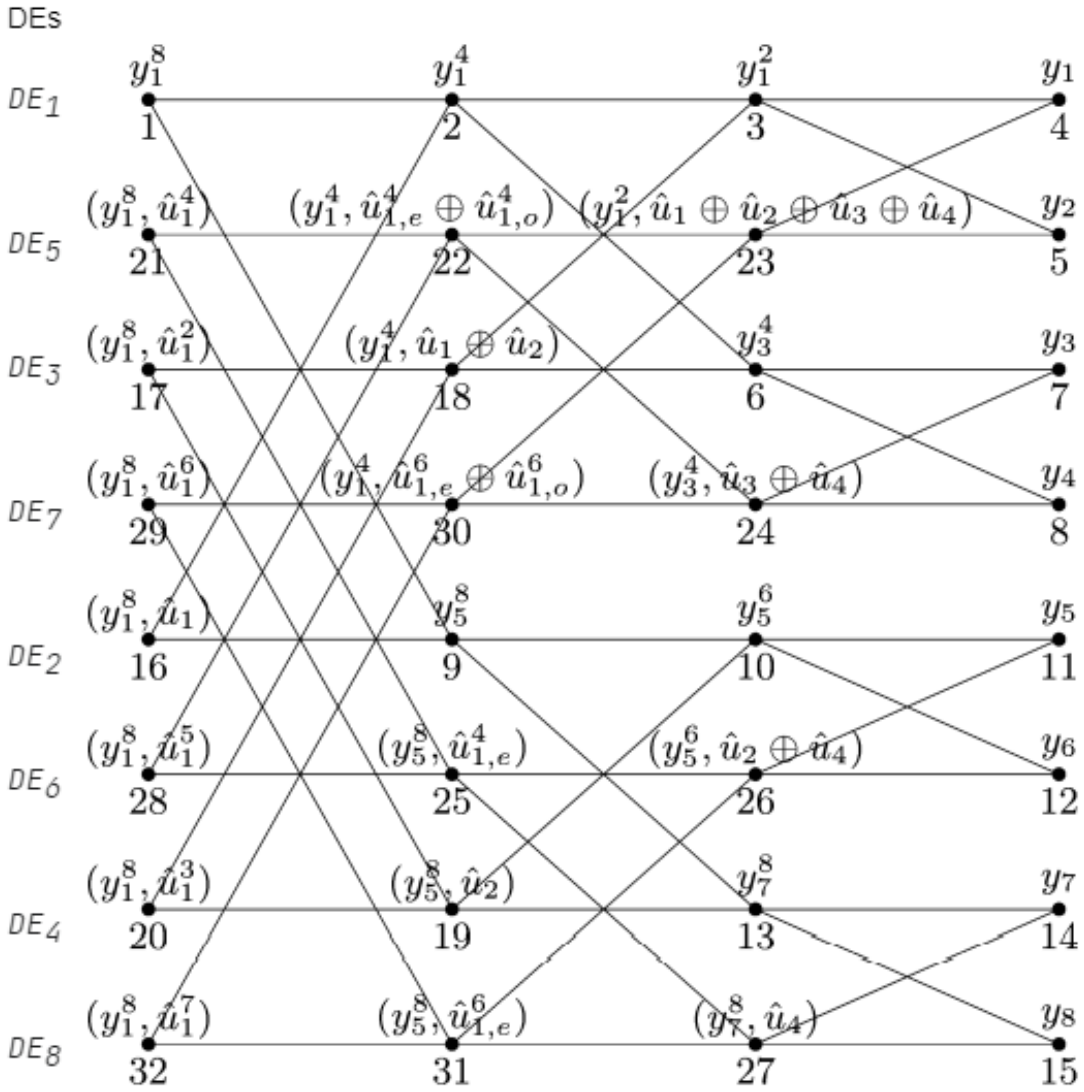


Figure 3-24 SC decoder graph for Polar codes at block length $N = 8$ [6]

As an example, we can assume that a parameter set (N, K, A, u_{A^c}) is equal to $(8, 5, \{3,5,6,7,8\}, \{0,0,0\})$, where *N* is the block length, *K* specifies the size of A, A is the information set ('active' variables) and u_{A^c} is the frozen bit set. As shown in Figure 3-24, and according to [6], there are a total of $N(1+\log_2 N) = 32$ nodes. From left to right, each column of nodes correspond to LLR requests at length 8, 4, 2, and 1 in turn, where the leftmost column is referred to as the decision level, while the rightmost column is

referred to as the channel level. The leftmost node of the decoder graph labeled $(y_1^8, \hat{u}_1^{i-1})$ is associated with the i-th Decision Element (DE), $1 \leq i \leq 8$. Thus, the order of the DEs from top to bottom is: $\{1,5,3,7,2,6,4,8\}$.

The decoding starts with DE 1. In general, a node activates all nodes connected to its right side, and the result is returned to all nodes connected to its left side. Thus, to obtain the value of $LLR_8^{(1)}(y_1^8)$ node 1 activates node 2 and node 9, at which point the control program is passed to node 2, and node 1 is pending until node 2 returns the result to node 1. In the same way, node 2 activates nodes 3 and 6, and node 3 activates nodes 4 and 5. These two nodes then return their calculated results $LLR_1^{(1)}(y_1^1)$ and $LLR_1^{(1)}(y_2^1)$ back to node 3 and node 23, respectively. Next, node 3 derives $LLR_2^{(1)}(y_1^2)$ from the messages received from nodes 4 and 5, and it returns to nodes 2 and 18. Node 6, similar to node 3, activates nodes 7 and 8, then returns its result to nodes 2 and 18. Node 2 derives $LLR_4^{(1)}(y_1^4)$ from the message received, then send it to nodes 1 and 16. Node 9 also returns $LLR_4^{(1)}(y_5^8)$ to nodes 1 and 16 in the same way. At this point, node 1 derives $LLR_8^{(1)}(y_1^8)$ from nodes 2 and 9, then sends the result obtained to DE 1. Since u_1 is a 'frozen' bit, and DE 1 ignores the received message, the choice is $\hat{u}_1 = 0$, and passes control to DE 2.

Next, DE 2 activates node 16. Since node 16 has already obtained the values of nodes 2 and 9 in the previous step, there is no need to activate other nodes, the value of $LLR_8^{(2)}(y_1^8, \hat{u}_1)$ can be directly obtained. For the same reason in the previous step, u_2 is a 'frozen' bit, DE 2 chooses $\hat{u}_2 = 0$, and passes control to DE 3.

Finally, the DE 3 activates node 17 to get the values of $LLR_8^{(3)}(y_1^8, \hat{u}_1^2)$. Node 17 activates nodes 18 and 19 because the values of nodes 3, 6, 10, and 13 have already been derived in the previous two steps, so no further activation of other nodes is needed. Since u_3 is an "active" bit, then DE 3 determines the value of $\hat{u}_3$ based on the value of $LLR_8^{(3)}(y_1^8, \hat{u}_1^2)$ in (3-67).

The rest of the nodes are chosen by DEs in the same way until all the values of $\hat{u}_i$ are obtained.

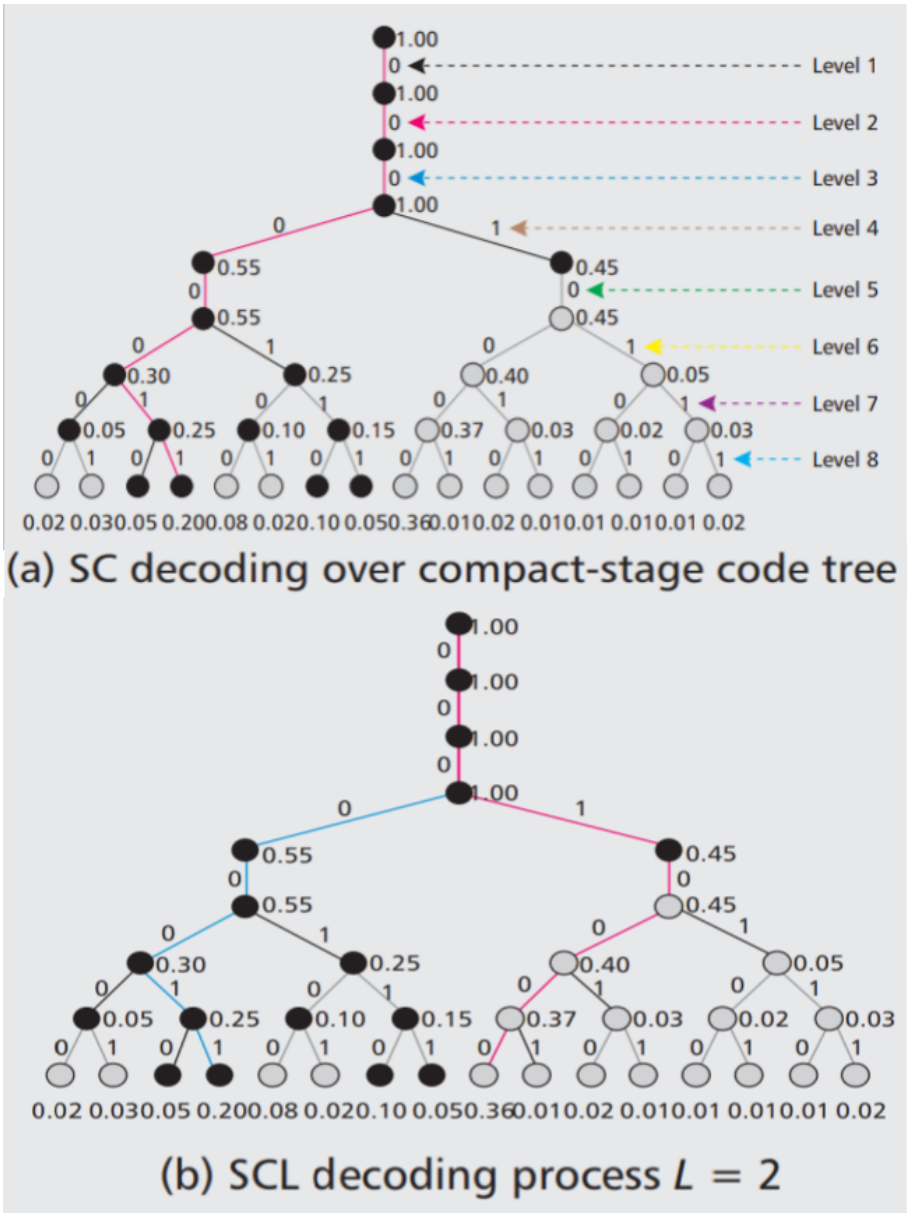


Figure 3-25 SC and SCL algorithm over code tree [36]

However, as shown in Figure 3-25 (a), it can be found that the SC algorithm performs decoding by searching the code tree level by level, and only one decision result is retained at each level, thus if an incorrect decision is made at one level, its subsequent decoding results will also be affected. Therefore, further research has proposed an improved method based on the SC algorithm called the successive cancellation list (SCL) algorithm.

Nowadays, the SCL algorithm has replaced the SC algorithm and is being widely used. According to [36-38], simply put, the SCL algorithm converts the Greedy One-Time-Pass search of the SC algorithm into a Breadth-First search [38], which allows exploring at most L candidate paths instead of the SC algorithm that can only explore one path. In some views, the SC algorithm can be considered as the SCL algorithm with $L = 1$.

As shown in Figure 3-25 (b), the SCL algorithm ($L = 2$) saves two candidate paths at each level, and finally, a more reliable path (red path) '0001 0000' was found instead of SC finding a less reliable path '0000 0011'.

3.2.3 NR Polar coding chain

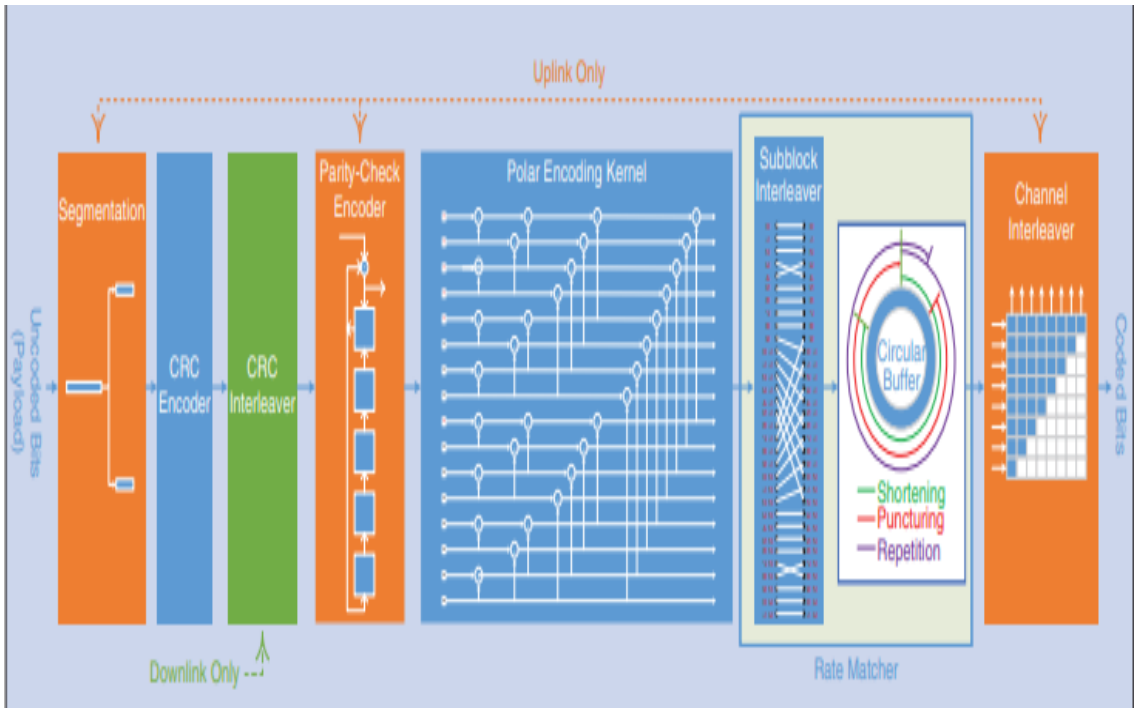


Figure 3-26 The NR Polar coding chain [16]

3.2.3.1 Segmentation

As shown in Figure 3-26, the segmentation process is performed only on the uplink. According to [8], when the payload size is $A \geq 360$ and the rate matching output sequence length $E \geq 1088$, or $A \geq 1033$, the code can be split into a miximum of two segments. The payload size A should not exceed 1706.

3.2.3.2 Rate matching

Rate matching is a process of adapting (increasing or decreasing the number of bits) the mother code length N to match the corresponding radio resources [16]. To achieve this process, the value of N must first be determined, as specified in [8][14], where the following steps are described:

[14] In clause 5.3.1 of the [8], pp. 13-14.

1) $n_1 = \begin{cases} \lceil \log_2 E \rceil - 1, & if\ E \le (9/8) \times 2^{(\lceil \log_2 E \rceil - 1)}\ y\ K/E \le (9/16) \\ \lceil \log_2 E \rceil, & otherwise \end{cases}$
2) $n_2 = \lceil \log_2(K/R_{min}) \rceil$, where K is the number of bits to encode, and R_{min} is the minimum code rate equal to $1/8$.
3) $N = 2^n$, where $n = max\{min\{n_1, n_2, n_{max}\}, 5\}$.

It should be noted that the calculation results of n_1 and n_2 are rounded up to the nearest integer, and then substituting into step 3) to calculate the maximum value n. Since the maximum mother code length (N_{max}) supported by Polar codes in 5G NR is 1024 for uplink and 512 for downlink, we can say that $n_{\max\_ul} = 10,\ n_{\max\_dl} = 9$ [39]. Finally, based on the obtained value for n, we can find the mother code length N that needs to be matched.

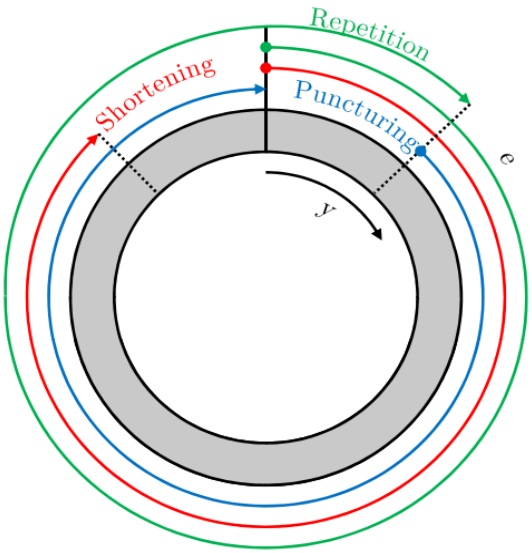


Figure 3-27 Circular buffer design for rate matching [39]

As shown in Figure 3-27, there are three matching methods: repetition, puncturing, and shortening. According to [8[15]-9,39], the output bit sequence of the interleaver y_n (see Section 3.2.3.3 for details) is written into a circular buffer as the input bit sequence of rate matching, the rate matching output size is denoted by E, therefore, the output bit sequence $e_k, k \in \{0,1,\cdots,E-1\}$ is defined as follows:

- Repetition (as shown by the green line)
  if $E \ge N$, $e_k = y_{mod(k,N)}$. It can be found that the first $(E-N)$ bits are transmitted twice.
- Shortening (as shown by the red line)
  if $E < N$ and $K/E > 7/16$, $e_k = y_k$. It can be found that the code is shortened and the last $(N-E)$ bits are not transmitted,

[15] In Clause 5.4.1.2 of [8], pp. 29.

- Puncturing (as shown by the blue line)
  if $E < N$ and $K/E \leq 7/16$, $e_k = y_{k+N-E}$. It can be found that the code is punctured and the first $(N-E)$ bits are not transmitted.

3.2.3.3 Interleaving

As shown in Figure 3-26, the interleaving process consists of three parts: the CRC interleaver, the sub-block interleaver, and the channel interleaver.

According to [16,39], the CRC interleaver works only on the downlink. The CRC bits are inserted evenly into the information blocks of the mother Polar codes. The distributed CRC bits can be checked in advance during the decoding process. When the candidate path of the decoding algorithm does not pass CRC verification, this path can be terminated early, reducing complexity and decoding latency.

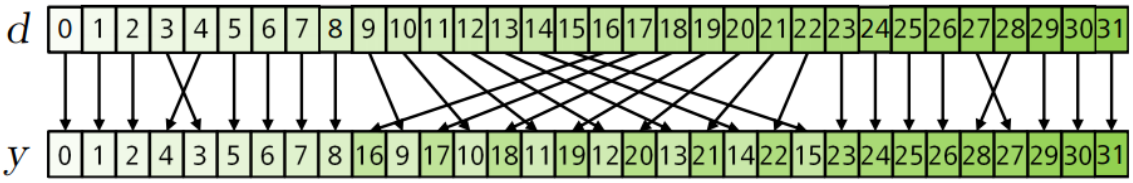


Figure 3-28 Design of the sub-block interleaver

As shown in Figure 3-26, the sub-block interleaver rearranges the coded bits before they are written into the circular buffer. As shown in Figure 3-28, and according to [39], the interleaver segments the N coded bits of block d into 32 blocks of length $N/32$ bits, and interleaves these blocks based on the corresponding 32 $p(i)$ values as shown in Figure 3-29. According to [8], the output sequence $y_n, n \in \{0, \cdots, N-1\}$ is defined as:

$$J(n) = P(i)x\left(\frac{N}{32}\right) + mod\left(n, \frac{N}{32}\right),$$

$$y_n = d_{J(n)}, \tag{3-72}$$

i	$P(i)$	i	$P(i)$	i	$P(i)$	i	$P(i)$	i	$P(i)$	i	$P(i)$	i	$P(i)$	i	$P(i)$
0	0	4	3	8	8	12	10	16	12	20	14	24	24	28	27
1	1	5	5	9	16	13	18	17	20	21	22	25	25	29	29
2	2	6	6	10	9	14	11	18	13	22	15	26	26	30	30
3	4	7	7	11	17	15	19	19	21	23	23	27	28	31	31

Figure 3-29 Sub-block interlearver pattern $p(i)$ [8][16]

[16] In clause 5.4.1.1.1 of [8], pp. 27-28.

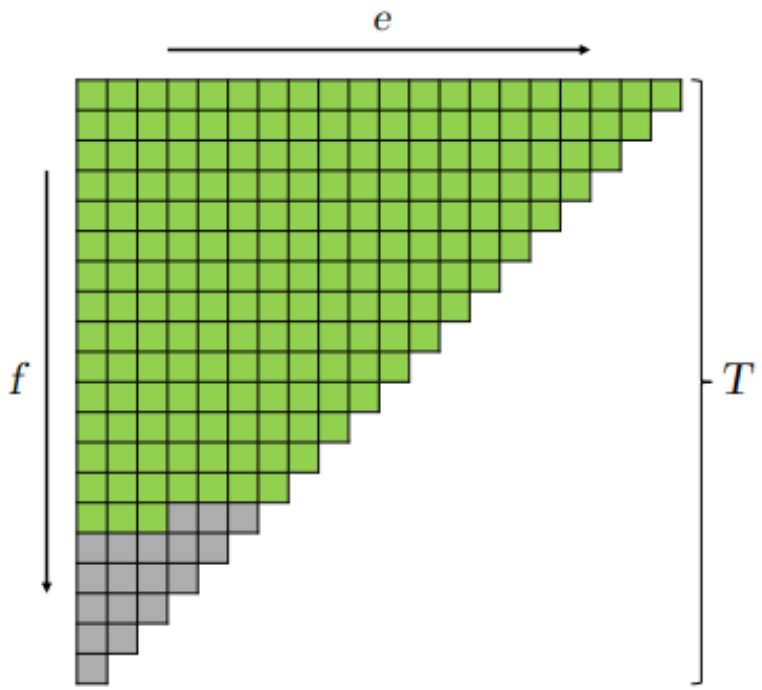


Figure 3-30 Channel interleaver [39]

As shown in Figure 3-26, in contrast to the CRC interleaver, the channel interleaver works only on the uplink. Since channel fading and reliability differences between individual bits can cause different codes to have different channel quality in uplink control information (UCI) transmissions, the channel interleaver is designed to average out this effect and improve the performance of Polar codes [16]. As shown in Figure 3-30, and according to [9,39], an isosceles triangular interleaver of length T bits is used, where denote T as the smallest integer such that $T(T+1)/2 \geq E$ [9]. The rate matching output bit sequence e is written to the interleaver row by row, and then the interleaver reads the bit sequence f column by column.

3.2.4 Conclusion

Section 3.2 describes in detail the underlying theory of Polar codes, as well as their coding and decoding algorithms, and finally also describes its specifications in 5G NR by 3GPP. It can be found that Polar code, as an emerging channel coding, attracts attention due to its excellent performance and eventually becomes the control channel coding scheme for 5G NR.

4. Simulation

4.1 Simulation environment and process

Chapter 3 clarifies the underlying theory of LDPC codes and Polar codes and their specification in 5G NR. Therefore, in this chapter, we will perform a series of experimental simulations and use these experimental results to understand these two channel codes more intuitively. We will use Matlab to evaluate the performance of these two channel codes vs Turbo codes. All simulation experiments will be performed on the same laptop (CPU: i5-8250U, RAM: 8G).

Furthermore, the simulation experiments for LDPC codes and Polar codes are based on the scripts provided in the NPTEL online course [40], and the Turbo codes are based on the pre-existing examples in the Matlab Communications Toolbox [41]. Moreover, these scripts were purposely modified before being used in the simulation experiments of this paper. These simulation scripts are detailed in Appendix 7.

Fixed parameter values	Maximum number of blocks $(\mathrm{num\_blocks})$	10 000
	$E_b/N_0\ (dB)$	-1 to 7 dB (in 0.5 dB steps)
Variable parameter values	Code rate (R)	1/3, 1/2, 5/6
	Block length (N)	128, 1024
	List size $(list\ or\ L)$	8, 32

Table 4-1 Key parameters

As shown in Table 4-1, the experiment will change the value of some key parameters to simulate different situations. Each experiment should ensure the range of E_b/N_0 and the value of $\mathrm{num\_blocks}$, and only one variable (R or N or $list$) can be changed per simulation.

First, the parameter R takes values of $\{1/3, 1/2, 5/6\}$, which correspond to three situations, i.e., low code rate, medium code rate, and high code rate, respectively. Since 5G NR has a widely use scenario, varying this parameter can help us understand the performance differences of these channel codes in different situations. Due to the specifications in the LTE standard, the Turbo codes will only simulate a code rate of $1/3$ [14].

Second, the parameter N represents the block length, where denotes $N = 128$ a short block, $N = 1024$ denotes a long block. This is because 5G NR uses two different channel coding schemes, generally, the block length of the data channel is longer while the block length of the control channel is shorter, so that the results obtained by varying this parameter can be used to justify the 5G NR channel coding scheme.

The final parameter L is only used for Polar codes and is mainly used to compare the performance difference between the SC and SCL decoding algorithms.

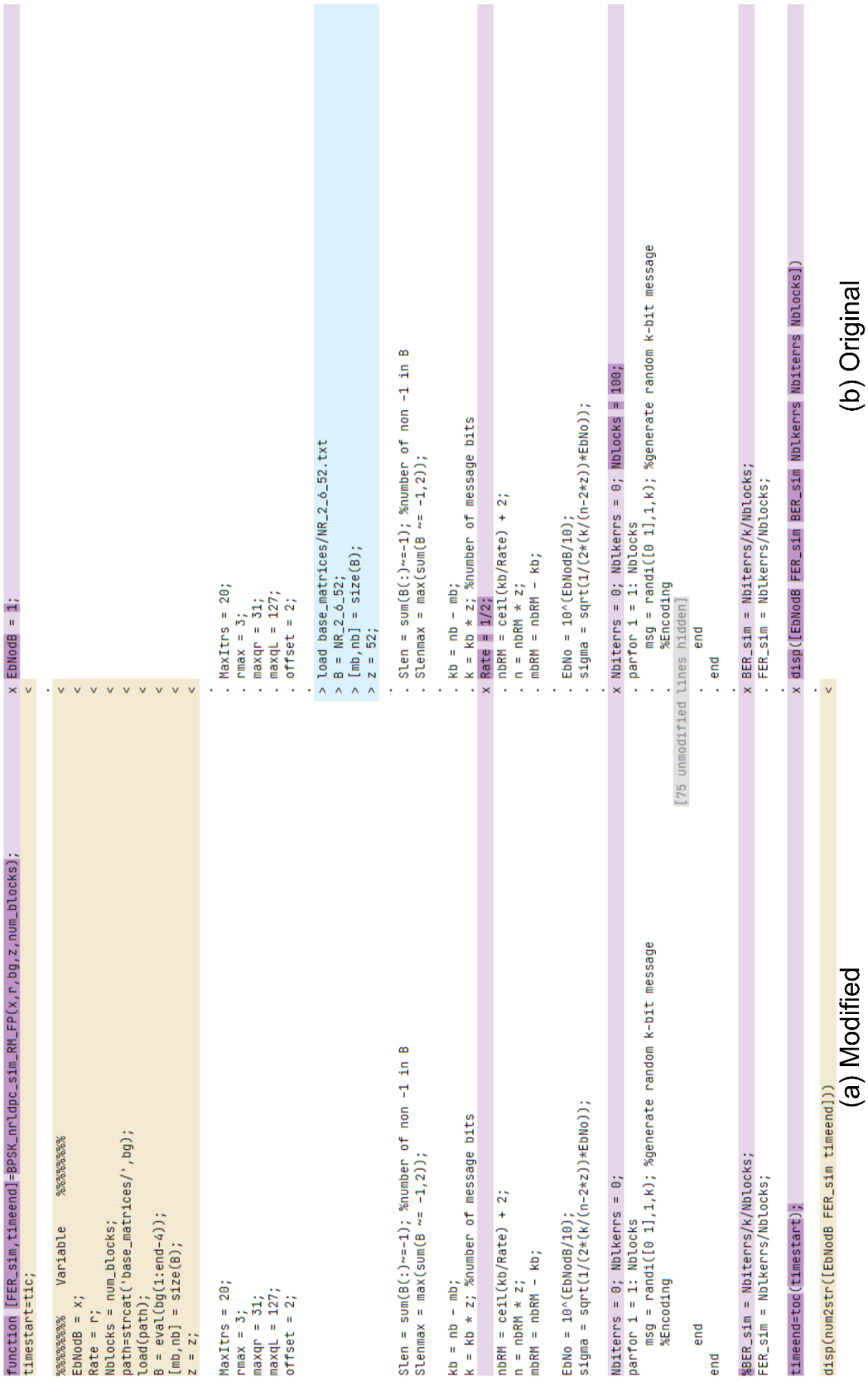


Figure 4-1 Comparisons of the source script of 'BPSK_nrldpc_sim_FP codes.m' with its modified script

Since the performance of LDPC codes must be evaluated in different cases, as shown in Figure 4-1 (b), the value of EbNodB and Rate are fixed, moreover, for different block lengths N , their base graph B and lifting size z need to be recalculated each time. Therefore, I decided to modify this script into a function (as shown in Figure 4-1 (a)) that can be called in the program 'main.m' (as shown in Appendix 7.1), which can be used to simulate different situations by changing different parameters for quick and repeated experiments.

As mentioned in the previous paragraph, it is necessary to calculate the corresponding BG and z values for different block lengths N. Therefore, referring to Section 3.1.2.2.3, I wrote a new function 'calculaZ.m' (as shown in Appendix 7.2) and called in 'main.m' to return the corresponding BG and z values for block lengths N, after passing it to the funcion 'BPSK_nrldpc_sim_FP codes.m' to facilitate the simulation.

Figure 4-2 Comparisons of the source script of 'nrpolar_sclistdecode_FP.m' with its modified script

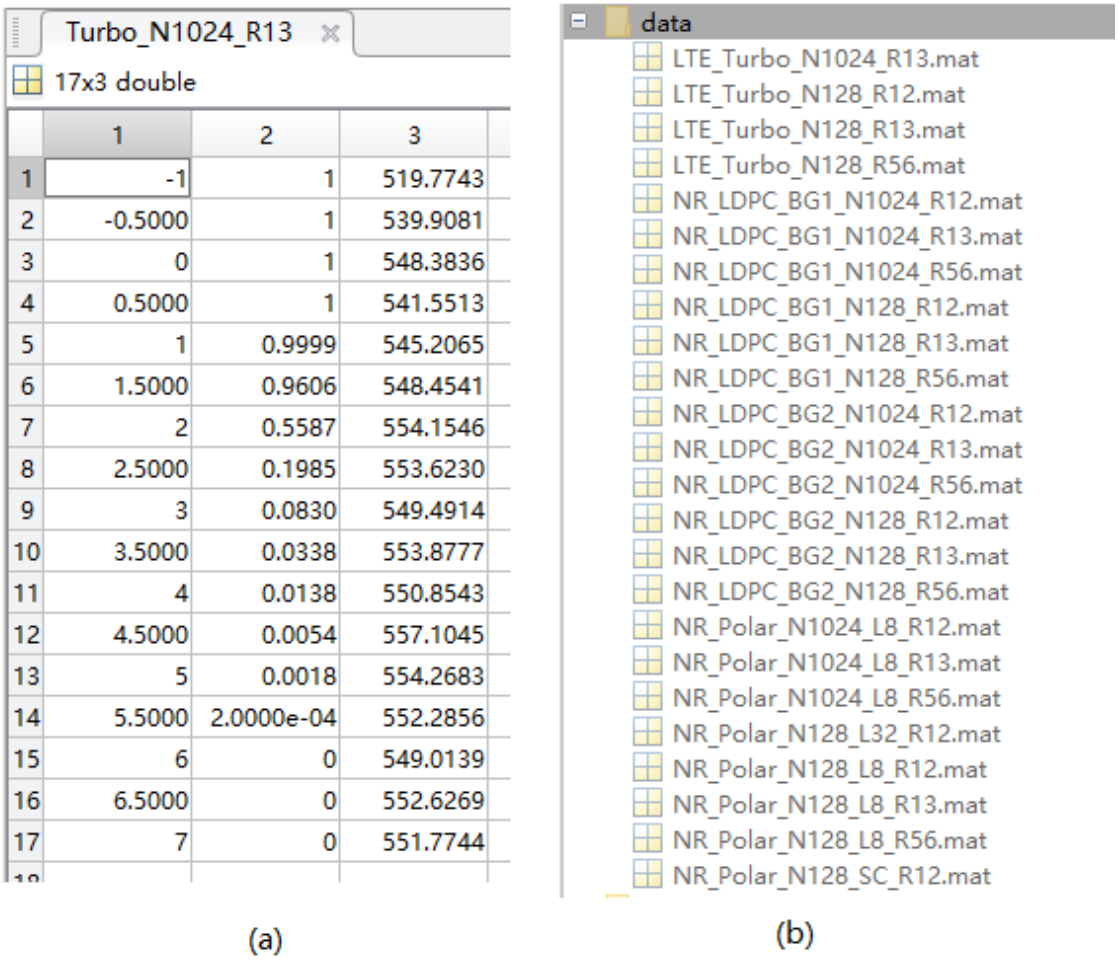
As shown in Figure 4-2, the Polar codes only need to vary the parameter R and $list\ (or\ L)$. Therefore, I also modified the two decoding algorithms SC and SCL to function form (as shown in Appendices 7.4 and 7.5) to facilitate the calls in 'main.m'.

Figure 4-3 Comparisons of the source script of 'Turbo.m' with its modified script

Similar to the LDPC and Polar codes, I also modified the original script (as shown in Figure 4-3 (b)) of the Turbo codes into a function form(as shown in Appendix 7.6) to facilitate the calls in 'main.m'. As shown in Figure 4-3, compared to the original script, I removed the 'for' loop of $ebNoIdx$, and modified the calculation equation and changed the return value ber of the original script to $ber\ and\ bler$ to ensure the consistency of the comparison results with other scripts.

Furthermore, I added the $timestart = tic$ and $timeend = toc(timestart)$ commands in each function to count the time (in seconds) used for the whole process.

Once the simulation results were obtained, as shown in Figure 4-4 (a), a new array 'codes_Nvalue_Rvalue= [EbNodB BLER time]' was created in the Matlab Command Window. Then it was saved with the 'save' command under a name whose format corresponds to''standard_codes_Nvalue_Rvalue.mat', and the files were saved in the data folder, as shown in Figure 4-4 (b).



Turbo_N1024_R13

17x3 double

	1	2	3
1	-1	1	519.7743
2	-0.5000	1	539.9081
3	0	1	548.3836
4	0.5000	1	541.5513
5	1	0.9999	545.2065
6	1.5000	0.9606	548.4541
7	2	0.5587	554.1546
8	2.5000	0.1985	553.6230
9	3	0.0830	549.4914
10	3.5000	0.0338	553.8777
11	4	0.0138	550.8543
12	4.5000	0.0054	557.1045
13	5	0.0018	554.2683
14	5.5000	2.0000e-04	552.2856
15	6	0	549.0139
16	6.5000	0	552.6269
17	7	0	551.7744

Figure 4-4 Experimental data processing and saving

By changing the parameters (R, N, and $list$) in Table 4-1, respectively, 24 results were obtained, as shown in Figure 4-4 (b). Next, two new functions were written (as

shown in Appendices 7.7 and 7.8), so that the previously saved results were imported using the $load$ command, and the $semilogy$ command was used to present the results of $BLER\ vs\ E_b/N_0(dB)$ and $TIME\ vs\ \ E_b/N_0\ (dB)$.

4.2 Results analysis

As shown in Figure 4-5, fix the codes rate R to $1/2$, and varied one parameter N $(128, 1024)$ or L $(8, 32)$ in turn, and carried out a total of 8 controlled experiments. The results are shown in Table 4-2:

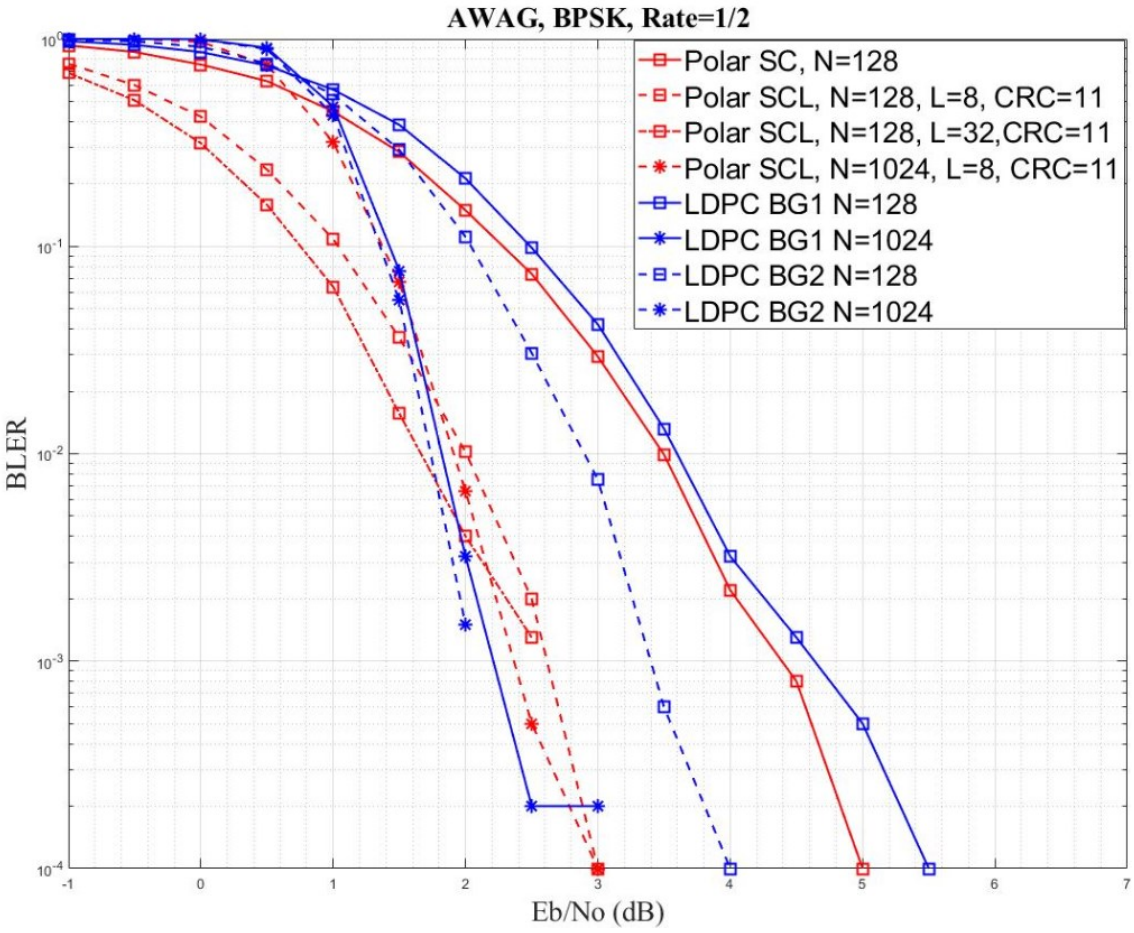


Figure 4-5 AWAG+BPSK channel@ $R = 1/2$, varying different block lengths N and list size L.

Block length (N)	List size (L)	Gain compared to LDPC codes@ $BLER = 10^{-2}$	
		BG1	BG2
128	1 (SC)	0.1 dB	-0.6 dB
	8	1.6 dB	0.9 dB
	32	2 dB	1.3 dB
1024	8	-0.1 dB	-0.2 dB

Table 4-2 Performance comparison of LDPC codes and Polar codes@ $R = 1/2$

It can be seen that Polar codes generally outperform LDPC codes for a block length $N = 128$ bits. With $BLER = 10^{-2}$ as error rate target, the SCL Polar codes have a 0.9 dB gain for $L = 8$ and 1.3 dB for $L = 32$ compared with BG2; the SC Polar codes also have 0.1 dB gain if compared with BG1 of LDPC codes.

Furthermore, when $N = 128$, comparing Polar codes with each other, it can be found that Polar codes using SCL decoding algorithm have much better performance than those using SC decoding algorithm, and the larger the list size, the better the performance of the codes.

Similarly, when $N = 128$, comparing LDPC codes with each other, it can be observed that BG2 performs much better than BG1, exhibiting a gain of 0.7 dB.

When N= 1024, the results are different and Polar codes no longer perform better than LDPC codes. When $BLER = 10^{-2}$, the gains are negative, -0.1 dB and -0.2 dB compared to BG1 and BG2, respectively. While there is no change in LDPC codes, BG2 still outperforms BG1, but the performance gap between them is significantly reduced to 0.1 dB compared to 0.7 dB in the case of the short block length.

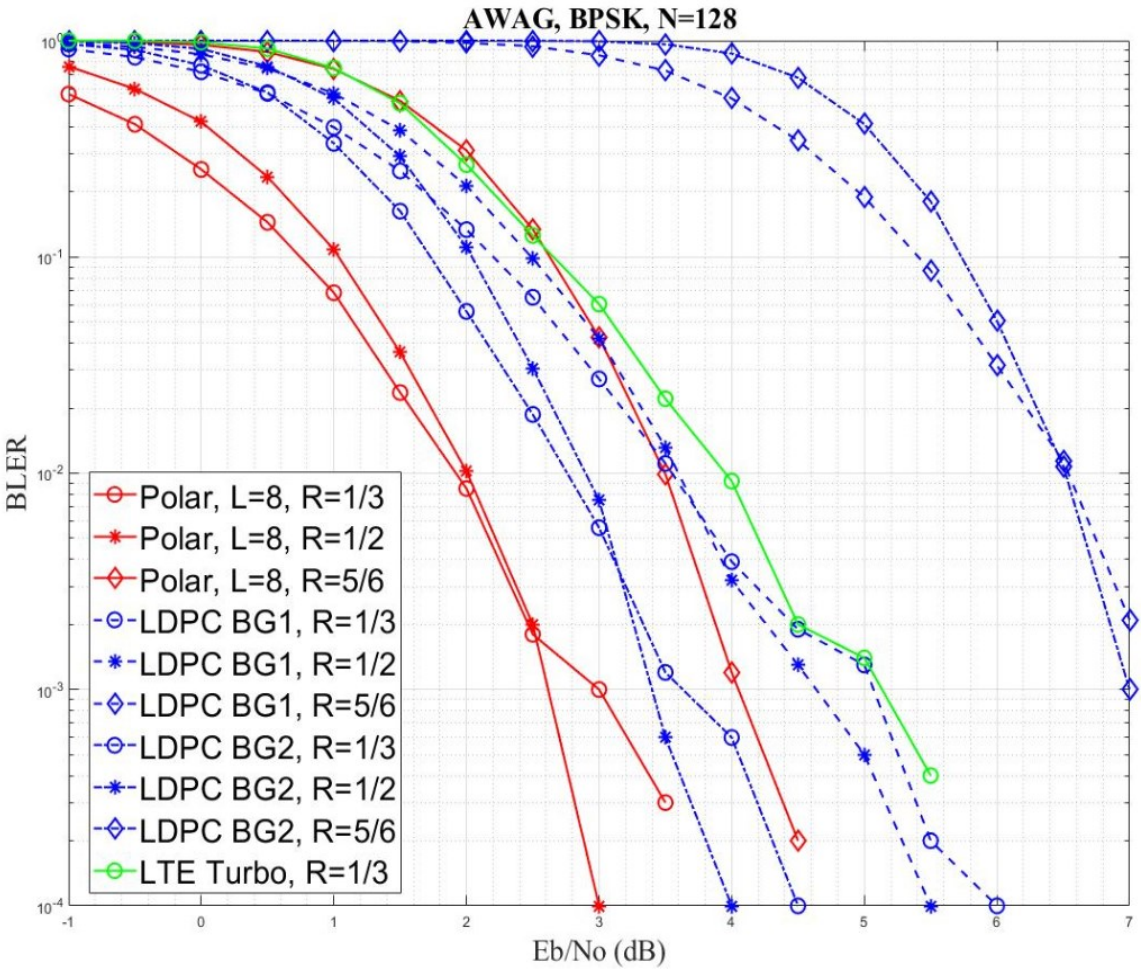


Figure 4-6 AWAG+BPSK channel@ $N = 128$, varying low, medium, and high code rate R

Code rate (R)	Gain compared to LDPC and Turbo codes@ $BLER = 10^{-2}$		
	BG1	BG2	Turbo
1/3	1.65 dB	0.85 dB	2 dB
1/2	1.6 dB	0.9 dB	N/A
5/6	3.1 dB	3 dB	N/A

Table 4-3 Performance comparison of LDPC codes and Polar codes@ $N = 128$

In the following experiments, we fixed the block length and varied the code rate. For the short block length $N = 128$, as shown in Figure 4-6, Polar codes generally outperform LDPC codes for any code rate. As shown in Table 4-3, for any code rate, Polar codes have positive gains compared to LDPC codes, and, additionally, the performance of BG2 is better than that of BG1. Especially for higher code rates, the Polar codes have a significantly better performance. Moreover, it can also be seen that both LDPC codes and Polar codes have better performance than Turbo codes when the code rate is 1/3.

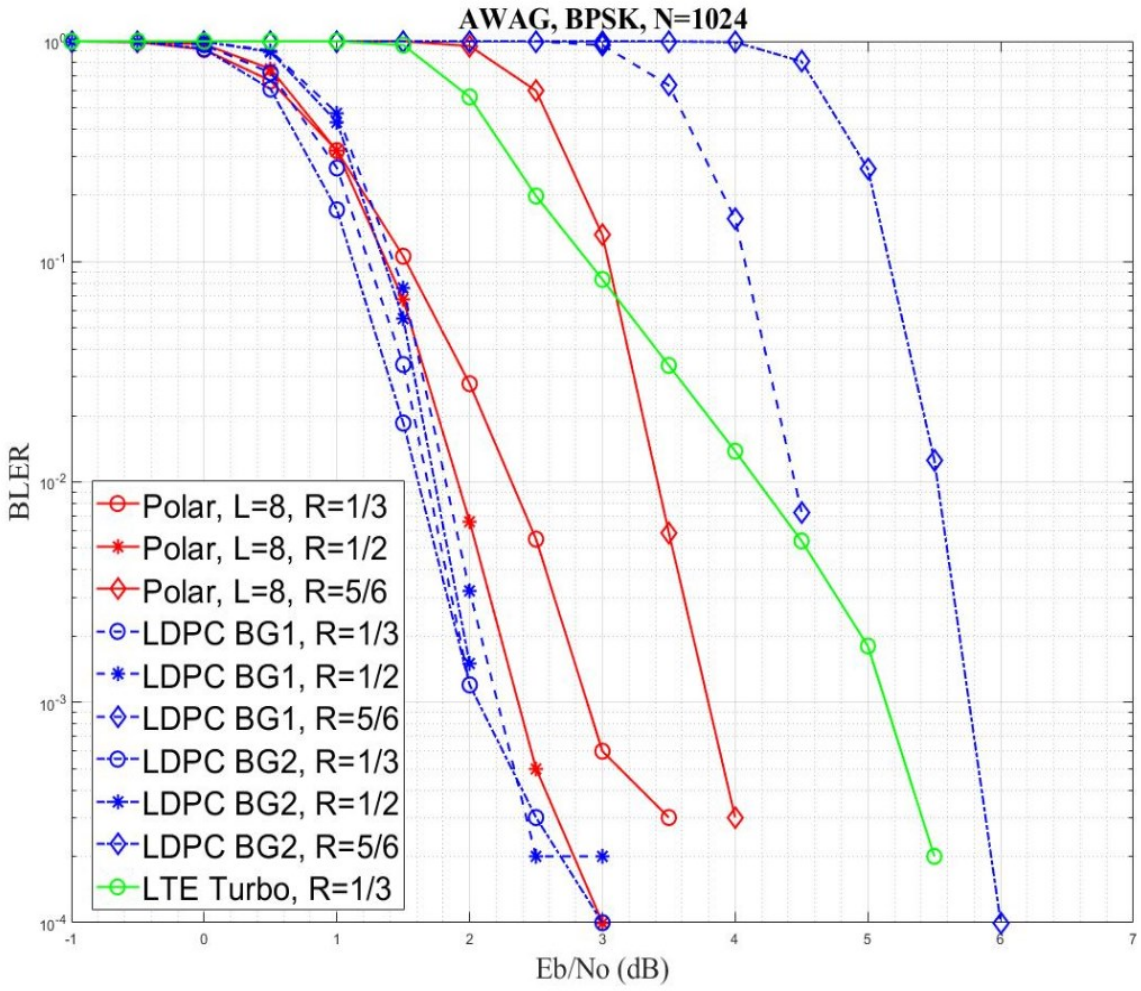


Figure 4-7 AWAG+BPSK channel@ $N = 1024$, varying low, medium, and high code rate R

Code rate (R)	Gain compared to LDPC and Turbo codes@ $BLER = 10^{-2}$		
	BG1	BG2	Turbo
1/3	-0.6 dB	-0.7 dB	1.9 dB
1/2	-0.1 dB	-0.2 dB	N/A
5/6	1.05 dB	2.1 dB	N/A

Table 4-4 Performance comparison of LDPC codes and Polar codes@ $N = 1024$

When the block length is $N = 1024$, the results are shown in Figure 4-7 and Table 4-4. In the case with code rate $R = 1/3$ and $1/2$, Polar codes perform worse than LDPC codes, as shown by the negative gains. However, for code rate $R = 5/6$, the Polar code outperforms LDPC code.

Comparing BG1 and BG2, it can be seen that for low or medium code rates, the performance gap is small, and only when the binary rate is high, the performance of BG1 is much better than BG2. When the code rate is $1/3$, the performance of Turbo codes is worse than that of the other two code types.

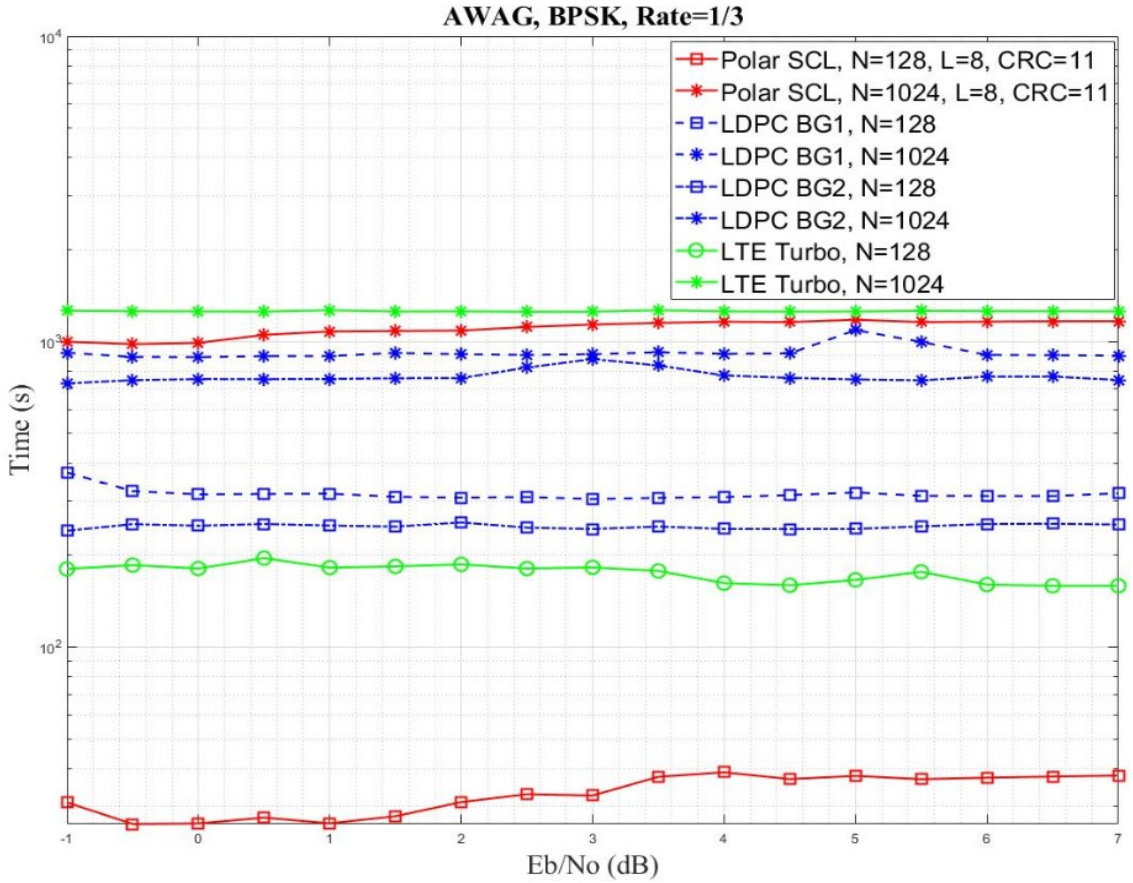


Figure 4-8 Total encoding and decoding time@ $R = 1/3$

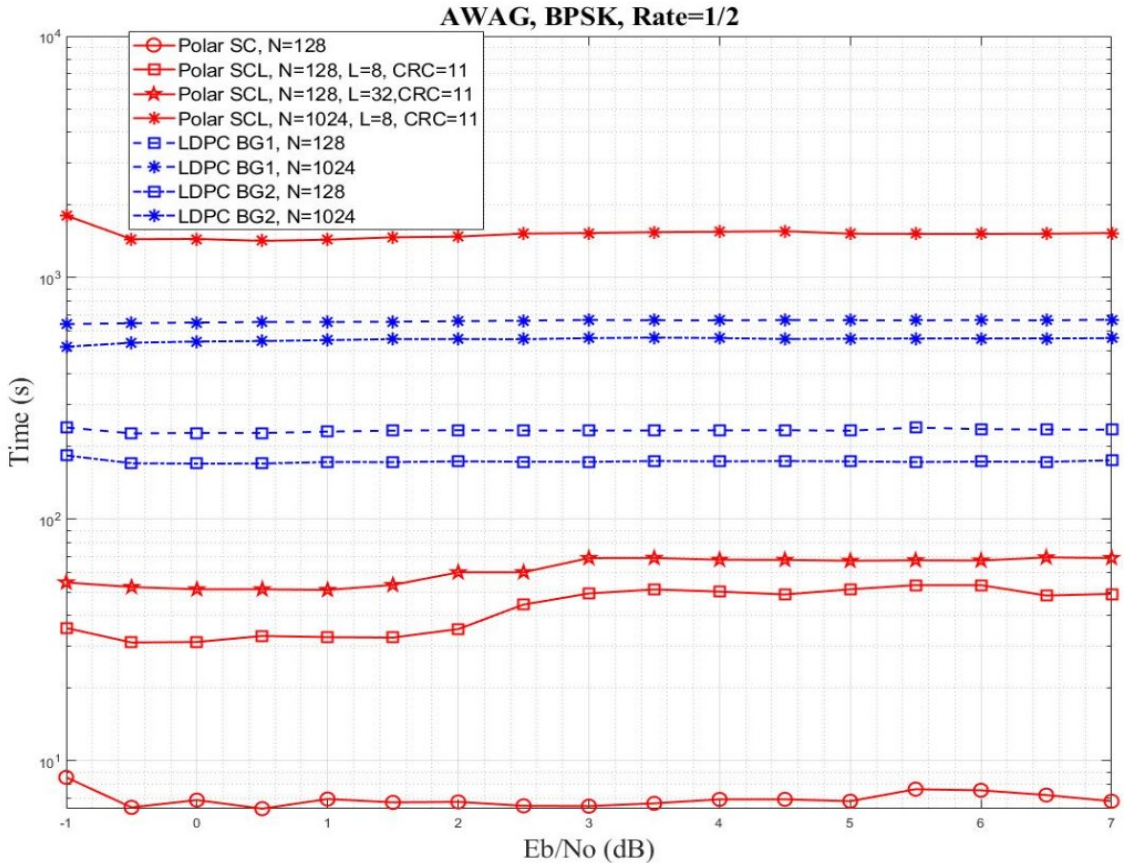


Figure 4-9 Total encoding and decoding time@ $R = 1/2$

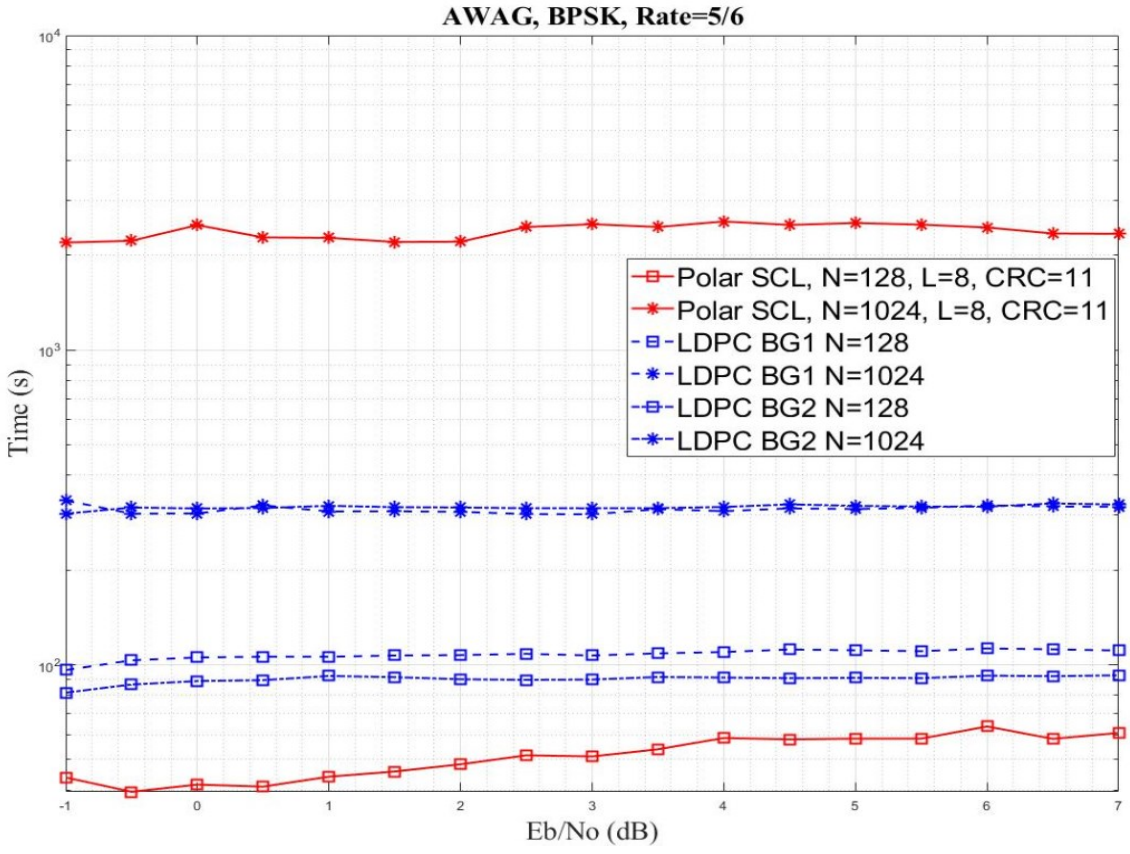


Figure 4-10 Total encoding and decoding time@ $R = 5/6$

Figure 4-8,Figure 4-9, and Figure 4-10 show the time taken for each channel code to complete the simulation for different block lengths at different code rates. It can be noticed that for short block lengths, Polar codes can complete the whole simulation quickly, regardless of the code rate. On the other hand, for long block lengths, LDPC codes outperform Polar codes, both BG1 and BG2, completing the whole simulation process at any code rate. Moreover, Turbo codes perform better than LDPC codes but worse than Polar codes at a $1/3$ code rate for short block lengths. For long block lengths, it takes longer than the other channel codes.

4.3 Discussion

It can be found through Section 4.1 that the findings of this study are restricted to $N = 128$ and 1024 and $\mathrm{num\_blocks} = 10\ 000$. Theoretically, the N and num_blocks should be increased to obtain more accurate simulation results. However, the weak performance of the laptop cannot support larger N and num_blocks.

Notwithstanding the limitations, this study does suggest that the performance of LDPC codes is proportional to the block length (N) and inversely proportional to the code rate (R). Comparing BG1 and BG2, BG1 outperforms BG2 with a high code rate and a long block length.

On the other hand, similar to LDPC codes, the performance of the Polar codes is proportional to the block length (N) and the list size ($list\ or\ L$), and inversely proportional to the code rate (R). Although Polar codes outperform LDPC codes in long block lengths, the whole process takes a long time and cannot meet the low latency communication requirements of 3GPP. In the case of the short block length, its performance is better than that of LDPC codes, and the time required is much lower than that of LDPC codes, regardless of code rate.

Finally, Turbo codes have worse performance than the other two in both short and long block lengths. The time required is also longer.

The reasons for these conclusions obtained can be attributed to the differences in principle and design between LDPC codes and Polar codes. As mentioned in Chapter 3, LDPC codes use the protograph method in 5GNR to construct its regular structure, QC-LDPC codes, and the protograph codes have high parallelism, while Polar codes are decoded less quickly than parallel decoding methods due to their serial decoding by recursive methods. Therefore, for long block size, although the improved SCL algorithm of Polar codes is similar to parallel decoding when L is large enough, the cost is to require a too high complexity to achieve a good performance and short latency, while LDPC codes strike a good balance between complexity and performance due to their nature parallelism, and for short block size, the low complexity decoding algorithm of Polar codes exhibits great advantages and has unmatched performance and latency. In

addition, the Turbo codes have worse performance and latency than the other two channel codes due to their high complexity of decoding.

In addition, since BG1 and BG2 are specified differently. As shown in Table 3-3, BG1 supports a larger code block size, and BG2 supports smaller code rates. Therefore, in general, BG1 is more suitable for high code rates and large code block sizes, while BG2 is mostly used for small code rates and small code block sizes.

Therefore, this study makes it possible to clarify the reasons for the choice of LDPC and Polar codes for 5G NR. Comparing these new channel codes with Turbo codes, the excellent performance of these new codes is highlighted, with LDPC codes as the coding scheme for the data channel due to their better performance and relatively low latency at long block lengths, and Polar codes as the coding scheme for the control channel due to their unmatched performance in short block lengths only.

5. Conclusion

In this paper, two channel codes, LDPC codes and Polar codes, of the 5G NR channel coding scheme are described in detail. Each code is described with its basic theory, coding and decoding algorithm, and its specification in 5G NR. Furthermore, we understand the unique features of each of these two channel codes, i.e., the parallelism of LDPC codes and the polarization and recursion characteristics of Polar codes. Moreover, by comparing with the LTE Turbo codes, we highlight the excellent performance of these two channel codes.

However, during the study process, we also found some limitations and drawbacks of these two channel codes. Compared with LDPC codes, Polar codes perform better than LDPC codes in short block size due to the lower complexity of the SC algorithm. However, due to the recursive characteristics of this algorithm, which makes Polar codes unlike LDPC codes with nature parallelism, perform worse in long block size.

In the case of LDPC codes, although 3GPP defines a series of regulations, such as the design of the base graph, the process of rate matching and interleaving to reduce its complexity and improve its performance, its complexity is still an obstacle to achieve the best performance under practical industrial hardware constraints. On the other hand, for Polar codes, although the interleaver with CRC distribution and the SCL algorithm can be used to reduce the decoding time, the latency is still high for long block size.

As the Focus Group on Technologies for Network 2030 (FG-NET-2030) describes in its report:“ In the next decade, ubiquitous connectivity will be more pronounced; among everything, by whichever means and everywhere.” Therefore, in the coming years, it will be a challenge to optimize and improve existing channel coding schemes or to use artificial intelligence to create new ones to meet the demand for more diverse and more complex application scenarios of 6G.

Finally, I sincerely hope that the following students will continue my study to explore the evolution and improvement of channel coding schemes in 5.5G or 6G, to find out how they meet the requirements of increasingly complex application scenarios.

6. References

[1] S. Lin and D. J. Costello, Error control coding : fundamentals and applications, 2nd ed., Pearson-Prentice Hall, 2004.

[2] M. Baldi, QC-LDPC code-based cryptography, Springer, Cham, 2014.

[3] T. T. B. Nguyen, T. Nguyen Tan and H. Lee, "Efficient QC-LDPC Encoder for 5G New Radio," *Electronics,* vol. 8, no. 6, p. 668, 2019.

[4] H. Li, B. Bai, X. Mu, J. Zhang and H. Xu, "Algebra-Assisted Construction of Quasi-Cyclic LDPC Codes for 5G New Radio," in *IEEE Access*, vol. 6, 2018, pp. 50229-50244.

[5] E. Arikan, "On the Origin of Polar Coding," *IEEE Journal on Selected Areas in Communications,* vol. 34, no. 2, pp. 209-223, Feb. 2016.

[6] E. Arikan, "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, Jul. 2009.

[7] E. Arikan, "Polar Codes Tutorial," 16 Jan. 2015. [Online]. Available: https://simons.berkeley.edu/talks/erdal-arikan-2015-01-14.

[8] 3GPP, "5G; NR; Multiplexing and channel coding (3GPP TS 38.212 version 16.3.0 Release 16)," 2020. [Online]. Available: https://cutt.ly/QmTIC0F.

[9] J. H. Bae, A. Abotabl, H.-P. Lin, K.-B. Song and J. Lee, "An overview of channel coding for 5G NR cellular communications," *APSIPA Transactions on Signal and Information Processing,* vol. 8, p. e17, 2019.

[10] G. L. Stüber, "Introduction," in *Principles of Mobile Communication*, 4th ed., Atlanta, USA, Springer, Cham, 2017, pp. 1-9.

[11] V. Pereira, T. Sousa, P. Mendes and E. Monteiro, "Evaluation of Mobile Communications: From Voice Calls to Ubiquitous," 2004.

[12] 3GPP, "Universal Mobile Telecommunications System (UMTS); Multiplexing and channel coding (FDD) (3GPP TS 25.212 version 16.0.0 Release 16)," July 2020. [Online]. Available: https://bit.ly/3Bul5tm.

[13] 3GPP2, "Physical Layer Standard for cdma2000 Spread," Sep 2009. [Online]. Available: https://bit.ly/3kLfuZN.

[14] 3GPP, "LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding (3GPP TS 36.212 version 14.15.0 Release 14)," 2021. [Online]. Available: https://bit.ly/3yC4Bg6.

[15] E. Dahlman, S. Parkvall and J. Sköld, "OFDM Transmission," in *4G LTE LTE-Advanced for Mobile Broadband*, 2011, pp. 27-44.

[16] D. Hui, S. Sandberg, Y. Blankenship, M. Andersson and L. Grosjean, "Channel Coding in 5G New Radio: A Tutorial Overview and Performance Comparison with 4G LTE," *Vehicular Technology Magazine,* vol. 13, no. 4, pp. 60-69, Dec. 2018.

[17] "Setting the scene for 5G: opportunities and challenges," 2018. [Online]. Available: http：//handle.itu.int/11.1002/pub/811d7a5f-en.

[18] E. Dahlman, S. Parkvall and J. Skold, "What is 5G?; 5G Standardization," in *5G NR: The Next Generation Wireless Access Technology*, 1st ed., Academic Press, 2018, pp. 1-24.

[19] Colegio Oficial de Ingenieros de Telecomunicación, "Modulación y multiacceso en 5G," 2018.

[20] A. V. Lopez, A. Chervyakov, G. Chance, S. Verma and Y. Tang, "Opportunities and Challenges of mmWave NR," *IEEE Wireless Communications,* vol. 26, no. 2, pp. 4-6, Apr 2019.

[21] J. Flores de valgas, J. F. Monserrat and H. Arslan, "Flexible Numerology in 5G NR: Interference Quantification and Proper Selection Depending on the Scenario," *Mobile Information Systems,* vol. 2021, 2021.

[22] 3GPP, "5G; NR; Base Station (BS) radio transmission and reception (3GPP TS 38.104 version 16.7.0 Release 16)," 29 4 2021. [Online]. Available: https://portal.etsi.org/webapp/workprogram/Report_WorkItem.asp?WKI_ID=62878.

[23] C. B. Schlegel and L. C. Perez, "A Brief History–The Drive Towards Capacity," in *Trellis and Turbo Coding: Iterative and Graph-Based Error Control Coding*, Wiley-IEEE Press, 2015, pp. 20-22.

[24] Z. Tu and S. Zhang, "Overview of LDPC Codes," *7th IEEE International Conference on Computer and Information Technology (CIT 2007),* pp. 469-474, 2007.

[25] O. Iscan, D. Lentner and W. Xu, "A Comparison of Channel Coding Schemes for 5G Short Message Transmission," *2016 IEEE Globecom Workshops (GC Wkshps),* pp. 1-6, 2016.

[26] J. Xu, F. Peng and J. Xu, "Research on 5G NR Channel Coding," *邮电设计技术,* pp. 16-21, MAR. 2019.

[27] E. Ram and Y. Cassuto, "LDPC Codes with Local and Global Decoding," in *2018 IEEE International Symposium on Information Theory (ISIT)*, 2018.

[28] S. J. Johnson, "Introducing Low-Density Parity-Check Codes," [Online]. Available: https://bit.ly/3zUoMaG.

[29] R. Jose and A. Pe, "Analysis of hard decision and soft decision decoding algorithms of LDPC codes in AWGN," *2015 IEEE International Advance Computing Conference (IACC),* pp. 430-435, 2015.

[30] A. A. Ovchinnikov and A. A. Fominykh, "Decoding of LDPC Codes for 5G Standard Using Source Distribution," *2020 Wave Electronics and its Application in Information and Telecommunication Systems (WECONF),* pp. 1-5, 2020.

[31] D. Peng, S. Zhu and J. Song, "A Novel Construction of QC-LDPC Codes Based on Combinatorial Mathematics," *Procedia Computer Science,* vol. 131, pp. 786-792, 2018.

[32] H. Wu and H. Wang, "A High Throughput Implementation of QC-LDPC Codes for 5G NR," in *IEEE Access*, vol. 7, 2019, pp. 185373-185384.

[33] V. L. Petrović, D. M. El Mezeni and A. Radošević, "Flexible 5G New Radio LDPC Encoder Optimized for High Hardware Usage Efficiency," *Electronics,* vol. 10, p. 1106, May 2021.

[34] F. Hamidi-Sepehr, A. Nimbalker and G. Ermolaev, "Analysis of 5G LDPC Codes Rate-Matching Design," in *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*, Porto, 2018.

[35] O. Gazi, Polar Codes, Springer, Singapore, 2019.

[36] K. Niu, K. Chen, J. Lin and Q. T. Zhang, "Polar codes: Primary concepts and practical decoding algorithms," *IEEE Communications Magazine,* vol. 52, no. 7, pp. 192-203, Jul. 2014.

[37] J. Wang and Y. Zhang, "Polar code and its characters," *Modern Electronic Technique,* vol. 35, no. 1, pp. 65-67, 2012.

[38] A. Balatsoukas-Stimming, 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, Oct. 2015.

[39] V. Bioglio, C. Condo and I. Land, "Design of Polar Codes in 5G New Radio," *IEEE Communications Surveys & Tutorials,* vol. 23, no. 1, pp. 29-40, 2020.

[40] A. Thangaraj, "NOC:LDPC and Polar Codes in 5G Standard," [Online]. Available: https://nptel.ac.in/courses/108/106/108106137/.

[41] Matlab Communications Toolbox, "High Rate Convolutional Codes for Turbo Coding," [Online]. Available: https://ww2.mathworks.cn/help/comm/ug/high-rate-convolutional-codes-for-turbo-coding.html.

7. Appendix: Matlab codes

7.1 Main.m

```
clear
clc

%%%%%%%%    Variable    %%%%%%%%
n=1024; %longitud de bloque
r=1/3; %tasa de codigo
num_blocks=10000; % numero maximo de bloques
list=8; %list size de SCL

%esta funcion solo se llama cuando se simulan los codigos LDPC
[z1,z2,bg1,bg2]=calculaZ(n,r);
bg=bg2;
z=z2;

%%%%%%%%    Definicion de matrices    %%%%%%%%
EbNodB=[-1:0.5:7].';
BLER=zeros(length(EbNodB),1);
BER=zeros(length(EbNodB),1);
time=zeros(length(EbNodB),1);

for i=1:length(EbNodB)
    %llama las distinas funciones para simular
    [bler,ber,t]=turbo(EbNodB(i),n,num_blocks);
    %[bler,t]=BPSK_nrldpc_sim_RM_FP(EbNodB(i),r,bg,z,num_blocks);
    %[bler,t]=nrpolar_scdecode_FP(EbNodB(i),r,n,num_blocks);
    %[bler,t]=nrpolar_sclistdecode_FP(EbNodB(i),r,n,list,num_blocks);

    BLER(i)=bler;
    BER(i)=ber;
    time(i)=t;
end
disp(num2str([EbNodB BLER BER time]))
```

Published with MATLAB® R2020b

7.2 calculaZ.m

```
function [z1,z2,BG1,BG2]=calculaZ(n,r)

% calcula z
table=[
2 4 8 16 32 64 128 256;%iLS=0
3 6 12 24 48 96 192 384;%iLS=1
5 10 20 40 80 160 320 0;%iLS=2
7 14 28 56 112 224 0 0;%iLS=3
9 18 36 72 144 288 0 0;%iLS=4
11 22 44 88 176 352 0 0;%iLS=5
13 26 52 104 208 0 0 0;%iLS=6
15 30 60 120 240 0 0 0;%iLS=7
```

```
];

% variable
N=n;
R=r;

% selecciona el valor de kb1 &kb2
kb1=22;
if N>640
    kb2=10;
elseif N>560
    kb2=9;
elseif N>192
    kb2=8;
else
    kb2=6;
end

info=['N=',num2str(N),', R=',num2str(R)];
disp(info)
%%%%%%% BG1 %%%%%%%
z1=ceil((N*R)/kb1);

temp=table-z1;
temp_min=min(abs(temp),[],'all');
[i,j]=find(temp==temp_min);
sol=table(i,j);
[i,j]=find(table==sol);
iLS_1=i-1;

BG1=['NR_1_',num2str(iLS_1),'_',num2str(sol),'.txt'];

%%%%%%% BG2 %%%%%%%
z2=ceil((N*R)/kb2);

temp2=table-z2;
temp_min2=min(abs(temp2),[],'all');
[i,j]=find(temp2==temp_min2);
sol2=table(i,j);
[i,j]=find(table==sol2);
iLS_2=i-1;

BG2=['NR_2_',num2str(iLS_2),'_',num2str(sol2),'.txt'];

end
```

Published with MATLAB® R2020b

7.3 BPSK_nrldpc_sim_RM_FP.m

```
function [FER_sim,timeend]=BPSK_nrldpc_sim_RM_FP(x,r,bg,z,num_blocks);
timestart=tic;

%%%%%%%    Variable    %%%%%%%
EbNodB = x;
Rate = r;
Nblocks = num_blocks;
```

```
path=strcat('base_matrices/',bg);
load(path);
B = eval(bg(1:end-4));
[mb,nb] = size(B);
z = z;

MaxItrs = 20;
rmax = 3;
maxqr = 31;
maxqL = 127;
offset = 2;

Slen = sum(B(:)~=-1); %number of non -1 in B
Slenmax = max(sum(B ~= -1,2));

kb = nb - mb;
k = kb * z; %number of message bits

nbRM = ceil(kb/Rate) + 2;
n = nbRM * z;
mbRM = nbRM - kb;

EbNo = 10^(EbNodB/10);
sigma = sqrt(1/(2*(k/(n-2*z))*EbNo));

Nbiterrs = 0; Nblkerrs = 0;
parfor i = 1: Nblocks
       msg = randi([0 1],1,k); %generate random k-bit message
       %Encoding
       cword = nrldpc_encode(B,z,msg);
    cword = cword(1:n);

    s = 1 - 2 * cword; %BPSK bit to symbol conversion
    r = s + sigma * randn(1,n); %AWGN channel I
    %Puncturing of message
    r(1:2*z) = 0;
    %quantization
    rq = floor(r/rmax*maxqr);
    rq(rq>maxqr) = maxqr;
    rq(rq<-(maxqr+1)) = -(maxqr+1);

    %Soft-decision, iterative message-passing layered decoding
    L = rq; %total belief
    itr = 0; %iteration number
    R = zeros(Slen,z); %storage for row processing
    treg = zeros(Slenmax,z); %register storage for minsum
    while itr < MaxItrs
        Ri = 0;
        for lyr = 1:mbRM
            ti = 0; %number of non -1 in row=lyr
            for col = find(B(lyr,1:nbRM) ~= -1)
                   ti = ti + 1;
                   Ri = Ri + 1;
                   %Subtraction
                   L((col-1)*z+1:col*z) = L((col-1)*z+1:col*z)-R(Ri,:);
                   %Row alignment and store in treg
                   temp = mul_sh(L((col-1)*z+1:col*z),B(lyr,col));
                   temp(temp>maxqr) = maxqr;
```

```
                temp(temp<-(maxqr+1)) = -(maxqr+1);
                treg(ti,:) = temp;
            end
            %minsum on treg: ti x z
            for i1 = 1:z %treg(1:ti,i1)
                [min1,pos] = min(abs(treg(1:ti,i1))); %first minimum
                min2 = min(abs(treg([1:pos-1 pos+1:ti],i1))); %second minimum
                S = 2*(treg(1:ti,i1)>=0)-1;
                parity = prod(S);
                %offset
                min1 = min1 - offset;
                if min1<0
                    min1 = 0;
                end
                min2 = min2 - offset;
                if min2<0
                    min2 = 0;
                end
                treg(1:ti,i1) = min1; %absolute value for all
                treg(pos,i1) = min2; %absolute value for min1 position
                treg(1:ti,i1) = parity*S.*treg(1:ti,i1); %assign signs
            end
            %column alignment, addition and store in R
            Ri = Ri - ti; %reset the storage counter
            ti = 0;
            for col = find(B(lyr,1:nbRM) ~= -1)
                    Ri = Ri + 1;
                    ti = ti + 1;
                    %Column alignment
                    R(Ri,:) = mul_sh(treg(ti,:),z-B(lyr,col));
                    %Addition
                    temp = L((col-1)*z+1:col*z)+R(Ri,:);
                    temp(temp>maxqL) = maxqL;
                    temp(temp<-(maxqL+1)) = -(maxqL+1);
                    L((col-1)*z+1:col*z) = temp;
            end
        end
        msg_cap = L(1:k) < 0; %decision
        itr = itr + 1;
    end

    %Counting errors
    Nerrs = sum(msg ~= msg_cap);
    if Nerrs > 0
             Nbiterrs = Nbiterrs + Nerrs;
             Nblkerrs = Nblkerrs + 1;
    end
end

%BER_sim = Nbiterrs/k/Nblocks;
FER_sim = Nblkerrs/Nblocks;

timeend=toc(timestart);

disp(num2str([EbNodB FER_sim timeend]))
```

Published with MATLAB® R2020b

7.4 nrpolar_scdecode_ FP.m

```
function [FER_sim,timeend]=nrpolar_scdecode_FP(x,r,N_len,num_blocks)

Tstart=tic;

%Reliability sequence % Table 5.3.1.2-1: Polar sequence 3GPP
Q=[0 1 2 4 8 16 32 3 5 64 9 6 17 10 18 128 12 33 65 20 256 34 24 36 7 129 66 512 11 40
68 130 ...
    19 13 48 14 72 257 21 132 35 258 26 513 80 37 25 22 136 260 264 38 514 96 67 41 144
28 69 42 ...
    516 49 74 272 160 520 288 528 192 544 70 44 131 81 50 73 15 320 133 52 23 134 384 76
137 82 56 27 ...
    97 39 259 84 138 145 261 29 43 98 515 88 140 30 146 71 262 265 161 576 45 100 640 51
148 46 75 266 273 517 104 162 ...
    53 193 152 77 164 768 268 274 518 54 83 57 521 112 135 78 289 194 85 276 522 58 168
139 99 86 60 280 89 290 529 524 ...
    196 141 101 147 176 142 530 321 31 200 90 545 292 322 532 263 149 102 105 304 296
163 92 47 267 385 546 324 208 386 150 153 ...
    165 106 55 328 536 577 548 113 154 79 269 108 578 224 166 519 552 195 270 641 523
275 580 291 59 169 560 114 277 156 87 197 ...
    116 170 61 531 525 642 281 278 526 177 293 388 91 584 769 198 172 120 201 336 62 282
143 103 178 294 93 644 202 592 323 392 ...
    297 770 107 180 151 209 284 648 94 204 298 400 608 352 325 533 155 210 305 547 300
109 184 534 537 115 167 225 326 306 772 157 ...
    656 329 110 117 212 171 776 330 226 549 538 387 308 216 416 271 279 158 337 550 672
118 332 579 540 389 173 121 553 199 784 179 ...
    228 338 312 704 390 174 554 581 393 283 122 448 353 561 203 63 340 394 527 582 556
181 295 285 232 124 205 182 643 562 286 585 ...
    299 354 211 401 185 396 344 586 645 593 535 240 206 95 327 564 800 402 356 307 301
417 213 568 832 588 186 646 404 227 896 594 ...
    418 302 649 771 360 539 111 331 214 309 188 449 217 408 609 596 551 650 229 159 420
310 541 773 610 657 333 119 600 339 218 368 ...
    652 230 391 313 450 542 334 233 555 774 175 123 658 612 341 777 220 314 424 395 673
583 355 287 183 234 125 557 660 616 342 316 ...
    241 778 563 345 452 397 403 207 674 558 785 432 357 187 236 664 624 587 780 705 126
242 565 398 346 456 358 405 303 569 244 595 ...
    189 566 676 361 706 589 215 786 647 348 419 406 464 680 801 362 590 409 570 788 597
572 219 311 708 598 601 651 421 792 802 611 ...
    602 410 231 688 653 248 369 190 364 654 659 335 480 315 221 370 613 422 425 451 614
543 235 412 343 372 775 317 222 426 453 237 ...
    559 833 804 712 834 661 808 779 617 604 433 720 816 836 347 897 243 662 454 318 675
618 898 781 376 428 665 736 567 840 625 238 ...
    359 457 399 787 591 678 434 677 349 245 458 666 620 363 127 191 782 407 436 626 571
465 681 246 707 350 599 668 790 460 249 682 ...
    573 411 803 789 709 365 440 628 689 374 423 466 793 250 371 481 574 413 603 366 468
655 900 805 615 684 710 429 794 252 373 605 ...
    848 690 713 632 482 806 427 904 414 223 663 692 835 619 472 455 796 809 714 721 837
716 864 810 606 912 722 696 377 435 817 319 ...
    621 812 484 430 838 667 488 239 378 459 622 627 437 380 818 461 496 669 679 724 841
629 351 467 438 737 251 462 442 441 469 247 ...
    683 842 738 899 670 783 849 820 728 928 791 367 901 630 685 844 633 711 253 691 824
902 686 740 850 375 444 470 483 415 485 905 ...
    795 473 634 744 852 960 865 693 797 906 715 807 474 636 694 254 717 575 913 798 811
379 697 431 607 489 866 723 486 908 718 813 ...
    476 856 839 725 698 914 752 868 819 814 439 929 490 623 671 739 916 463 843 381 497
930 821 726 961 872 492 631 729 700 443 741 ...
    845 920 382 822 851 730 498 880 742 445 471 635 932 687 903 825 500 846 745 826 732
```

```
446 962 936 475 853 867 637 907 487 695 746 ...
    828 753 854 857 504 799 255 964 909 719 477 915 638 748 944 869 491 699 754 858 478
968 383 910 815 976 870 917 727 493 873 701 ...
    931 756 860 499 731 823 922 874 918 502 933 743 760 881 494 702 921 501 876 847 992
447 733 827 934 882 937 963 747 505 855 924 ...
    734 829 965 938 884 506 749 945 966 755 859 940 830 911 871 639 888 479 946 750 969
508 861 757 970 919 875 862 758 948 977 923 ...
    972 761 877 952 495 703 935 978 883 762 503 925 878 735 993 885 939 994 980 926 764
941 967 886 831 947 507 889 984 751 942 996 ...
    971 890 509 949 973 1000 892 950 863 759 1008 510 979 953 763 974 954 879 981 982
927 995 765 956 887 985 997 986 943 891 998 766 ...
    511 988 1001 951 1002 893 975 894 1009 955 1004 1010 957 983 958 987 1012 999 1016
767 989 1003 990 1005 959 1011 1013 895 1006 1014 1017 1018 ...
    991 1020 1007 1015 1019 1021 1022 1023]+1;

%%%%%%%%    Variable    %%%%%%%%
Rate = r;
N = N_len;
K = floor(N*Rate);
n = log2(N);
EbNodB = x;
Nblocks = num_blocks;

rmax = 4; %max received value
maxqr = 31; %max integer received value

EbNo = 10^(EbNodB/10);
sigma = sqrt(1/(2*Rate*EbNo));

Q1 = Q(Q<=N); %reliability sequence for N

F = Q1(1:N-K); %Frozen positions: Q1(1:N-K)
%Message positions: Q1(N-K+1:end)

%Simulate
Nbiterrs = 0; Nblkerrs = 0;
parfor blk = 1:Nblocks
    msg = randi([0 1],1,K); %generate random K-bit message

    u = zeros(1,N);

    u(Q1(N-K+1:end)) = msg; %assign message bits

    m = 1; %number of bits combined
    for d = n-1:-1:0
        for i = 1:2*m:N
            a = u(i:i+m-1); %first part
            b = u(i+m:i+2*m-1); %second part
            u(i:i+2*m-1) = [mod(a+b,2) b]; %combining
        end
        m = m * 2;
    end
    cword = u;

    s = 1 - 2 * cword; %BPSK bit to symbol conversion
    r = s + sigma * randn(1,N); %AWGN channel I
    %quantization
    rq = floor(r/rmax*maxqr);
```

```
    rq(rq>maxqr) = maxqr;
    rq(rq<-(maxqr+1)) = -(maxqr+1);

    %SC decoder
    L = zeros(n+1,N); %beliefs
    ucap = zeros(n+1,N); %decisions
    ns = zeros(1,2*N-1); %node state vector

    satx = @(x) min(max(x,-(maxqr+1)),maxqr); %saturate FP value
    f = @(a,b) (1-2*(a<0)).*(1-2*(b<0)).*min(abs(a),abs(b)); %minsum
    g = @(a,b,c) satx(b+(1-2*c).*a); %g function

    L(1,:) = rq; %belief of root

    node = 0; depth = 0; %start at root
    done = 0; %decoder has finished or not
    while (done == 0) %traverse till all bits are decoded
        %leaf or not
        if depth == n
            if any(F==(node+1)) %is node frozen
                ucap(n+1,node+1) = 0;
            else
                if L(n+1,node+1) >= 0
                    ucap(n+1,node+1) = 0;
                else
                    ucap(n+1,node+1) = 1;
                end
            end
            if node == (N-1)
                done = 1;
            else
                node = floor(node/2); depth = depth - 1;
            end
        else
            %nonleaf
            npos = (2^depth-1) + node + 1; %position of node in node state vector
            if ns(npos) == 0 %step L and go to left child
                %disp('L')
                %disp([node depth])
                temp = 2^(n-depth);
                Ln = L(depth+1,temp*node+1:temp*(node+1)); %incoming beliefs
                a = Ln(1:temp/2); b = Ln(temp/2+1:end); %split beliefs into 2
                node = node *2; depth = depth + 1; %next node: left child
                temp = temp / 2; %incoming belief length for left child
                L(depth+1,temp*node+1:temp*(node+1)) = f(a,b); %minsum and storage
                ns(npos) = 1;
            else
                if ns(npos) == 1 %step R and go to right child
                    %disp('R')
                    %disp([node depth])
                    temp = 2^(n-depth);
                    Ln = L(depth+1,temp*node+1:temp*(node+1)); %incoming beliefs
                    a = Ln(1:temp/2); b = Ln(temp/2+1:end); %split beliefs into 2
                    lnode = 2*node; ldepth = depth + 1; %left child
                    ltemp = temp/2;
                    ucapn = ucap(ldepth+1,ltemp*lnode+1:ltemp*(lnode+1)); %incoming
decisions from left child
                    node = node *2 + 1; depth = depth + 1; %next node: right child
```

```
                    temp = temp / 2; %incoming belief length for right child
                    L(depth+1,temp*node+1:temp*(node+1)) = g(a,b,ucapn); %g and storage
                    ns(npos) = 2;
                else %step U and go to parent
                    temp = 2^(n-depth);
                    lnode = 2*node; rnode = 2*node + 1; cdepth = depth + 1; %left and
right child
                    ctemp = temp/2;
                    ucapl = ucap(cdepth+1,ctemp*lnode+1:ctemp*(lnode+1)); %incoming
decisions from left child
                    ucapr = ucap(cdepth+1,ctemp*rnode+1:ctemp*(rnode+1)); %incoming
decisions from right child
                    ucap(depth+1,temp*node+1:temp*(node+1)) = [mod(ucapl+ucapr,2)
ucapr]; %combine
                    node = floor(node/2); depth = depth - 1;
                end
            end
        end
    end

    msg_cap = ucap(n+1,Q1(N-K+1:end));

    %Counting errors
    Nerrs = sum(msg ~= msg_cap);
    if Nerrs > 0
        Nbiterrs = Nbiterrs + Nerrs;
        Nblkerrs = Nblkerrs + 1;
    end
end

BER_sim = Nbiterrs/K/Nblocks;
FER_sim = Nblkerrs/Nblocks;

timeend=toc(Tstart);

disp(num2str([EbNodB FER_sim timeend]))
```

Published with MATLAB® R2020b

7.5 nrpolar_sclistdecode_ FP.m

```
function [FER_sim,timeend]=nrpolar_sclistdecode_FP(x,r,N_len,list,num_blocks)

timestart=tic;
%Reliability sequence
Q=[0 1 2 4 8 16 32 3 5 64 9 6 17 10 18 128 12 33 65 20 256 34 24 36 7 129 66 512 11 40
68 130 ...
    19 13 48 14 72 257 21 132 35 258 26 513 80 37 25 22 136 260 264 38 514 96 67 41 144
28 69 42 ...
    516 49 74 272 160 520 288 528 192 544 70 44 131 81 50 73 15 320 133 52 23 134 384 76
137 82 56 27 ...
    97 39 259 84 138 145 261 29 43 98 515 88 140 30 146 71 262 265 161 576 45 100 640 51
148 46 75 266 273 517 104 162 ...
    53 193 152 77 164 768 268 274 518 54 83 57 521 112 135 78 289 194 85 276 522 58 168
139 99 86 60 280 89 290 529 524 ...
    196 141 101 147 176 142 530 321 31 200 90 545 292 322 532 263 149 102 105 304 296
163 92 47 267 385 546 324 208 386 150 153 ...
```

```
    165 106 55 328 536 577 548 113 154 79 269 108 578 224 166 519 552 195 270 641 523
275 580 291 59 169 560 114 277 156 87 197 ...
    116 170 61 531 525 642 281 278 526 177 293 388 91 584 769 198 172 120 201 336 62 282
143 103 178 294 93 644 202 592 323 392 ...
    297 770 107 180 151 209 284 648 94 204 298 400 608 352 325 533 155 210 305 547 300
109 184 534 537 115 167 225 326 306 772 157 ...
    656 329 110 117 212 171 776 330 226 549 538 387 308 216 416 271 279 158 337 550 672
118 332 579 540 389 173 121 553 199 784 179 ...
    228 338 312 704 390 174 554 581 393 283 122 448 353 561 203 63 340 394 527 582 556
181 295 285 232 124 205 182 643 562 286 585 ...
    299 354 211 401 185 396 344 586 645 593 535 240 206 95 327 564 800 402 356 307 301
417 213 568 832 588 186 646 404 227 896 594 ...
    418 302 649 771 360 539 111 331 214 309 188 449 217 408 609 596 551 650 229 159 420
310 541 773 610 657 333 119 600 339 218 368 ...
    652 230 391 313 450 542 334 233 555 774 175 123 658 612 341 777 220 314 424 395 673
583 355 287 183 234 125 557 660 616 342 316 ...
    241 778 563 345 452 397 403 207 674 558 785 432 357 187 236 664 624 587 780 705 126
242 565 398 346 456 358 405 303 569 244 595 ...
    189 566 676 361 706 589 215 786 647 348 419 406 464 680 801 362 590 409 570 788 597
572 219 311 708 598 601 651 421 792 802 611 ...
    602 410 231 688 653 248 369 190 364 654 659 335 480 315 221 370 613 422 425 451 614
543 235 412 343 372 775 317 222 426 453 237 ...
    559 833 804 712 834 661 808 779 617 604 433 720 816 836 347 897 243 662 454 318 675
618 898 781 376 428 665 736 567 840 625 238 ...
    359 457 399 787 591 678 434 677 349 245 458 666 620 363 127 191 782 407 436 626 571
465 681 246 707 350 599 668 790 460 249 682 ...
    573 411 803 789 709 365 440 628 689 374 423 466 793 250 371 481 574 413 603 366 468
655 900 805 615 684 710 429 794 252 373 605 ...
    848 690 713 632 482 806 427 904 414 223 663 692 835 619 472 455 796 809 714 721 837
716 864 810 606 912 722 696 377 435 817 319 ...
    621 812 484 430 838 667 488 239 378 459 622 627 437 380 818 461 496 669 679 724 841
629 351 467 438 737 251 462 442 441 469 247 ...
    683 842 738 899 670 783 849 820 728 928 791 367 901 630 685 844 633 711 253 691 824
902 686 740 850 375 444 470 483 415 485 905 ...
    795 473 634 744 852 960 865 693 797 906 715 807 474 636 694 254 717 575 913 798 811
379 697 431 607 489 866 723 486 908 718 813 ...
    476 856 839 725 698 914 752 868 819 814 439 929 490 623 671 739 916 463 843 381 497
930 821 726 961 872 492 631 729 700 443 741 ...
    845 920 382 822 851 730 498 880 742 445 471 635 932 687 903 825 500 846 745 826 732
446 962 936 475 853 867 637 907 487 695 746 ...
    828 753 854 857 504 799 255 964 909 719 477 915 638 748 944 869 491 699 754 858 478
968 383 910 815 976 870 917 727 493 873 701 ...
    931 756 860 499 731 823 922 874 918 502 933 743 760 881 494 702 921 501 876 847 992
447 733 827 934 882 937 963 747 505 855 924 ...
    734 829 965 938 884 506 749 945 966 755 859 940 830 911 871 639 888 479 946 750 969
508 861 757 970 919 875 862 758 948 977 923 ...
    972 761 877 952 495 703 935 978 883 762 503 925 878 735 993 885 939 994 980 926 764
941 967 886 831 947 507 889 984 751 942 996 ...
    971 890 509 949 973 1000 892 950 863 759 1008 510 979 953 763 974 954 879 981 982
927 995 765 956 887 985 997 986 943 891 998 766 ...
    511 988 1001 951 1002 893 975 894 1009 955 1004 1010 957 983 958 987 1012 999 1016
767 989 1003 990 1005 959 1011 1013 895 1006 1014 1017 1018 ...
    991 1020 1007 1015 1019 1021 1022 1023]+1;

%%%%%%%%    Variable    %%%%%%%%
N = N_len;
Rate = r;
nL = list; %list size
```

```
crcL = 11;
A = floor((N*Rate)-crcL); %A=N*R-crc
crcg = fliplr([1 1 1 0 0 0 1 0 0 0 0 1]); %CRC polynomial
K = A + crcL; %CRC length = crcL
EbNodB = x;
n = log2(N);
Nblocks = num_blocks;

rmax = 3; %max received value
maxqr = 31; %max integer received value

EbNo = 10^(EbNodB/10);
sigma = sqrt(1/(2*Rate*EbNo));

Q1 = Q(Q<=N); %reliability sequence for N

F = Q1(1:N-K); %Frozen positions: Q1(1:N-K)
%Message positions: Q1(N-K+1:end)

satx = @(x,th) min(max(x,-th),th); %saturate FP value
f = @(a,b) (1-2*(a<0)).*(1-2*(b<0)).*min(abs(a),abs(b)); %minsum
g = @(a,b,c) satx(b+(1-2*c).*a,maxqr); %g function

%Simulate
Nbiterrs = 0; Nblkerrs = 0;
for blk = 1:Nblocks
    msg = randi([0 1],1,A); %generate random K-bit message
    [quot,rem] = gfdeconv([zeros(1,crcL) fliplr(msg)],crcg);
    msgcrc = [msg fliplr([rem zeros(1,crcL-length(rem))])];

    u = zeros(1,N);

    u(Q1(N-K+1:end)) = msgcrc; %assign message bits

    %encoding
    m = 1; %number of bits combined
    for d = n-1:-1:0
        for i = 1:2*m:N
            a = u(i:i+m-1); %first part
            b = u(i+m:i+2*m-1); %second part
            u(i:i+2*m-1) = [mod(a+b,2) b]; %combining
        end
        m = m * 2;
    end
    cword = u;

    s = 1 - 2 * cword; %BPSK bit to symbol conversion
    r = s + sigma * randn(1,N); %AWGN channel I
    %quantization
    r = satx(r,rmax);
    rq = round(r/rmax*maxqr);

    %nL SC decoders
    LLR = zeros(nL,n+1,N); %beliefs in nL decoders
    ucap = zeros(nL,n+1,N); %decisions in nL decoders
    PML = Inf*ones(nL,1); %Path metrics
    PML(1) = 0;
    ns = zeros(1,2*N-1); %node state vector
```

```
    LLR(:,1,:) = repmat(rq,nL,1,1); %belief of root
    DML = zeros(nL,N);
    PMLL = zeros(nL,N);

    node = 0; depth = 0; %start at root
    done = 0; %decoder has finished or not
    while (done == 0) %traverse till all bits are decoded
        %leaf or not
        if depth == n
            DM = squeeze(LLR(:,n+1,node+1)); %decision metrics
            DML(:,node+1) = DM;
            PMLL(:,node+1) = PML;
            if any(F==(node+1)) %is node frozen
                ucap(:,n+1,node+1) = 0; %set all decisions to 0
                PML = PML + abs(DM).*(DM < 0); %if DM is negative, add |DM|
            else
                dec = DM < 0; %decisions as per DM
                PM2 = [PML; PML+abs(DM)];
                [PML, pos] = mink(PM2,nL); %In PM2(:), first nL are as per DM
                                           %next nL are opposite of DM
                pos1 = pos > nL; %surviving with opposite of DM: 1, if pos is above nL
                pos(pos1) = pos(pos1) - nL; %adjust index
                dec = dec(pos); %decision of survivors
                dec(pos1) = 1 - dec(pos1); %flip for opposite of DM
                LLR = LLR(pos,:,:); %rearrange the decoder states
                ucap = ucap(pos,:,:);
                ucap(:,n+1,node+1) = dec;
            end
            if node == (N-1)
                done = 1;
            else
                node = floor(node/2); depth = depth - 1;
            end
        else
            %nonleaf
            npos = (2^depth-1) + node + 1; %position of node in node state vector
            if ns(npos) == 0 %step L and go to left child
                %disp('L')
                %disp([node depth])
                temp = 2^(n-depth);
                Ln = squeeze(LLR(:,depth+1,temp*node+1:temp*(node+1))); %incoming
beliefs
                a = Ln(:,1:temp/2); b = Ln(:,temp/2+1:end); %split beliefs into 2
                node = node *2; depth = depth + 1; %next node: left child
                temp = temp / 2; %incoming belief length for left child
                LLR(:,depth+1,temp*node+1:temp*(node+1)) = f(a,b); %minsum and storage
                ns(npos) = 1;
            else
                if ns(npos) == 1 %step R and go to right child
                    %disp('R')
                    %disp([node depth])
                    temp = 2^(n-depth);
                    Ln = squeeze(LLR(:,depth+1,temp*node+1:temp*(node+1))); %incoming
beliefs
                    a = Ln(:,1:temp/2); b = Ln(:,temp/2+1:end); %split beliefs into 2
                    lnode = 2*node; ldepth = depth + 1; %left child
                    ltemp = temp/2;
```

```
                    ucapn =
squeeze(ucap(:,ldepth+1,ltemp*lnode+1:ltemp*(lnode+1))); %incoming decisions from left
child
                    node = node *2 + 1; depth = depth + 1; %next node: right child
                    temp = temp / 2; %incoming belief length for right child
                    LLR(:,depth+1,temp*node+1:temp*(node+1)) = g(a,b,ucapn); %g and
storage
                    ns(npos) = 2;
                else %step U and go to parent
                    temp = 2^(n-depth);
                    lnode = 2*node; rnode = 2*node + 1; cdepth = depth + 1; %left and
right child
                    ctemp = temp/2;
                    ucapl =
squeeze(ucap(:,cdepth+1,ctemp*lnode+1:ctemp*(lnode+1))); %incoming decisions from left
child
                    ucapr =
squeeze(ucap(:,cdepth+1,ctemp*rnode+1:ctemp*(rnode+1))); %incoming decisions from right
child
                    ucap(:,depth+1,temp*node+1:temp*(node+1)) = [mod(ucapl+ucapr,2)
ucapr]; %combine
                    node = floor(node/2); depth = depth - 1;
                end
            end
        end
    end
    %check CRC
    msg_capl = squeeze(ucap(:,n+1,Q1(N-K+1:end))); %get candidate messages
    cout = 1; %candidate codeword to be outputted, initially set to best PM
    for c1 = 1:nL
        [q1,r1] = gfdeconv(fliplr(msg_capl(c1,:)),crcg);
        if isequal(r1,0) %check if CRC passes
            cout = c1;
            break
        end
    end
    msg_cap = msg_capl(cout,1:A);

    %Counting errors
    Nerrs = sum(msg ~= msg_cap);
    if Nerrs > 0
        Nbiterrs = Nbiterrs + Nerrs;
        Nblkerrs = Nblkerrs + 1;
    end
end

BER_sim = Nbiterrs/A/Nblocks;
FER_sim = Nblkerrs/Nblocks;

timeend=toc(timestart);

disp(num2str([EbNodB FER_sim timeend]))
```

Published with MATLAB® R2020b

7.6 Turbo.m

```
function [bler,ber,timeend]=turbo(x,n,num_blocks)
timestart=tic;
%%%%%%%%    Variable    %%%%%%%%
EbNo = x;                % Eb/No values to loop over
blkLength = n;          % Block length
maxNumBlks = num_blocks;          % maximum number of blocks per Eb/No value


numIter = 3;               % Number of decoding iterations
trellis = poly2trellis([5 4],[23 35 0; 0 5 13]);
k = log2(trellis.numInputSymbols);      % number of input bits
n = log2(trellis.numOutputSymbols);     % number of output bits
intrIndices = randperm(blkLength/k)';   % Random interleaving
decAlg = 'True App';                    % Decoding algorithm
modOrder = 2;                           % PSK-modulation order BPSK

cEnc1 = comm.ConvolutionalEncoder('TrellisStructure',...
        trellis,'TerminationMethod','Truncated');
cEnc2 = comm.ConvolutionalEncoder('TrellisStructure',...
        trellis,'TerminationMethod','Truncated');
cAPPDec1 = comm.APPDecoder('TrellisStructure',trellis,...
            'TerminationMethod','Truncated','Algorithm',decAlg);
cAPPDec2 = comm.APPDecoder('TrellisStructure',trellis,...
            'TerminationMethod','Truncated','Algorithm',decAlg);

bpskMod = comm.BPSKModulator;
bpskDemod = comm.BPSKDemodulator('DecisionMethod','Log-likelihood ratio', ...
    'VarianceSource','Input port');

awgnChan = comm.AWGNChannel('NoiseMethod','Variance', ...
    'VarianceSource','Input port');

numferr=0;
bitError = comm.ErrorRate; % BER measurement
bitsPerSymbol = log2(modOrder);
turboEncRate =k/(2*n);

% Calculate the noise variance from EbNo
EsNo = EbNo+ 10*log10(bitsPerSymbol);
SNRdB = EsNo + 10*log10(turboEncRate); % Account for code rate
noiseVar = 10^(-SNRdB/10);

for  numBlks = 1:maxNumBlks
        % Generate binary data
        data = randi([0 1],blkLength,1);

        % Turbo encode the data
        [encodedData,outIndices] = helperTurboEnc(data,cEnc1,cEnc2, ...
            trellis,blkLength,intrIndices);

        % Modulate the encoded data
        modSignal = bpskMod(encodedData);

        % Pass the modulated signal through an AWGN channel
        receivedSignal = awgnChan(modSignal,noiseVar);
```

```
        % Demodulate the noisy signal using LLR to output soft bits
        demodSignal = bpskDemod(receivedSignal,noiseVar);

        % Turbo decode the demodulated data
        receivedBits = helperTurboDec(-demodSignal,cAPPDec1,cAPPDec2, ...
            trellis,blkLength,intrIndices,outIndices,numIter);

        % Calculate the error statistics
        errStates = bitError(data,receivedBits);
        numferr = numferr + any(receivedBits~=data);
end

ber= errStates(1);
bler = numferr/maxNumBlks;
timeend=toc(timestart);
disp(num2str([EbNo bler ber timeend]))
```

Published with MATLAB® R2020b

7.6.1 helpTurboEnc.m

```
function [yEnc,outIndices] =
helperTurboEnc(data,hCEnc1,hCEnc2,trellis,blkLength,intrIndices)
% Turbo encoding using two parallel convolutional encoders.
% No tail bits handling and assumes no output stream puncturing.

    % Trellis parameters
    k = log2(trellis.numInputSymbols);
    n = log2(trellis.numOutputSymbols);
    cLen = blkLength*n/k;

    punctrVec = [0;0;0;0;0;0];      % assumes all streams are output
    N = length(find(punctrVec==0));

    % Encode random data bits
    y1 = step(hCEnc1, data);
    y2 = step(hCEnc2, reshape(intrlv(reshape(data, k, [])',intrIndices)', [], 1));
    y1D = reshape(y1(1:cLen), n, []);
    y2D = reshape(y2(1:cLen), n, []);
    yDTemp = [y1D; y2D];
    y = yDTemp(:);

    % Generate output indices vector using puncturing vector
    idx = 0 : 2*n : (blkLength - 1)*2*(n/k);
    punctrVecIdx = find(punctrVec==0);
    dIdx = repmat(idx, N, 1) + punctrVecIdx;
    outIndices = dIdx(:);
    yEnc = y(outIndices);
end
```

Published with MATLAB® R2020b

7.6.2 helpTurboDec.m

```
function yDec =
helperTurboDec(yEnc,cAPPDec1,cAPPDec2,trellis,blkLength,intrIndices,inIndices,numIter)
% Turbo decoding using two a-posteriori probability (APP) decoders

    % Trellis parameters
    k = log2(trellis.numInputSymbols);
    n = log2(trellis.numOutputSymbols);
    rCodLen = 2*(n/k)*blkLength;
    typeyEnc = class(yEnc);

    % Re-order encoded bits according to outIndices
    x = zeros(rCodLen, 1);
    x(inIndices) = yEnc;

    % Generate output of first encoder
    yD = reshape(x(1:rCodLen), 2*n, []);
    lc1D = yD(1:n, :);
    Lc1_in = lc1D(:);

    % Generate output of second encoder
    lc2D   = yD(n+1:2*n, :);
    Lc2_in = lc2D(:);

    % Initialize unencoded data input
    Lu1_in = zeros(blkLength, 1, typeyEnc);

    % Turbo Decode
    out1 = zeros(blkLength/k, k, typeyEnc);
    for iterIdx = 1 : numIter
        [Lu1_out, ~] = step(cAPPDec1, Lu1_in, Lc1_in);
        tmp = Lu1_out(1:blkLength);
        Lu2_in = reshape(tmp, k, [])';
        [Lu2_out, ~] = step(cAPPDec2, ...
            reshape(Lu2_in(intrIndices, :)', [], 1), Lc2_in);
        out1(intrIndices, :) = reshape(Lu2_out(1:blkLength), k, [])';
        Lu1_in = reshape(out1', [], 1);
    end
    % Calculate llr and decoded bits for the final iteration
    llr = reshape(out1', [], 1) + Lu1_out(1:blkLength);
    yDec = cast((llr>=0), typeyEnc);
end
```

Published with MATLAB® R2020b

7.7 Sim_BLER.m

```
clc
clear

% Polar code
%SC decode
load data\NR_Polar_N128_SC_R12.mat Polar_N128_SC_R12;

%SCL decode
```

```
%N=128
load data\NR_Polar_N128_L8_R13.mat Polar_N128_L8_R13;
load data\NR_Polar_N128_L8_R12.mat Polar_N128_L8_R12;
load data\NR_Polar_N128_L8_R56.mat Polar_N128_L8_R56;

load data\NR_Polar_N128_L32_R12.mat Polar_N128_L32_R12;
%N=1024
load data\NR_Polar_N1024_L8_R13.mat Polar_N1024_L8_R13;
load data\NR_Polar_N1024_L8_R12.mat Polar_N1024_L8_R12;
load data\NR_Polar_N1024_L8_R56.mat Polar_N1024_L8_R56;

% LDPC code
%BG1
%N=128
load data\NR_LDPC_BG1_N128_R13.mat LDPC_BG1_N128_R13;
load data\NR_LDPC_BG1_N128_R12.mat LDPC_BG1_N128_R12;
load data\NR_LDPC_BG1_N128_R56.mat LDPC_BG1_N128_R56;
%N=1024
load data\NR_LDPC_BG1_N1024_R13.mat LDPC_BG1_N1024_R13;
load data\NR_LDPC_BG1_N1024_R12.mat LDPC_BG1_N1024_R12;
load data\NR_LDPC_BG1_N1024_R56.mat LDPC_BG1_N1024_R56;

%BG2
%N=128
load data\NR_LDPC_BG2_N128_R13.mat LDPC_BG2_N128_R13;
load data\NR_LDPC_BG2_N128_R12.mat LDPC_BG2_N128_R12;
load data\NR_LDPC_BG2_N128_R56.mat LDPC_BG2_N128_R56;
%N=1024
load data\NR_LDPC_BG2_N1024_R13.mat LDPC_BG2_N1024_R13;
load data\NR_LDPC_BG2_N1024_R12.mat LDPC_BG2_N1024_R12;
load data\NR_LDPC_BG2_N1024_R56.mat LDPC_BG2_N1024_R56;

% Turbo code
load data\LTE_Turbo_N128_R13.mat Turbo_N128_R13;
load data\LTE_Turbo_N1024_R13.mat Turbo_N1024_R13;
```

7.7.1 BLER-EbNodB total R=1/2

```
figure()
%polar
semilogy(Polar_N128_SC_R12(:,1),Polar_N128_SC_R12(:,2),'rs-');
hold on;
semilogy(Polar_N128_L8_R12(:,1),Polar_N128_L8_R12(:,2),'rs--');
hold on;
semilogy(Polar_N128_L32_R12(:,1),Polar_N128_L32_R12(:,2),'rs-.');
hold on;
semilogy(Polar_N1024_L8_R12(:,1),Polar_N1024_L8_R12(:,2),'r*--');
hold on;

%ldpc
semilogy(LDPC_BG1_N128_R12(:,1),LDPC_BG1_N128_R12(:,2),'bs-');
hold on;
semilogy(LDPC_BG1_N1024_R12(:,1),LDPC_BG1_N1024_R12(:,2),'b*-');
hold on;
semilogy(LDPC_BG2_N128_R12(:,1),LDPC_BG2_N128_R12(:,2),'bs--');
hold on;
semilogy(LDPC_BG2_N1024_R12(:,1),LDPC_BG2_N1024_R12(:,2),'b*--');
hold on;
```

```
grid on
grid minor
axis([-1 7 1e-4 1]);
xlabel('Eb/No (dB)','FontName','Times','FontSize',16);
ylabel('BLER','FontName','Times','FontSize',16);
title('AWAG, BPSK, Rate=1/2','FontName','Times','FontSize',14);
legend('Polar SC, N=128','Polar SCL, N=128, L=8, CRC=11',...
'Polar SCL, N=128, L=32,CRC=11','Polar SCL, N=1024, L=8, CRC=11',...
'LDPC BG1 N=128','LDPC BG1 N=1024','LDPC BG2 N=128','LDPC BG2 N=1024',...
'FontSize',18,'location','best');
```

7.7.2 BLER-EbNodB N=128 diferente Rate

```
figure()
%polar
semilogy(Polar_N128_L8_R13(:,1),Polar_N128_L8_R13(:,2),'ro-');
hold on;
semilogy(Polar_N128_L8_R12(:,1),Polar_N128_L8_R12(:,2),'r*-');
hold on;
semilogy(Polar_N128_L8_R56(:,1),Polar_N128_L8_R56(:,2),'rd-');
hold on;

%ldpc
semilogy(LDPC_BG1_N128_R13(:,1),LDPC_BG1_N128_R13(:,2),'bo--');
hold on;
semilogy(LDPC_BG1_N128_R12(:,1),LDPC_BG1_N128_R12(:,2),'b*--');
hold on;
semilogy(LDPC_BG1_N128_R56(:,1),LDPC_BG1_N128_R56(:,2),'bd--');
hold on;

semilogy(LDPC_BG2_N128_R13(:,1),LDPC_BG2_N128_R13(:,2),'bo-.');
hold on;
semilogy(LDPC_BG2_N128_R12(:,1),LDPC_BG2_N128_R12(:,2),'b*-.');
hold on;
semilogy(LDPC_BG2_N128_R56(:,1),LDPC_BG2_N128_R56(:,2),'bd-.');
hold on;

%turbo
semilogy(Turbo_N128_R13(:,1),Turbo_N128_R13(:,2),'go-');
hold on;

grid on
grid minor
axis([-1 7 1e-4 1]);
xlabel('Eb/No (dB)','FontName','Times','FontSize',16);
ylabel('BLER','FontName','Times','FontSize',16);
title('AWAG, BPSK, N=128','FontName','Times','FontSize',14);
legend('Polar, L=8, R=1/3','Polar, L=8, R=1/2','Polar, L=8, R=5/6',...
'LDPC BG1, R=1/3','LDPC BG1, R=1/2','LDPC BG1, R=5/6',...
'LDPC BG2, R=1/3','LDPC BG2, R=1/2','LDPC BG2, R=5/6',...
'LTE Turbo, R=1/3','FontSize',18,'location','best');
```

7.7.3 BLER-EbNodB N=1024

```
figure()
%polar
semilogy(Polar_N1024_L8_R13(:,1),Polar_N1024_L8_R13(:,2),'ro-');
hold on;
semilogy(Polar_N1024_L8_R12(:,1),Polar_N1024_L8_R12(:,2),'r*-');
```

```
hold on;
semilogy(Polar_N1024_L8_R56(:,1),Polar_N1024_L8_R56(:,2),'rd-');
hold on;

%ldpc
semilogy(LDPC_BG1_N1024_R13(:,1),LDPC_BG1_N1024_R13(:,2),'bo--');
hold on;
semilogy(LDPC_BG1_N1024_R12(:,1),LDPC_BG1_N1024_R12(:,2),'b*--');
hold on;
semilogy(LDPC_BG1_N1024_R56(:,1),LDPC_BG1_N1024_R56(:,2),'bd--');
hold on;

semilogy(LDPC_BG2_N1024_R13(:,1),LDPC_BG2_N1024_R13(:,2),'bo-.');
hold on;
semilogy(LDPC_BG2_N1024_R12(:,1),LDPC_BG2_N1024_R12(:,2),'b*-.');
hold on;
semilogy(LDPC_BG2_N1024_R56(:,1),LDPC_BG2_N1024_R56(:,2),'bd-.');
hold on;

%turbo
semilogy(Turbo_N1024_R13(:,1),Turbo_N1024_R13(:,2),'go-');
hold on;


grid on
grid minor
axis([-1 7 1e-4 1]);
xlabel('Eb/No (dB)','FontName','Times','FontSize',16);
ylabel('BLER','FontName','Times','FontSize',16);
title('AWAG, BPSK, N=1024','FontName','Times','FontSize',14);
legend('Polar, L=8, R=1/3','Polar, L=8, R=1/2','Polar, L=8, R=5/6',...
'LDPC BG1, R=1/3','LDPC BG1, R=1/2','LDPC BG1, R=5/6',...
'LDPC BG2, R=1/3','LDPC BG2, R=1/2','LDPC BG2, R=5/6',...
'LTE Turbo, R=1/3','FontSize',18,'location','best');
```

Published with MATLAB® R2020b

7.8 Sim_Time.m

```
clc
clear

% Polar code
%SC decode
load data\NR_Polar_N128_SC_R12.mat Polar_N128_SC_R12;

%SCL decode
%N=128
load data\NR_Polar_N128_L8_R13.mat Polar_N128_L8_R13;
load data\NR_Polar_N128_L8_R12.mat Polar_N128_L8_R12;
load data\NR_Polar_N128_L8_R56.mat Polar_N128_L8_R56;

load data\NR_Polar_N128_L32_R12.mat Polar_N128_L32_R12;
%N=1024
load data\NR_Polar_N1024_L8_R13.mat Polar_N1024_L8_R13;
load data\NR_Polar_N1024_L8_R12.mat Polar_N1024_L8_R12;
load data\NR_Polar_N1024_L8_R56.mat Polar_N1024_L8_R56;
```

```
% LDPC code
%BG1
%N=128
load data\NR_LDPC_BG1_N128_R13.mat LDPC_BG1_N128_R13;
load data\NR_LDPC_BG1_N128_R12.mat LDPC_BG1_N128_R12;
load data\NR_LDPC_BG1_N128_R56.mat LDPC_BG1_N128_R56;
%N=1024
load data\NR_LDPC_BG1_N1024_R13.mat LDPC_BG1_N1024_R13;
load data\NR_LDPC_BG1_N1024_R12.mat LDPC_BG1_N1024_R12;
load data\NR_LDPC_BG1_N1024_R56.mat LDPC_BG1_N1024_R56;

%BG2
%N=128
load data\NR_LDPC_BG2_N128_R13.mat LDPC_BG2_N128_R13;
load data\NR_LDPC_BG2_N128_R12.mat LDPC_BG2_N128_R12;
load data\NR_LDPC_BG2_N128_R56.mat LDPC_BG2_N128_R56;
%N=1024
load data\NR_LDPC_BG2_N1024_R13.mat LDPC_BG2_N1024_R13;
load data\NR_LDPC_BG2_N1024_R12.mat LDPC_BG2_N1024_R12;
load data\NR_LDPC_BG2_N1024_R56.mat LDPC_BG2_N1024_R56;

% Turbo code
load data\LTE_Turbo_N128_R13.mat Turbo_N128_R13;
load data\LTE_Turbo_N1024_R13.mat Turbo_N1024_R13;
```

7.8.1 Time R=1/3

```
figure()
%polar
semilogy(Polar_N128_L8_R13(:,1),Polar_N128_L8_R13(:,3),'rs-');
hold on;
semilogy(Polar_N1024_L8_R13(:,1),Polar_N1024_L8_R13(:,3),'r*-');
hold on;

%ldpc
semilogy(LDPC_BG1_N128_R13(:,1),LDPC_BG1_N128_R13(:,3),'bs--');
hold on;
semilogy(LDPC_BG1_N1024_R13(:,1),LDPC_BG1_N1024_R13(:,3),'b*--');
hold on;
semilogy(LDPC_BG2_N128_R13(:,1),LDPC_BG2_N128_R13(:,3),'bs-.');
hold on;
semilogy(LDPC_BG2_N1024_R13(:,1),LDPC_BG2_N1024_R13(:,3),'b*-.');
hold on;

%turbo
semilogy(Turbo_N128_R13(:,1),Turbo_N128_R13(:,3),'go-');
hold on;
semilogy(Turbo_N1024_R13(:,1),Turbo_N1024_R13(:,3),'g*-');
hold on;

grid on
grid minor
axis([-1 7 0 1e4]);
xlabel('Eb/No (dB)','FontName','Times','FontSize',16);
ylabel('Time (s)','FontName','Times','FontSize',16);
title('AWAG, BPSK, Rate=1/3','FontName','Times','FontSize',14);
legend('Polar SCL, N=128, L=8, CRC=11','Polar SCL, N=1024, L=8, CRC=11',...
'LDPC BG1, N=128','LDPC BG1, N=1024','LDPC BG2, N=128',...
```

```
'LDPC BG2, N=1024','LTE Turbo, N=128','LTE Turbo, N=1024',...
'FontSize',16,'location','best');
```

7.8.2 Time R=1/2

```
figure()
%polar
semilogy(Polar_N128_SC_R12(:,1),Polar_N128_SC_R12(:,3),'ro-');
hold on;
semilogy(Polar_N128_L8_R12(:,1),Polar_N128_L8_R12(:,3),'rs-');
hold on;
semilogy(Polar_N128_L32_R12(:,1),Polar_N128_L32_R12(:,3),'rp-');
hold on;
semilogy(Polar_N1024_L8_R12(:,1),Polar_N1024_L8_R12(:,3),'r*-');
hold on;

%ldpc
semilogy(LDPC_BG1_N128_R12(:,1),LDPC_BG1_N128_R12(:,3),'bs--');
hold on;
semilogy(LDPC_BG1_N1024_R12(:,1),LDPC_BG1_N1024_R12(:,3),'b*--');
hold on;
semilogy(LDPC_BG2_N128_R12(:,1),LDPC_BG2_N128_R12(:,3),'bs-.');
hold on;
semilogy(LDPC_BG2_N1024_R12(:,1),LDPC_BG2_N1024_R12(:,3),'b*-.');
hold on;

grid on
grid minor
axis([-1 7 0 1e4]);
xlabel('Eb/No (dB)','FontName','Times','FontSize',16);
ylabel('Time (s)','FontName','Times','FontSize',16);
title('AWAG, BPSK, Rate=1/2','FontName','Times','FontSize',14);
legend('Polar SC, N=128','Polar SCL, N=128, L=8, CRC=11',...
'Polar SCL, N=128, L=32,CRC=11','Polar SCL, N=1024, L=8, CRC=11',...
'LDPC BG1, N=128','LDPC BG1, N=1024','LDPC BG2, N=128','LDPC BG2, N=1024',...
'FontSize',12,'location','best');
```

7.8.3 Time R=5/6

```
figure()
%polar
semilogy(Polar_N128_L8_R56(:,1),Polar_N128_L8_R56(:,3),'rs-');
hold on;
semilogy(Polar_N1024_L8_R56(:,1),Polar_N1024_L8_R56(:,3),'r*-');
hold on;

%ldpc
semilogy(LDPC_BG1_N128_R56(:,1),LDPC_BG1_N128_R56(:,3),'bs--');
hold on;
semilogy(LDPC_BG1_N1024_R56(:,1),LDPC_BG1_N1024_R56(:,3),'b*--');
hold on;
semilogy(LDPC_BG2_N128_R56(:,1),LDPC_BG2_N128_R56(:,3),'bs-.');
hold on;
semilogy(LDPC_BG2_N1024_R56(:,1),LDPC_BG2_N1024_R56(:,3),'b*-.');
hold on;


grid on
grid minor
```

```
axis([-1 7 0 1e4]);
xlabel('Eb/No (dB)','FontName','Times','FontSize',16);
ylabel('Time (s)','FontName','Times','FontSize',16);
title('AWAG, BPSK, Rate=5/6','FontName','Times','FontSize',14);
legend('Polar SCL, N=128, L=8, CRC=11','Polar SCL, N=1024, L=8, CRC=11',...
'LDPC BG1 N=128','LDPC BG1 N=1024','LDPC BG2 N=128','LDPC BG2 N=1024',...
'FontSize',16,'location','best');
```

Published with MATLAB® R2020b

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR