CHANNEL CODING TECHNIQUES FOR 5G USING POLAR CODES

by

SUPREET HUILGOL

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

April 2017

Copyright © by Supreet Huilgol 2017

All Rights Reserved

ACKNOWLEDGEMENTS

In an endeavor to successfully complete my thesis, I received assistance from many people and I take this opportunity to thank those who have helped me along the way to achieve this success.

I would like to express my heartfelt gratitude and thank Dr. Qilian Liang for being my supervisor, mentor and inspiring source during my thesis. I would also like to thank Dr. Ioannis schizas and Dr. William Dillon for being in my thesis committee.

I would like to thank all my team members at U.S. Cellular, Chicago who provided me constant support during my Internship.

I would also like to thank our Wireless Communication and Networking Lab members for their support and inputs during my research.

Far too many people to mention individually have assisted me in so many ways during my research work especially I would like to thank Rahul for his support

Last but not the least, I would like to thank my parents Rajendra and Sudha Huilgol, my wife Priya and other family members for supporting me to pursue my masters.

18 Apr 2017

ABSTRACT

CHANNEL CODING TECHNIQUES FOR 5G USING POLAR CODES

Supreet Huilgol, M.S.

The University of Texas at Arlington

Supervising Professor: Dr. Qilian Liang

Coding and modulation in the crown known as the communications technology, embodies a national basic theory of the overall strength of communication science. Channel coding is a way of encoding data in a communication channel that adds patterns of redundancy into the transmission path in order to lower the error rate. Such methods are widely used in wireless communications.

5G is the coming fifth-generation wireless broadband technology based on the IEEE 802.11ac standard. 5G will provide better speeds and coverage than the current 4G. It operates with a 5 Ghz signal and is set to offer speeds of up to 1 Gb/s for tens of connections or tens of Mb/s for tens of thousands of connections. Commonly accepted use cases for 5G networks are eMBB (Enhanced Mobile Broadband), Massive IoT (Internet of Things) and URLLC (Ultra-Reliable and Low Latency Communications). eMBB covers Internet access with high data rates to enable rich media applications, cloud storage and applications, and augmented reality for entertainment.

All these demanding scenarios make use of many 5G standards of which polar codes is used as the channel coding scheme for eMBB scenario as short codes for control channel. A new class of codes, polar codes, recently made a breakthrough in coding theory. In 2008**,** Erdal Arıkan at Bilkent University invented polar codes, providing a new mathematical framework to solve this problem. The construction itself was first described by Stolte, and later independently by Erdal Arıkan in 2008

This thesis focuses on study of the key technology of polar code including the construction encoding and decoding. In this work, we analyze a method, known as channel polarization, to construct block codes that achieve the symmetric capacity of any binary-input discrete memoryless channel (B-DMC). The proof of their capacity achieving property is also given. In particular, we show that the algorithm can find almost all the “good” channels with computing complexity which is essentially linear in block-length. This thesis explores the structure and features of polar codes to improve their performance using Gaussian approximation-based construction of polar codes. Several schemes of polar codes are compared with each other like successive cancellation decoding(SC), list decoding(LS), list decoding with CRC(LS+CRC) and finally the existing adaptive decoder is shown to outperform all the schemes.

Table of Contents

List of Figures

List of Acronyms:

RAT – Radio Access Technology
LTE – Long Term Evolution
5G – 5th Generation
MIMO – Multiple Input Multiple Output
APP – A-Posteriori Probability
BCJR – Bahl Janelik Cockie Raviv
3GPP – 3rd Generation Partnership Project
ETSI – European Telecommunications Standard Institute
ITU-T– International Telecommunication Union
CRC – Cyclic Redundancy Check
SC–Successive Cancellation
SCL–Successive Cancellation List decoding
SCL-CRC–Cyclic Redundancy Check Aided Successive Cancellation List Decoding
LDPC – Low density Parity Check Codes
CTC– Cyclic Turbo Codes
ECC–Error Correcting
TDD–Time Division Duplexing
FDD–Frequency Division duplexing
URLLC –Ultra-Reliable and Low Latency Communications
CP-Cyclic Prefix
UPI-User Payload Identifier
MmW - Millimeter Wave
MIMO-Multi Input Multi Output
EMBB-Enhanced Mobile Broadband
URLCC-Ultra Reliable and Low Latency Communications
IoT-Internet of Things
B-DMC-Binary Discrete Memoryless Channel
ML-Maximum Likelihood
MAP-Maximum A Posteriori
CA-SCL-CRC Aided Successive Cancellation List Decoding
LR-Likelihood Ratio
BP-Belief Propagation
BER-Bit Error Rate
FER-Frame Error Rate
BLER-Block Error Rate

Chapter 1 – Introduction

1.1 5G Radio Access Technology

5G stands for 5th generation. To be more precise it can be said as fifth generation wireless systems or fifth generation mobile networks. It is the planned upcoming generation standard beyond the current LTE radio access technology standards. Qualcomm designed a chip in 2016[27] known as the Snapdragon X50 modem which supports signal frequencies for up to 28 GHz band, also known as millimeter wave (MmW) spectrum with 800 MHz bandwidth support. The peak data rates that would be supported by 5G technology are 5Gbps for downloads. Most of the RAN planning for the 5G network are currently ongoing and 5G planning aims at higher capacity than current 4G, allowing a sophisticated density of mobile broadband users, and supporting device-to-device, ultra-reliable, and massive machine communications[27][20]. Following are some of the standards [27] that are going to define the 5th generation mobile networks and they are as follows.

- Data rates of 100's of Mbps for big cities with large density of population
- 1 gigabit per second simultaneously to many workers on the same office floor
- Thousands of concurrent connections for multiple wireless sensors
- Compared to LTE signaling efficiency and Latency is significantly enhanced
- Signaling efficiency is reduced and geographical foot print or coverage is enhanced
- The Data rates of tens of megabits per second for tens of thousands of users

A typical 5G network architecture involves the following components and is shown in Figure1.

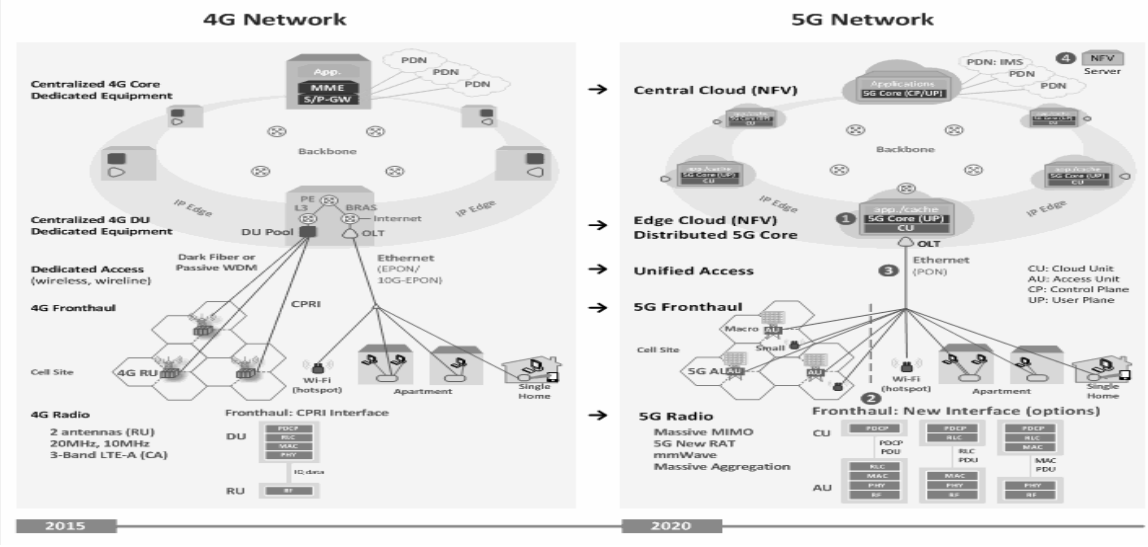


Figure 1 : Elements of 5G new RAT [27]

As seen in the figure 1 there is a clear transformation depicted from 4G to 5G. Instead of a centralized 4G core, there is a centralized cloud which consists of Applications for IP and 5G core(CP/UPI). Instead of a centralized 4G DU there is an edge cloud distributed

5G core. In 5G a unified access is present instead of a dedicated access as in 4G. The various components of 5G radio are massive MIMO, massive career aggregation, millimeter wave are shown in figure 2[28]

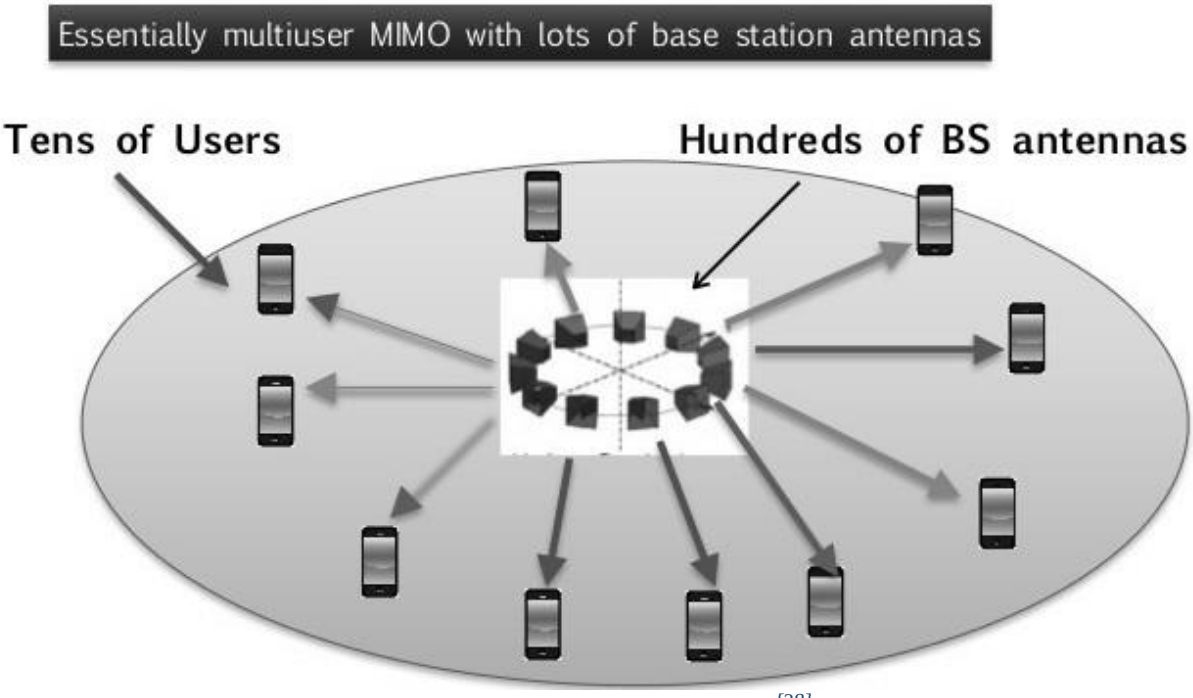


Figure 2 : Massive-MIMO elements for 5G [38]

Massive MIMO refers to the use of many service antennas operating coherently and adaptively, to significantly augment the bandwidth in mobile networks, as opposed to the conventional point-to-point MIMO. Massive MIMO plays a very important role in enhancing the 5G technology. As MIMO contains large number of antennas it focuses the reception and transmission of the energy of the signal into very smaller regions of space. This brings about enormous improvement in the throughput and energy efficiency. MIMO is said to clean break with the ongoing practice by using many service antennas that are operated adaptively and coherently particularly when combined. This is said to happen with simultaneous scheduling of many user terminals (e.g., tens or hundreds). Extra antennas help by focusing the transmission and reception of signal energy into ever-smaller regions of space. This brings huge improvements in throughput and energy efficiency, in particularly when combined with simultaneous scheduling of a large number of user terminals (e.g., tens or hundreds). Massive MIMO was initially intended for TDD operation mode, but can actually be useful also in FDD mode of operation.

1.2 Recent trends in 5G

Figure 3 below illustrates commonly accepted use cases for 5G networks: eMBB (Enhanced Mobile Broadband), Massive IoT (Internet of Things) and URLLC (Ultra-Reliable and Low Latency Communications) [29]. eMBB covers Internet access with high data rates to enable rich media applications, cloud storage and applications, and augmented reality for entertainment. Massive IoT applications include dense sensor networks for smart homes / buildings, remote health monitoring, and logistics tracking [39].

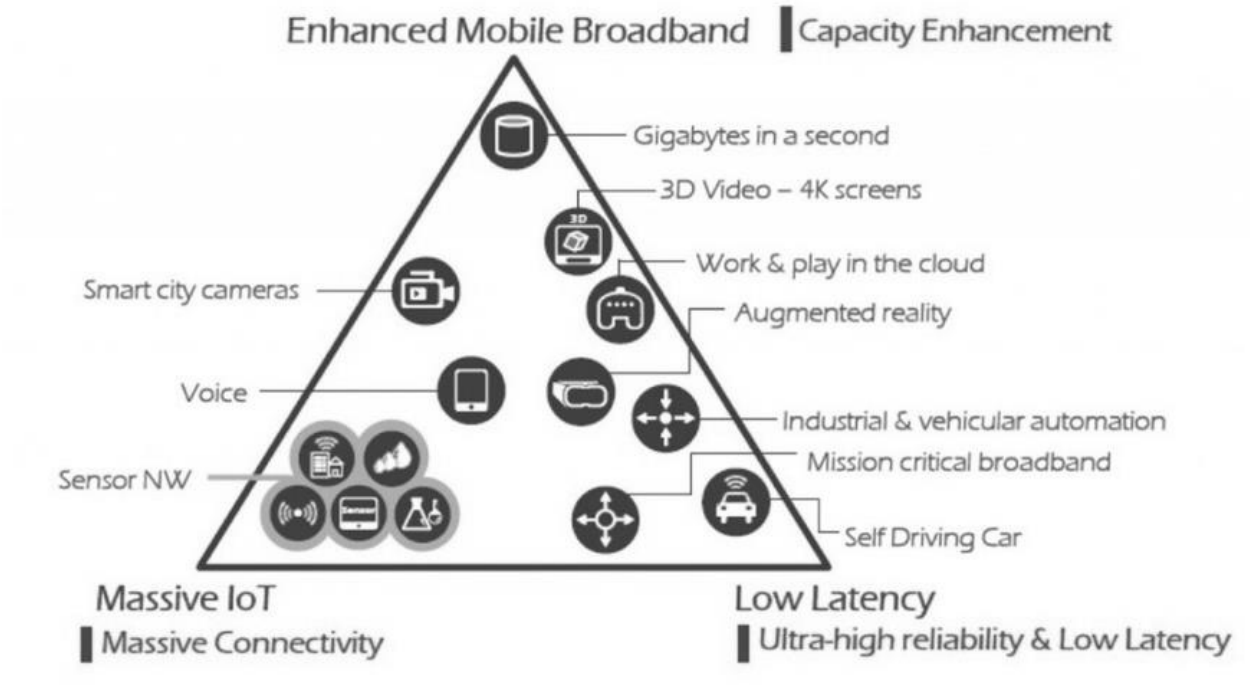


Figure 3: Commonly accepted use cases for 5G network [39]

URLLC covers critical applications that demand ultra-high reliability and low latency such as industrial automation, driverless cars, remote surgery, and smart grids. Recently the trend of self-driving cars has come in lime light by Uber and Google. 5G is one of the key elements in enabling the self-driving cars. It is also supported by IOT which is again a part of 5G technology.

1.3 Introduction to Polar codes

A new channel coding has flourished known as Polar coding and it is a channel coding scheme that was invented by Erdal Arıkan at Bilkent University (Ankara, Turkey) [14] [15]. Polar Codes are said to achieve channel capacity in a given binary discrete memoryless channel. This can be achieved only when the block size is large enough. The complexity of encoding and decoding is less and these codes can be successfully decoded. They have been used for testing and are eventually going to be deployed by Huawei in 5G networks by 2020[39].

A polarcode is said to be a linear block error correcting code [40]. Multiple concatenation that is recursive is said to be the basic building block for the polarcode and this is the basis for the code construction. Physical transformation of the channel takes place which transforms the physical channels to virtual channels and this transformation is based on multiple concatenation that is recursive. When the multiple channels multiply and accumulate, there is a chance that most of the channels either become good or bad and the idea behind polarcode is to make use of the good channels and the idea behind this to send the data through the good channels at rate 1 and send data through the bad channels at rate 0. In other words, the channels are said to enter the polarized state from the normal

state. The code construction was initially presented by stotle and later Erdal Arikan developed it[40][1]. It is the first code with an explicit construction to provably achieve the channel capacity for symmetric binary-input, discrete, memoryless channels (B-DMC) with polynomial dependence on the gap to capacity. One more important thing to be noted here is that polar codes the encoding and decoding complexity with modest construction which renders them to be used in a lot of applications.

Chapter 2 – Polar Codes for eMBB (Enhanced Mobile Broadband)

A new channel coding has come up known as Polar coding and it is a channel coding scheme that was invented by Erdal Arıkan at Bilkent University (Ankara, Turkey) [14][15]. Polar Codes are said to achieve channel capacity in a given binary discrete memoryless channel. This can be achieved only when the block size is large enough. The complexity of encoding and decoding is less and these codes can be successfully decoded. They have been used for testing and are eventually going to be deployed by Huawei in 5G networks by 2020[40]

As the demand for various accepted cases in 5G like eMBB, Massive IOT and URLCC increases, there is a need for stronger channel coding efficiency than Turbo Codes. Also, there is an increase in the capacity i.e. the maximum number of subscribers that a channel can sustain at a given point of time. So, Polar codes has been introduced to support increased channel capacity and improved bit error rate. SCL decoders with CRC achieve about 30% reduction in bit rate when compared to its predecessor Turbo Code at the same given conditions

2.1 Polar Codes Channel Model

The block diagram for Polar codes encoder is shown in figure 20[1]

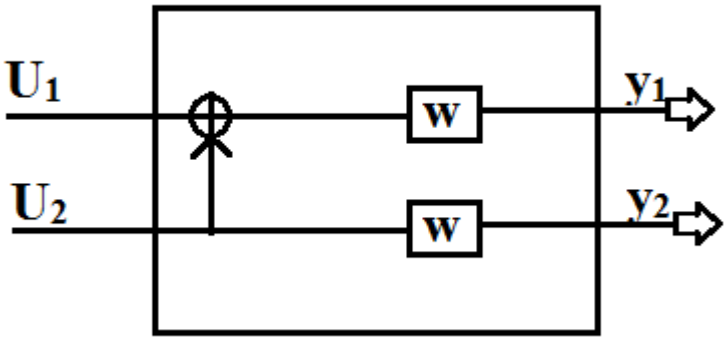


Figure 4: Polar code basic Channel w_2[1]

Before proceeding with the Polarcode we construct and specify a channel W_2[1] which is a Binary discrete memoryless channel(BDMC) and it is to achieve symmetric capacity I(W) which is the highest rate and this is subjected to using the I/P letters of the channel which are equally probable. It is possible to synthesize or create a second set of N binary input channels out of N independent copies of a given B-DMC Wand the channels have the properties $\{W_N^{(i)}: 1 <= i <= N\}$. One thing to be noted here is that when N becomes bigger, 2 things tend to happen i.e. some of the fraction of channels for the indices I for $I(W_N^{(i)})$ is near 0 approaches 1- I(w) and rest of the fraction of channels for indices I for $I(W^{(i)}_N)$ is near 1 approaches I(w).These polarized channels $\{W_N^{(i)}\}$ are in good condition for channel coding. So, one should make sure that the channels which are almost near 1 or are 1 have to be sent data through rate of 1 and some of the channels for which capacity is 0 need to send data at rate 0 i.e. the channels with capacity 0 are said to be junk channels. Therefore, Codes implemented on this

idea are called polar codes. We are trying to prove a fact that there always exists a sequence of polar codes {Cn; n -1} given any binary discrete memoryless channel with I(W) > 0 and any target rate R < I(W). The sequence of polar codes are such that Cn has block length N=2n and the bounding for the successive cancellation decoder is bounded as $Pe(N, R)<=O(N^{-1/4})$ which is said to be independent of the code rate for the probability of block error under successive cancellation decoding. The complexity of the decoders and encoders that achieve this performance are having a complexity of *O (N log N).*

For a Binary Discrete memoryless channel W, there are basically two parameters of interest. One is the symmetric capacity I(W) and the second one is the Bhattacharya parameter. Given a B-DMC W, there are two channel parameters of primary interest i.e. the symmetric capacity I(W) and the second one is the Bhattacharya parameter [1].

$$I(W)=\sum_{y \in Y} \sum_{x \in X} \tfrac{1}{2} w(y|x) \log \frac{w(y|x)}{\tfrac{1}{2}w(y|0)+\tfrac{1}{2}w(y|1)}$$

and the Bhattacharyya parameter is given by [1]

$$Z(w)=\sum_{y \in Y} \sqrt{w(y|0)w(y|1)}$$

The symmetric capacity parameter is used as a measure of rate and the Bhattacharya parameter is used as a parameter of reliability, respectively. Symmetric capacity is the highest rate at which any reliable communication will take place across the channel W. This will be done using the inputs for the channel W with equal frequency. Bhattacharya parameter is Z(W) is said to be an upper bound on the probability of maximum-likelihood (ML) decision error when W is used only once to transmit either of the two i.e. 0 or 1.

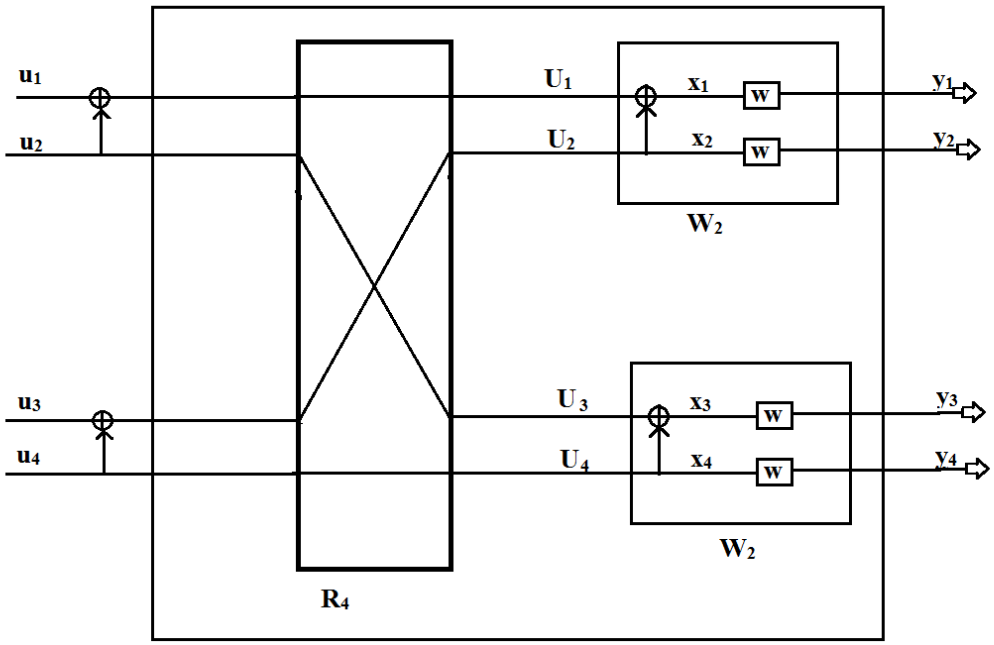


Figure 5: Polar code channel W_4 and its relationship with W and W_2[1]

We define a parameter known as Kronecker product and the Kronecker product of a matrix with dimension MXN for the parameters A and B is given by the following matrix

$$A \text{ X } B = \begin{bmatrix} A11B & \cdots & A1nB \\ \vdots & \ddots & \vdots \\ Am1B & \cdots & AmnB \end{bmatrix}$$

MR X NS is the matrix dimension for the above-mentioned matrix. The function definition of Kronecker power for all n≥1 is defined as $A^{\otimes n} = A \otimes A^{\otimes(n-1)}$. There is a convention to be followed and the convention is $A^{\otimes 0}=1$. The next convention to be followed is regarding 1_A and it is an indicator function for the set A. Therefore

$$1_A(x)=1 \{x \in A\}$$
$$=0\{\text{otherwise}\}$$

Three standard notations are used to denote the behavior of the functions using the Landau notation described and they are O(N), o(N), ω(N)[1][2]. We make use of certain notations to denote few of the realizations and few of the probability assignment. For example, the random variable X and Y will be denoting their realizations or sample values and the lower-case letters x and y will be used to specify their probability assignment. For example, PX specifies the probability assignment on X for X a RV. If we want to denote the joint probability we use P (X, Y). The mutual information and its conditional form will be denoted by the following i.e. I (X; Y), I (X; Y |Z) respectively.

The following are the standard notations that will be used throughout this section and in the upcoming sections

Standard Landau Notations O(N), o(N), ω(N)[1][2] denotes the asymptotic behavior of various functions

X, Y - Random variables

x, y - Realizations of random variables(RV's) of X and Y respectively

P_X - Probability assignment on a random variable X

$P_{X, Y}$ – Joint probability assignment for a join ensemble of RVs

I (X; Y), I(X; Y |Z) – denotes the mutual information and it's conditional form respectively

A row vector (a1, . . . , aN) will be denoted by a^N_1

$a^j_{i<=}$i, j<=N, 1 ≤i, j ≤ N, to denote the subvector (ai, . . . , aj); if j < i, aji is regarded as void.

$a^j_{1, e}$ can be used to denote the subvector with even indices (ak : 1 ≤k ≤ j; k even)

For example if we can consider the following a^5_1= (5,4,6,2,1), $a^5_{1,e}$=(4,2).

The notation 0^N_1 can be used to denote the all zero vector

2.1.1 Channel Polarization

Channel polarization can be defined as a process by which we use N independent copies of a given B-DMC W to create a second set of N channels $\{W_N^{(i)} : 1 \le i \le N\}$ that show a polarization effect meaning, when N becomes big, all the symmetric capacity terms $\{I(W_N^{(i)})\}$ tend towards 0 or 1 for all except a vanishing fraction of indices i. Channel polarization consists of two phases.

The theorem of polarization is stated below. It is more related to the concept of Source Polarization. As we can see from the Theorem below when the bit-channel capacities polarize the no of channels either tend to move towards 1 or 0. We are looking for fraction of terns for which conditional entropy lies between δ and 1- δ. For instance, let's take δ to be something like 1%. In this case, one would expect the interval between δ and 1- δ to be large because it occupies almost all the interval as N-Infinity, the terms between δ and 1- δ goes to 0 so there is almost nothing which means there are almost no channels in the middle, and all have shifted to either the lower end 0 or upper end 1[1].

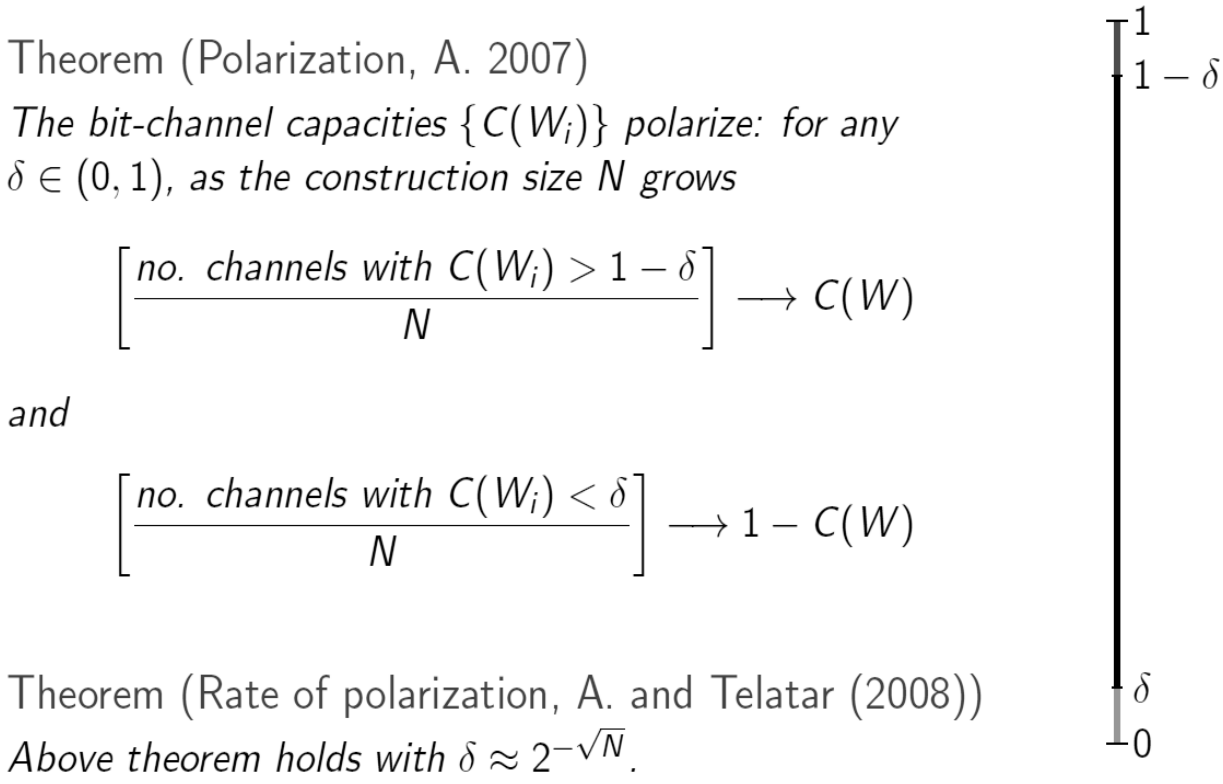


Figure 6: Channel polarization depicting 1- δ and δ [41]

Two phases can be described in the process of channel polarization
1.Channel combining phase
2.Channel splitting phase

2.1.1.1 Channel combining phase

Here in this phase we can combine together copies of a given Binary Discrete Memoryless Channel W in a recursive way to output vector channel given by the following equation WN: XN → YN N = 2^n which means that N is always a power of 2 and n is greater than or equal to 0. Recursion always will begin at the 0^{th} level and we set W_1=W with only 1 set of W. n=1 signifies the first level of the recursion where in 2 independent copies of W_1 are combines together which is shown in figure 4. After combining the two copies we obtain the channel W_2:X_2 →Y_2. The transitional probabilities of this new channel obtained would be represented by the following equation [1][2]

$$W_2(y_1, y_2|u_1, u_2) = W(y_1|u_1 \oplus u_2)W(y_2|u_2)$$

Once we obtain the channel we can again combine two copies of W_2 to obtain a single copy of the channel and now this channel would be called as W_4. This recursion would be represented by the following W4 : X4 → Y4 with transition probabilities[1][2]

$$W_4(y_1^4|u_1^4) = W_2(y_1^2|u_1 \oplus u_2, u_3 \oplus u_4)W_2(y_3^4|u_2, u_4)$$

In Fig. 6, we have got a mapping operation where mapping takes place between the input and output (s1, s2, s3, s4) to v^4_1 = (s1, s3, s2, s4) i.e. the mapping can be directly represented as U_{41} → X_{41} from the input of W_4 to the input of W^4 and the following matrix vector used to represent the G_4[1]

$$G_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

From this vector, we can derive a particular relation given by the equation $W_4(y^4_1|u^4_1)$ = $W_4(y_4{}^1|u^4_1\ G_4)$ and this relation is going to be between the input and output W_4 and W^4 and these two are nothing but the transitional probabilities. We can see the general form of recursion in figure 6. Here two copies of half a channel are combined to produce a single channel i.e. two copies of $W_{N/2}$ to produce a single copy of W_N. Transformation in the process takes step by step i.e. u^N_1 to W_N into s^N_1 i.e. u^N_1 is initially transformed to W_N and then into s^N_1 where such that for the following condition being satisfied $1 \leq i \leq N/2$ the equation holds true i.e. s_{2i-1} = $u_{2i-1} \oplus u_{2i}$ and $s_{2i} = u_{2i}$. There is a special name according to the function it performs for R_N i.e. it is known also as the reverse shuffle operator since it also does reverse shuffling for the channel space that it represents and this reverse shuffle operator produces the O/P v^N_1 = (s1, s3, . . . , s_{N-1}, s_2, s_4, . . . , s_N) by working on the I/P s^N_1.Once this is done this starts acting as an I/P to the 2 copies of the channel of WN/2 as shown in the figure 6.The mapping can be seen neatly from u^N_1→ v^N_1 and this mapping is said to be linear i.e. over the form of GF(2). Mostly the other mapping can also be told as linear which is the mapping u^N_1→ x^N_1 and this is the overall mapping from end to end. It can be told as being linear because all the other channel mappings that lie under this linear mapping are also linear and therefore this form can be denoted using a matrix G_N which is again a linear operation such that the following [1] holds true

$$x^N_1 = u^N_1 G_N$$

G_N is nothing but the generator matrix and the size of the generator matrix is N. The 2 channels W_N and W^N are having a transitional probabilities relation which can be represented using the following [1] equation

$$W_N(y_1^N|u_1^N) = W^N(y_1^N|u_1^N G_N)$$

$y^N_1 \in Y^N$, $u^N_1 \in X^N$ is the condition to be satisfied by the relation above and this holds true for all of the variables that are getting satisfied by the above equation. There is one more relation that needs to be mentioned regarding the G_N matrix and it is given by the following equation where B_N is the permutation matrix and F = [1 0 1 1]

$$G_N = B_N F^{\otimes n} \{N = 2^n, n \geq 0\}$$ [1]

F is the variable that is going to completely satisfy the channel combining operation.

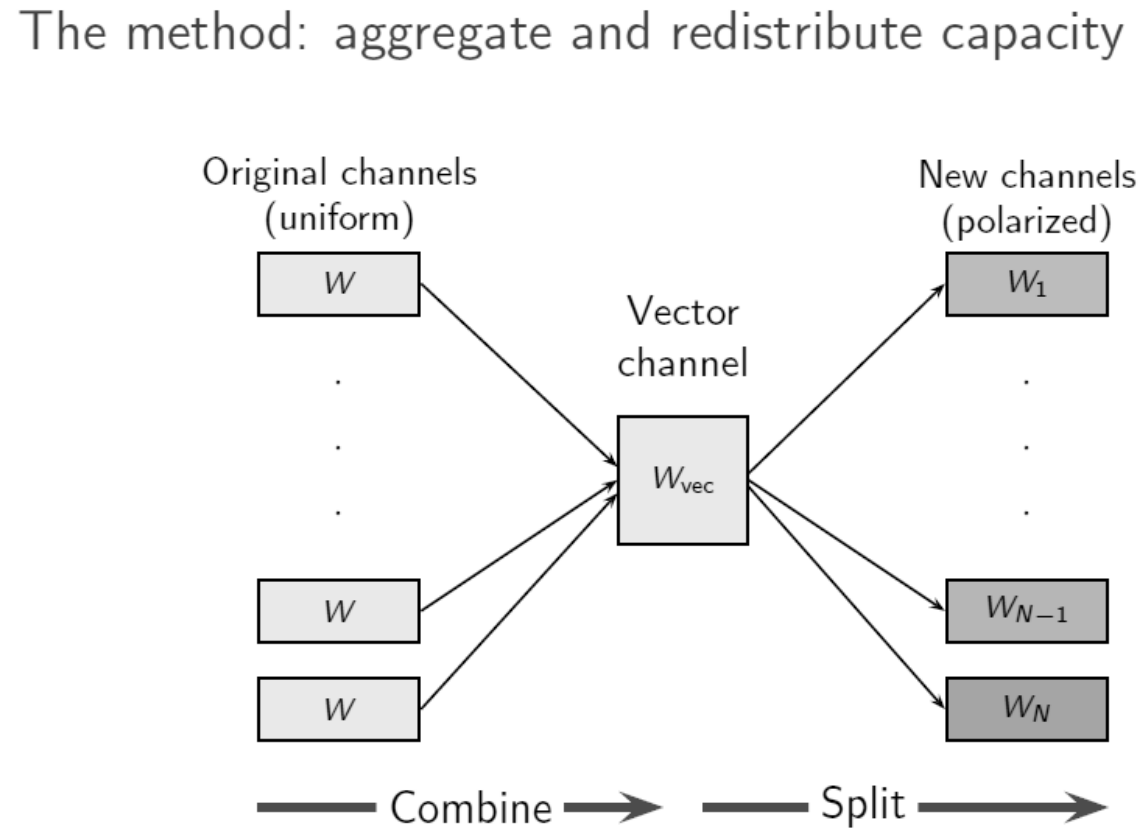


Figure 7: Channel Polarization-Channel combining and splitting [41]

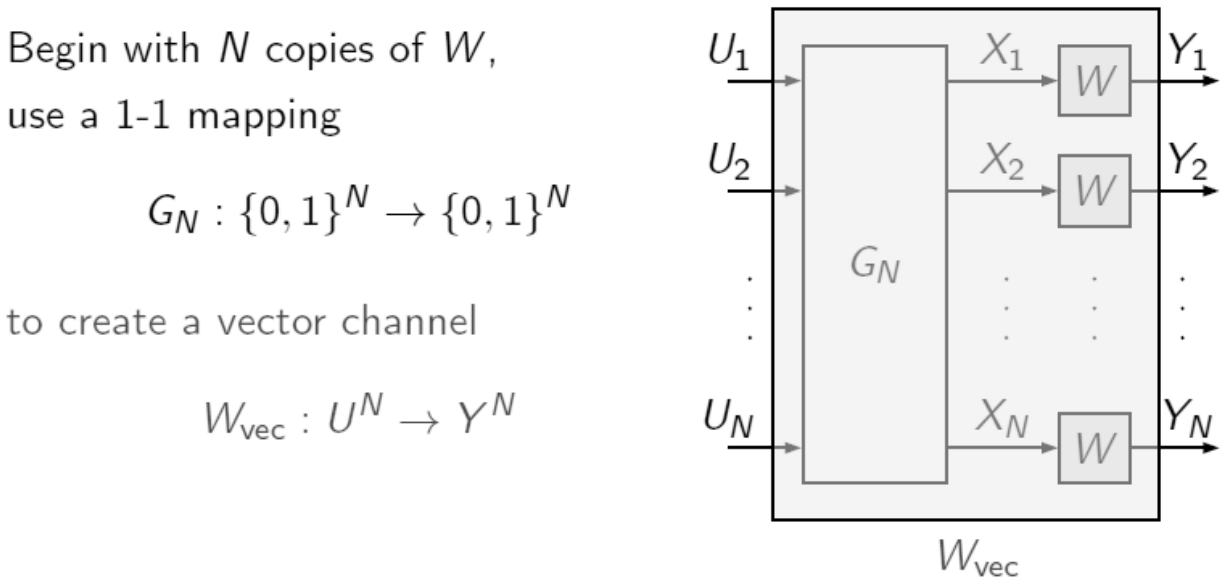


Figure 8: Channel Polarization vector matrix G_N [41]

2.1.1.2 Channel Splitting Phase

In the previous sections, we have manufactured W_N from the vector channel W^N, and now we will again do the reverse operation of combining i.e. we will be splitting the channel

W_N back into N binary input channels $W^{(i)}_N$: $X \rightarrow Y^N \times X^{i-1}$, $1 \leq i \leq N$ and the equations involving the splitting phase are defined by the transitional probabilities [1]

$$W_N^{(i)}(y_1^N, u_1^{i-1}|u_i) \triangleq \sum_{u_{i+1}^N \in \mathcal{X}^{N-i}} \frac{1}{2^{N-1}} W_N(y_1^N|u_1^N)$$

Here (y^N_1 , u^{i-1}_1) signifies the O/P of $W^{(i)}N$ and u_i its I/P.

We will get to know more information on this when we try to consider a genie aided decoder for the successive cancellation decoding where the i^{th} DE after observing the y^N_1 and the previous inputs u^{i-1}_1 and this is said to be supplied by the aided genie irrespective of any decision errors at the previous stages. Here we can try to estimate $W^{(i)}_N$ as being the effective channel seen by the i_{th} decision element If u^N1 is a-priori uniform on X_N.

2.2 Encoding Complexity

Complexity estimation of any encoding decoding system becomes a challenge when the complexity starts exponentially for any channel coding scheme. We can consider that our system runs on a single processor system with a RAM. Any complexities expressed here after will be the time complexities i.e. the complexities will increase/decrease with respect to time. For this purpose, we will consider a random G_N-coset code which has the following parameters given as

(*N,K,A, uAc*).

When we try to estimate the complexity we also need to take into account a parameter known as the worst-case encoding complexity. This will try to tell the worst scenario where the complexity is at its peak. Let us consider a block length of N and let the worst case encoding complexity be denoted by $\chi_E(N)$over all the parameters of (N, K, A, uAc) codes. We know the complexity of the scalar modulo addition is one unit and the complexity of the reverse shuffle operation R_N as N units. We can start deriving the encoding complexity operation as we see from figure 5 the equation for $\chi_E(N)$ is given by $\chi_E(N) \leq N/2+N+2\chi_E(N/2)$. If we consider an initial value of $\chi_E(2) = 3$, we obtain by induction that $\chi_E(N) \leq 3/2N \log N$ for all $N = 2n$, $n \geq 1$. Therefore, we can arrive to a conclusion that the encoding complexity of the polarcode encoder is *O (N log N).*

The implementation of the encoder has been shown in figure 9. We consider N=8 and this is very specific of the form $G_N = B_N F^{\otimes n}$ [1] i.e. the encoder implementation is specific to only N=8.We consider a bit reversed version as the channel is recursive and the I/P given the circuit are a bit reversed version of the u^8_1 i.e. u_1^8, i.e., $\tilde{u}_1^8 = u_1^8 B_8$. The O/P of the encoder is given by the equation which is of the form x^8_1= ˜u^8_1 $F^{\otimes 3}$ = u^8_1 G_8. From these equations, we can clearly see two things with respect to the complexity. One is the complexity of B_N is O(N) and the second one is that the complexity of the Kronecker product $F^{\otimes n}$ is *O (N log N).* Therefore, basically the overall complexity is *O (N log N).*

We can also have A substitute for the current implementation of the encoder. Applying u^8_1 u^8_1 in natural index order without bit reversing at the input of the circuit. If we carried out such an operation we would obtain a natural O/P rather than the bit reversed O/P and then the O/P would be given by the equation ˜x^8_1= u^8_1 $F^{\otimes 3}$ at the output. Later the encoding can be accomplished by a bit-reversal operation after ˜x^8_1= u^8_1 $F^{\otimes 3}$ i.e. post reverse bit operation i.e.:

$u^8_1 = \tilde{} \; x^8_1 B_8 = u^8_1 \; G_8$[1][3][6]. From this we can get to know that there can exist many parallel implementation substitutes for the encoding circuit of Figure 9 for the Kronecker product $F^{\otimes n}$. Once we can derive all the other parallel implementation alternatives we can also get to know that there can be tradeoffs between all these implementations with respect to the Latency and the complexity.

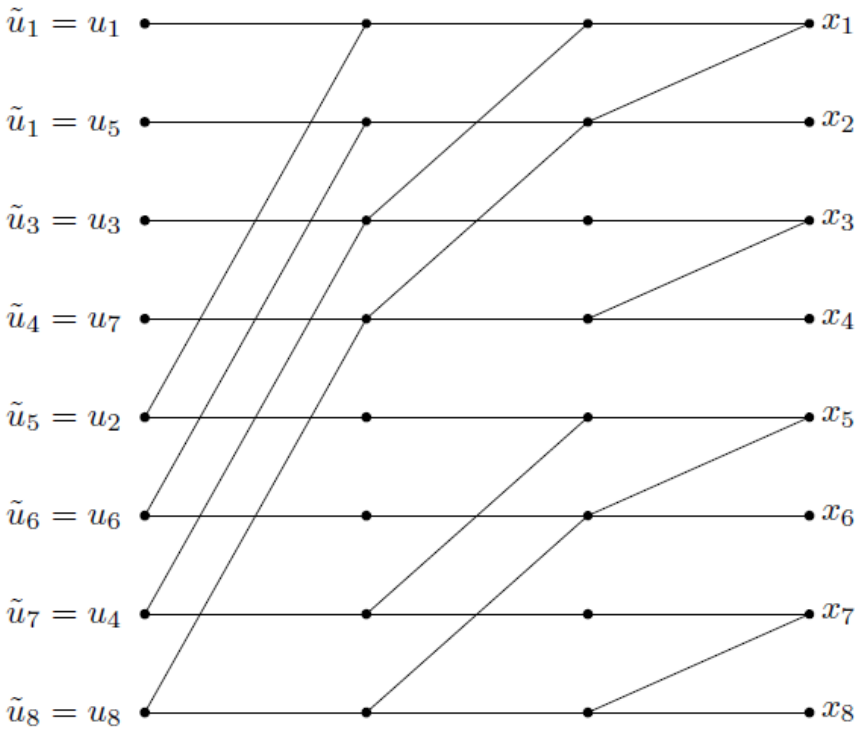


Figure 9: implementation of the transformation F^{x3}. All signal flow is from left to right. [1]

As we can see from figure 9 Each edge carries a signal 0 or 1. Each node adds (mod-2) the signals on all incoming edges from the left and sends the result out on all the right edges. (Edges carrying the signals u_i and x_i are not shown). In actual implementation of polar codes, it may be preferable to use F^{Xn} in place of B_n F^{Xn} as the encoder mapping in order to simplify the implementation. In that case, the SC decoder should compensate for this by decoding the elements of the source vector U_1^N in the bit reversed index order. We have included B_N as part of the encoder in this paper in order to have a SC decoder that decodes U_1^N in the natural index order, which simplified the notation

2.3 Systematic Polar Codes

As the Coding theory says almost every linear block code can be converted or can be transformed into an equivalent systematic code. Systematic encoding is not new. In the recent past almost all the codes like CTC, Turbo codes have been shown to have a superior performance with respect to BER BLER and FER performance. Systematic Polar Coding has been introduced by Arikan[3].Systematic Coding is nothing but a scheme where the input bits are embedded in the encoded output. Unlike the Non-Systematic Scheme in the Systematic

encoding the information bits appear as a part of the codeword transparently i.e there is no coding applied for these bits and they can be directly observed by the decoder. By doing so,

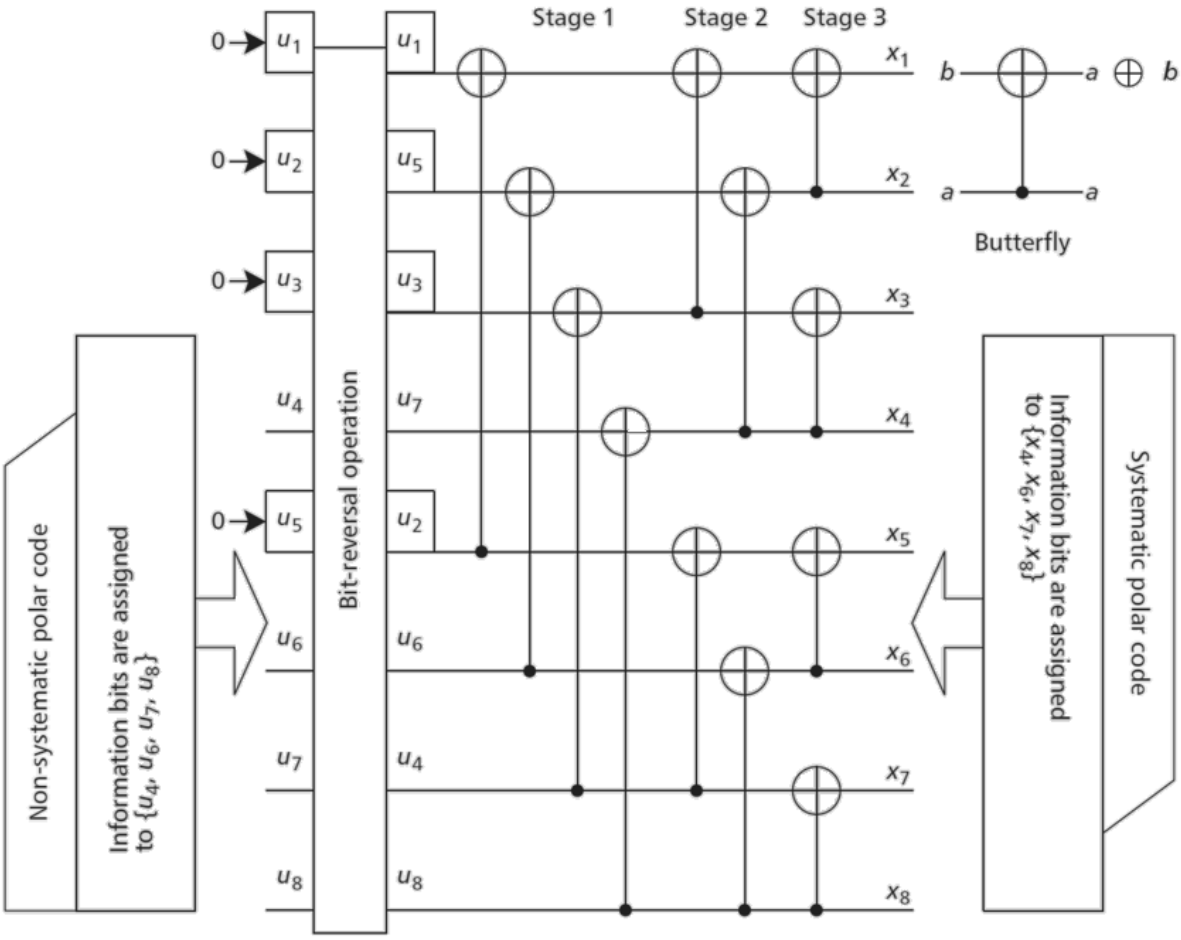


Figure 10 Systematic Polar Codes[42]

though the BLER is said to be same, an improvement in the performance of BER can seen. One of the most important advantage of systematic Polar Coding is that they are more robust against error propagation under SC decoding. As discussed earlier Compared with the original (non-systematic) polar code, the systematic polar code has the same block error rate (BLER), but is superior to the former in bit error rate (BER) performance. A block diagram of the systematic Polar Codes is given in figure 10.

$$x = U_A G_A + U_A^c \; G_A^c$$

$$x_B = U_A G_{AB} + U_A^c \, G_A^c{}_B$$

$$x_B^c = U_A G_{AB}^c + U_A^c \; G_A^c{}_B^c$$

$$U_A = x_B - (U_A^c \, G_A^c{}_B) \, (G_{AB})^{-1}$$

Where $U = (U_A, \; U_A^c)$ for some $A \in (1,\ldots,N)$, U_A consists of user data and U_A^C Consists of frozen bits. $x=uG$, $x, u \in F^N$, $G \in F^{N \times N}$. GA and G_A^c are the submatrices of G consisting of rows with indices A and A^c. G^{AB} denotes the submatrix of G consisting of the array of elements $(G_{i,j})$ with $i \in A$ and $j \in B$. More precisely for any nonsystematic encoder parameter

(A,U_A^c) we say that a systematic encoder (B, U_A^c) exists if the above equations for X_B and X_B^c hold true. At last we find out U_A and substitute it in $X_B^{C.}$

Chapter 3 – Architecture of different polar decoders

3.1 Successive cancellation decoder

Since introduced firstly in 2009, polar coding usually comes with SC decoder as a suboptimal decoding method. Nowadays, even though there are some decoding methods performing better, SC decoding still plays an integral role in polar coding. Just as the name implies, a SC decoder means decoding the bits in order from $u1$ to u_N. This also helps to realize channel polarization. As we have stated before, the mutual information $I(u_i; y^N_1 , u^{i-1}_1)$ or Bhattacharyya parameter $Z(u_i; y^N_1 , u^{i-1}_1)$ is required to predict the channel performance. Then the decoder should know the knowledge of u^{i-1}_1 when decoding u_i. Generally, the decoder only knows the values of frozen bits $\{u_j , j \in Ac\}$ in advance. As the SC decoder decode bits consecutively, it at least provides an estimate of u^{i-1}_1 when decoding u_i. Arikan has shown that if the information set A is properly
specified, this has no impact on the error performance if the code length is large enough.

Consider a polar code with the parameter vector (N,K,A, uAc). The decoder needs to retrieve the information and generate an estimate $\hat{u}^N_1$ of u^N_1 with the knowledge of y^N1, A, and uAc . The likelihood ratio (LR) is defined as [1][4]

$$L_N^{(i)}(y_1^N, u_1^{i-1}) = \frac{W_N^{(i)}(y_1^N, u_1^{i-1}|u_i = 0)}{W_N^{(i)}(y_1^N, u_1^{i-1}|u_i = 1)}$$

For convenience, the frozen bits are always set to 0. Then a SC decoder would take the following steps to achieve translating the source bits. For each i from 1 to N:

• if $i \in Ac$, $\hat{u}_i = u_i$.

• if $i \in A$, calculate the LR $L^{(i)}_N (y^N_1, u^{i-1}_1)$ and make the decision as

$$\hat{u}_i = \begin{cases} 0, & if \quad L_N^{(i)}(y_1^N, u_1^{i-1}) \geq 1 \\ 1, & otherwise \end{cases}$$

The LR can be recursively calculated as [1]

$$L_N^{(2i-1)}(y_1^N, \hat{u}_1^{2i-2}) = \frac{L_{N/2}^{(i)}(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2}) \cdot L_{N/2}^{(i)}(y_{N/2+1}^N, \hat{u}_{1,e}^{2i-2}) + 1}{L_{N/2}^{(i)}(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2}) + L_{N/2}^{(i)}(y_{N/2+1}^N, \hat{u}_{1,e}^{2i-2})}$$

$$L_N^{(2i)}(y_1^N, \hat{u}_1^{2i-1}) = [L_{N/2}^{(i)}(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2})]^{1-2\hat{u}_{2i-1}} \cdot L_{N/2}^{(i)}(y_{N/2+1}^N, \hat{u}_{1,e}^{2i-2})$$

The recursive calculation of LRs can be traced back to code length 1 with the LR $L^{(1)}_1 (y_i) = W(y_i|0)/W(y_i|1)$. $L^{(1)}_1 (y_i)$ is the soft information observed from the channel.

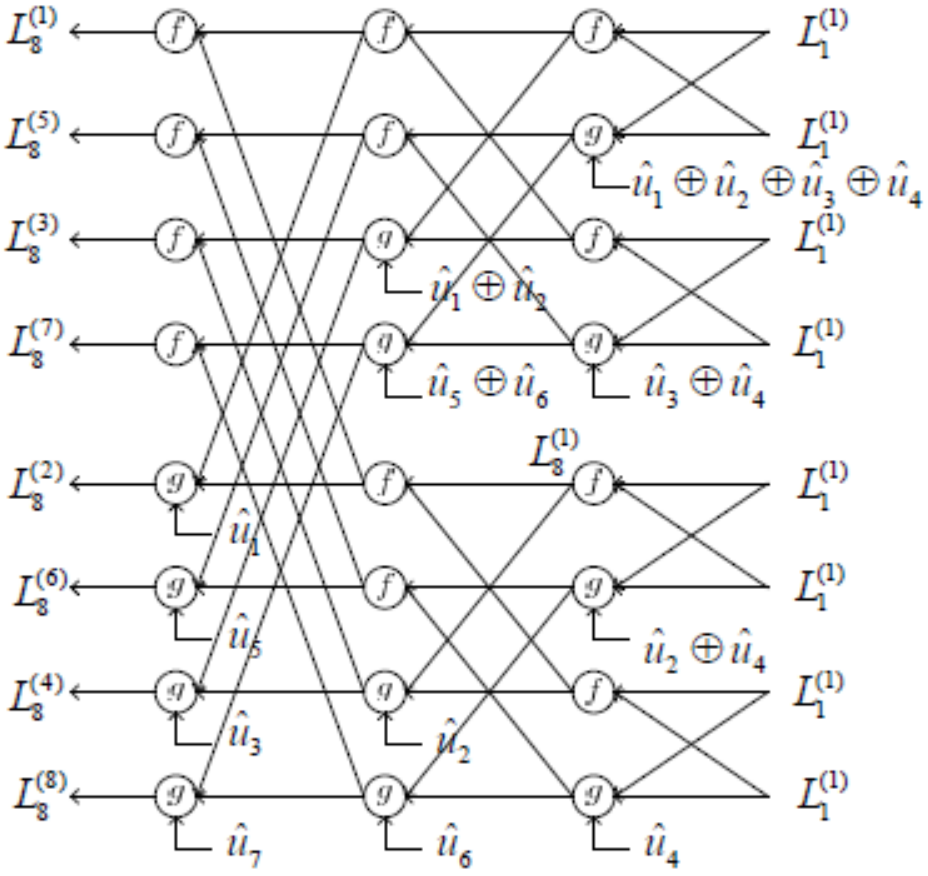


Figure 11: N=8 SC polar decoder architecture[43]

Figure 10 shows the implementation of an SC decoder with code length N=8. The calculation of $L^{(i)}_N$ is conducted from right to left. In this figure, f and g represent two functions as follows

$$\begin{cases} f(a,b) = \frac{1+ab}{a+b} \\ g(a,b,\hat{u}_s) = a^{1-2\hat{u}_s}b \end{cases}$$

The variables a and b denote $L^{(i)}_{N/2}(y^{N/2}_1, \hat{u}^{2i-2}_{1,0} \oplus \hat{u}2i-21,e)$ and $L^{(i)}_{N/2}(y^{N}_{N/2+1}, \hat{u}+^{2i-2}_{1,e})$, respectively.

3.1.1 Successive Cancellation Decoding Complexity

Now let us consider the SC decoding complexity. For traditional SC decoding, the LRs of frozen bits do not need to be computed. To analyze the decoding complexity in a simple manner, we consider computing the full set of $\{L^{(i)}_N(y^N_1, \hat{u}^{i-1}_1), i = 1, ..., N\}$.When making a decision, the frozen bits $\{\hat{u}_i, i \in Ac\}$ are set to the values which have been published to the decoder regardless of $L(i)N(yN1, \hat{u}i-11)$. The process of calculating $L^{(i)}_N(y^N_1, \hat{u}^{i-1}_1)$ is similar to calculating $W^{(i)}_N(y^N_1, \hat{u}^{i-1}_1)$ shown in figure 28

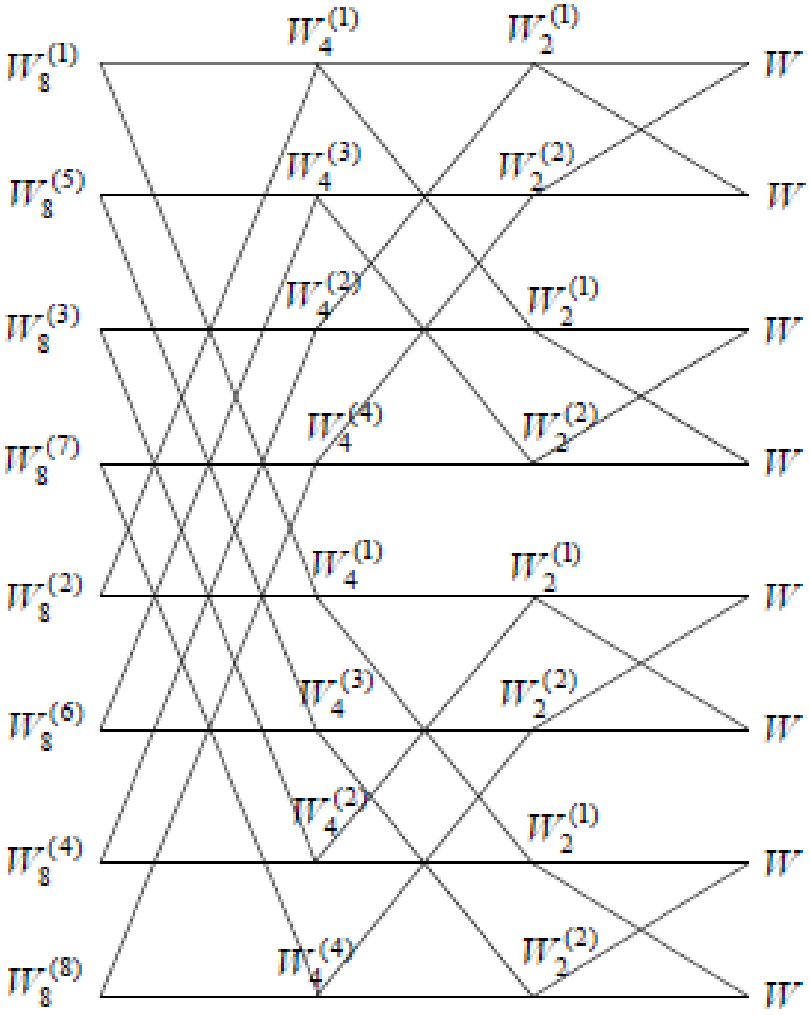


Figure 12: N=8 recursive transformation of channel [1]

3.1.2 SC Decoding Steps

The steps of SC decoding algorithm are explained as follows:

- *Step (1)* The rightmost side variables in figure11 are initialized with channel observations. For $i = 1, 2, ..., N$, $L(1)1\ (yi) = W(yi|ui=0)W(yi|ui=1)$.
- *Step (2)* The first bit $u1$ is needed to be decoded in step (2). If $u1$ is a frozen bit, $u1$ is set to 0. Otherwise, update the likelihood ratios from the rightmost side to the left most side according to the rules of calculating f and g and obtain the required $L(1)N$ .If $L(1)N > 1$, $u1$ is determined to 0. If $L(1)N < 1$, $u1$ is determined to 1. If $L(1)\ N = 1$, $u1$ is determined to 0 or 1 with equal probability.
- *Step (3)* In this step, $u2$ is decoded. The process is like step (2) except that when calculating $L(2)N$, the value $\hat{u}1$ is used when computing g functions. $u2$ is determined when $L(2)N$ is obtained finally.
- *Step (4)* Next, the remaining bits $u3,...,uN$ are decoded in the order. Decoding each bit would make use of the knowledge of previous bits.

Now we give an example to explain the SC decoding process.

3.1.3 An example on Successive Cancellation Decoding

An Example: SC decoding. Polar codes with code length N = 8, code rate R = 1/3 are transmitted under AWGN channel (Eb/N0=1dB) using SC decoding. The information set is A = {4, 6, 8}. The values of information bits are all one. The frozen bits u_1, u_2, u_3, u_5, u_7 are known as 0. The calculations of the likelihood ratios for the information bits are shown in the following figure. Note that we set $L^{(i)}_N(y_1^N, u_1^{i-1})$ as in the following equation. Therefore, u_i is decided to be 0 if $L^{(i)}_N$ is larger than 1 and vice versa. The steps of calculating the likelihood ratio for the fourth bit u4 is described as follows and we show it in figure13

$$L_N^{(i)}(y_1^N, u_1^{i-1}) = \frac{W_N^{(i)}(y_1^N, u_1^{i-1}|0)}{W_N^{(i)}(y_1^N, u_1^{i-1}|1)}$$

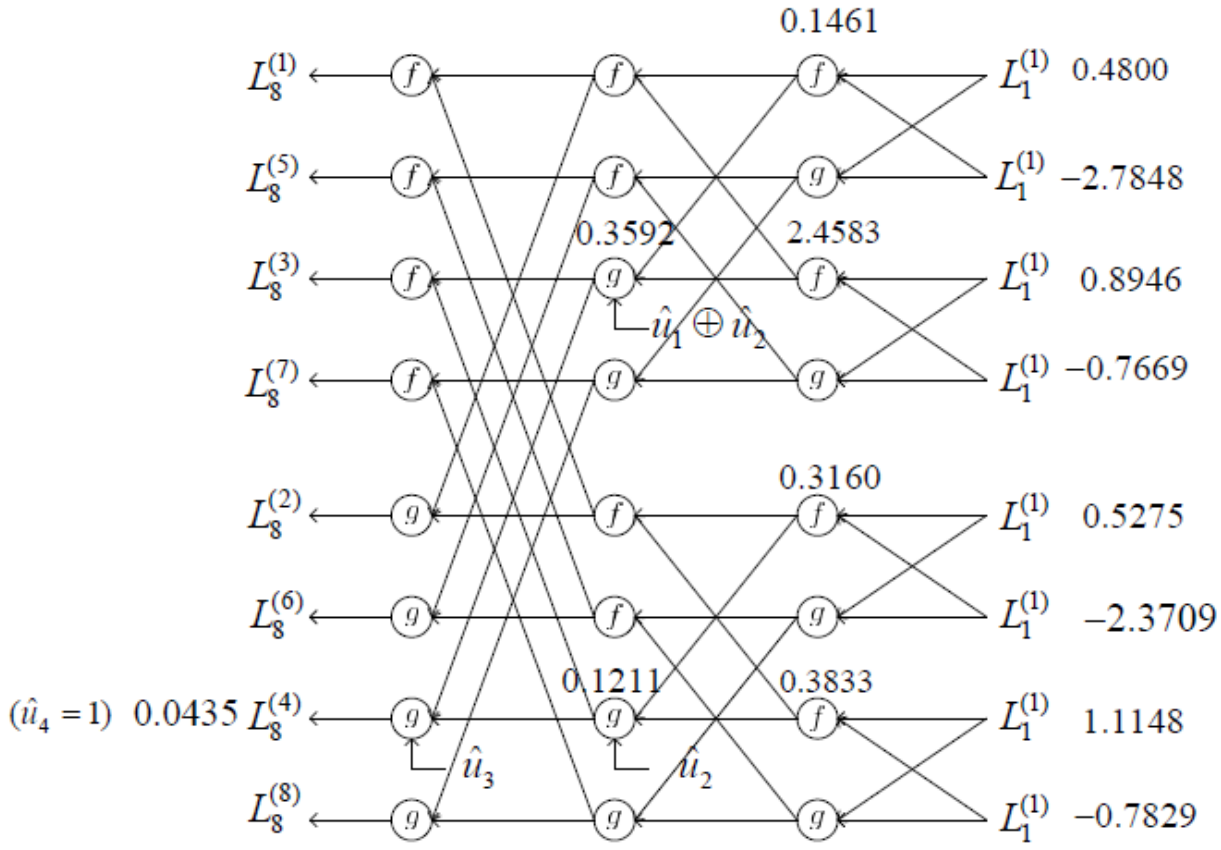


Figure 13 An example N=8 of SC decoder[43]

- *Step (1)* In the beginning, the rightmost side variables in Figure 12 are initialized as *{*0.4800, −2.7848, 0.8946, −0.7669, 0.5275, −2.3709, 1.1148, −0.7829*}*. These values are all channel observations
- *Step (2)* We judge whether *u*1 is an information bit before decoding it. Since the frozen set is *{*1, 2, 3, 5*}*, then we know that *u*1 is a frozen bit and its value is 0.
- *Step (3)* Similar to Step (2), we know that *u*2 and *u*3 are both frozen bits and there is no need to decode them. We skip them and decode *u*4 directly. When decoding *u*4, the likelihood ratio information is updated from right to left. We just take some of the

numerical values as an example. The value 0.1461 is obtained by calculating a *f* function: 1+0.4800 · (−2.7848)0.4800−2.7848 = 0.1461. The value 0.3592 is obtained by a *g* function 0.14611−2(^*u*1⊕^*u*2) · 2.4583 = 0.3592.

Note that we have known the values of *u*1 and *u*2 before calculating. The remaining values are obtained in a similar way. After calculating L(4)8 = 0.0435, we judge *u*4 to be 1 because *L*(4)8 is smaller than 1.The rest bits *u*5, ..., *u*8 are decoded in a similar method and show the full result 5in figure 30

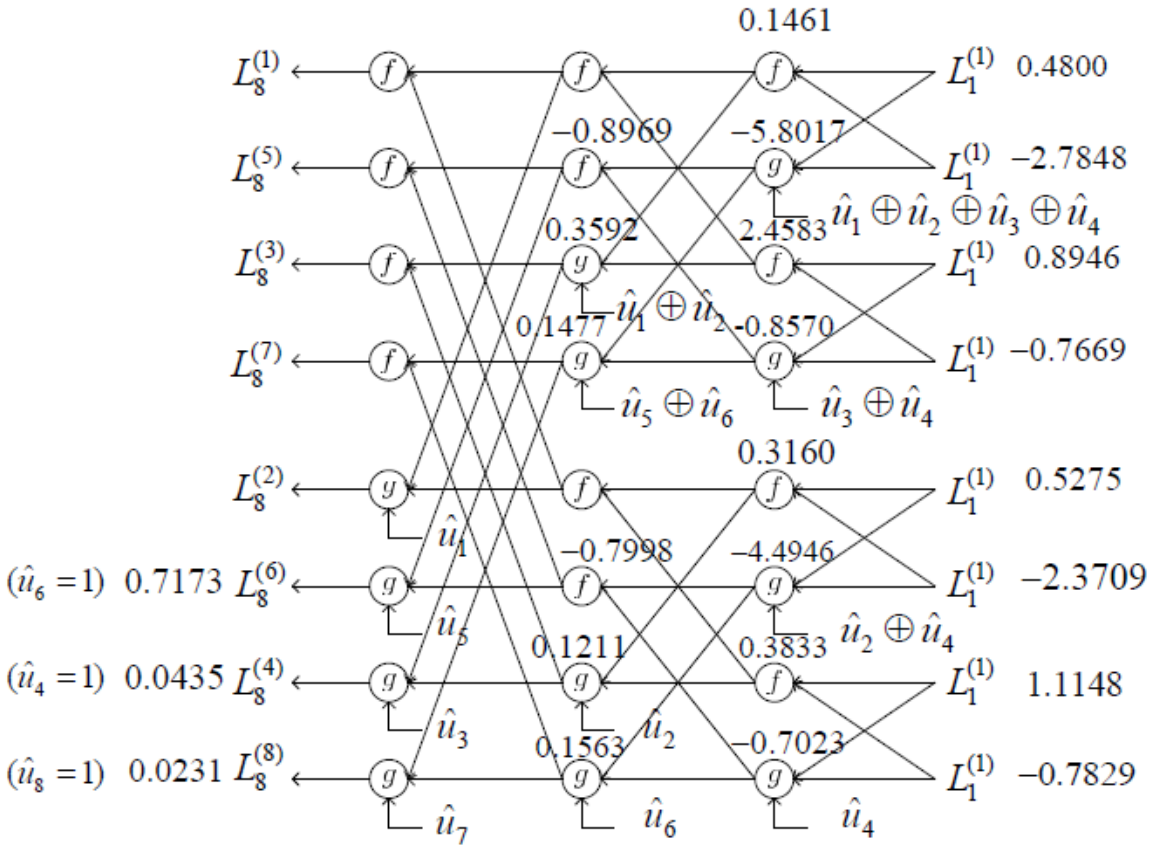


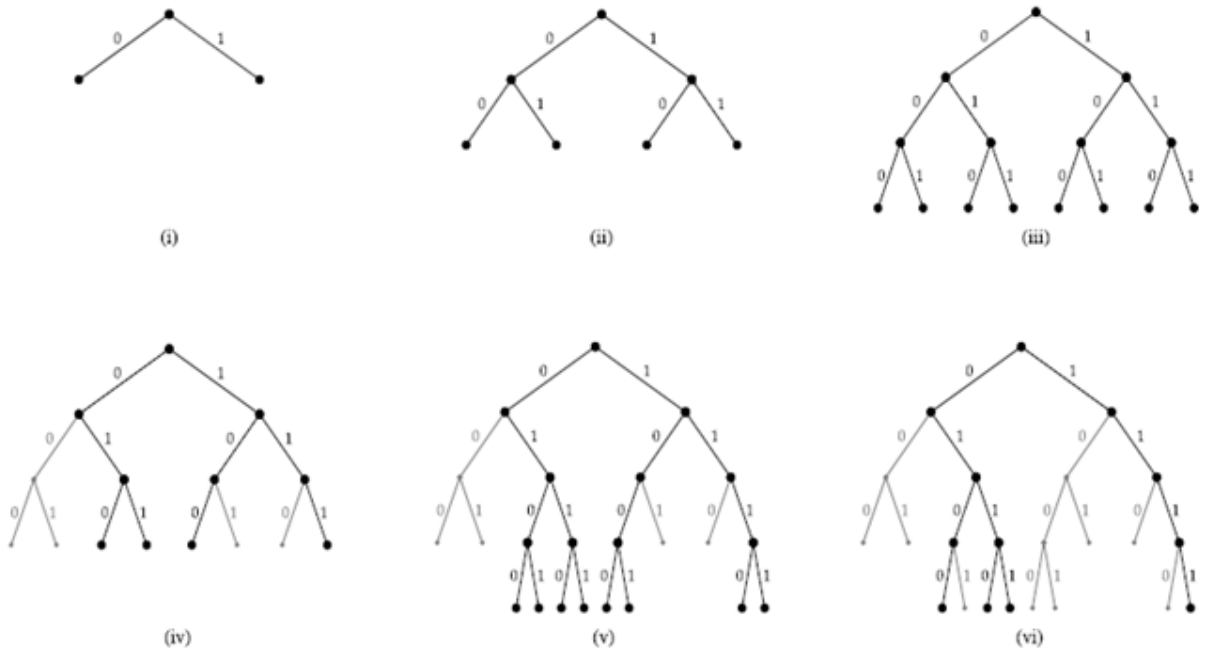
Figure 14 An example of N=8 SC decoder further steps[43]

3.2 Successive Cancellation List Decoding

Successive Cancellation List decoding (SCL) [4] is an extension of the basic successive cancellation(SC) decoder. In this kind of decoder, at each stage of decoding, L decoding paths are considered simultaneously, here L being an integer. This is very much like trellis decoding. After the decoding process comes to an end, the path that will be selected is the most likely probable path and that is nominated as the solitary codeword in the o/p of the decoder. Simulations depict about the consequential performance being too near to ML decoding, also for the reasonable values of *L*. Otherwise, if we allow a genie to choose the codeword that is transmitted from the list of the decoder, the outcomes are analogous with respect to the performance of Low Density Parity Check codes. It can be shown as simple cyclic redundancy check precoding can be implemented using such a genie. The quantity of decoding paths for individual information bit can be doubled by using precise listdecoding algorithm that attains this presentation, later L most likely paths are saved and all other paths are discarded using this

pruning process. Nevertheless, upfront execution of the procedure requires Ω (Ln^2) time, whereas the successive cancellation decoder requires $O(n \log n)$ as complexity of the novel SC decoder. Hence to implement such a decoder, some algorithmic alterations are applied to the existing polarcode in order to go through this problem. An effective, mathematically steady, execution of the projected list decoder is planned that takes only $O(L*n*\log n)$ time space and $O(L*n)$ space. We define a parameter known as the List Size L. Usually greater value of L specify lesser error rates nevertheless lengthier running times and larger memory usage. It can be well-known at this point that SCL decoding is not a new idea: Reed-Muller codes has also been implemented based on list decoding.

In the case of successive cancellation decoder, we must always decide about the value that is going to be of $\hat{u}_\phi$. In the case of a successive cancellation list decoder, we inspect both options as a substitute of determining to set unfrozen $\hat{u}_\phi$ to any one of 0 or 1 i.e. at each stage we inspect both options., Based on the values of $\mathbf{\hat{u}}^{\phi}_0$ let a "path" be a certain decision, for $0 \leq \phi < n$. When we split the decoding path $\mathbf{\hat{u}}^{\phi}_0$ into two paths when we decode a non-frozen bit $\hat{u}_{\phi+1}$. $\mathbf{\hat{u}}^{\phi}_0$ will be set as a prefix for the both new paths. Each among the two new paths will end with "0" and "1". As each fragmentation doubles the amount of paths to be inspected, we must prune all of the fragmented paths, and also extreme amount of paths allowable is L which happens to be the specified list size. Obviously, we would prefer to save the "best" paths at every phase. So, for this purpose we require a pruning criterion. The pruning criterion that we select is going to keep the paths that are most likely.



Evolution of decoding paths. We assume for simplicity that $n = 4$ and all bits are unfrozen. The list size is $L = 4$: each level has at most 4 nodes with paths that continue downward. Discontinued paths are colored gray. (i) Algorithm starts. First unfrozen bit can be either 0 or 1. (ii) Algorithm continues. Second unfrozen bits can be either 0 or 1. The number of paths is not more than $L = 4$, so no need to prune yet. (iii) Considering all options for first, second, and third bits results in 8 decoding paths; too much, since $L = 4$. (iv) Prune the 8 paths into $L = 4$ most promising paths. (v) Continue the 4 active paths by considering both options of the fourth unfrozen bit. The number of paths doubles to 8, which is too much ($L = 4$). (vi) Again, prune to $L = 4$ best paths.

Figure 15: Evolution of decoding paths [4]

A "lazy-copy" procedure is comprehended where the low-level functions and data structures can be described. Here our aim is to keep the algorithm describing procedure as simple as

possible, and some noticeable optimizations is evaded. Some of the data structures are described and set in Algorithm 1.

Algorithm 1 : `initializeDataStructures()`

```
1  inactivePathIndices ← new stack with capacity L
2  activePath ← new boolean array of size L
3  arrayPointer_P ← new 2-D array of size (m + 1) × L, the
   elements of which are array pointers
4  arrayPointer_C ← new 2-D array of size (m + 1) × L, the
   elements of which are array pointers
5  pathIndexToArrayIndex ← new 2-D array of size (m + 1) × L
6  inactiveArrayIndices ← new array of size m + 1, the elements
   of which are stacks with capacity L
7  arrayReferenceCount ← new 2-D array of size (m + 1) × L
   // Initialization of data structures
8  for λ = 0, 1, ..., m do
9      for s = 0, 1, ..., L − 1 do
10         arrayPointer_P[λ][s] ← new array of float pairs of size
           2^(m−λ)
11         arrayPointer_C[λ][s] ← new array of bit pairs of size
           2^(m−λ)
12         arrayReferenceCount[λ][s] ← 0
13         push(inactiveArrayIndices[λ], s)
14 for ℓ = 0, 1, ..., L − 1 do
15     activePath[ℓ] ← false
16     push(inactivePathIndices, ℓ)
```

[4]

An index is described for Each path *l* and where the value of *l* is $0 \leq l < L$. In the beginning of the algorithm only one path is active. There will be 2 states for paths termed as active and inactive as the algorithm takes its course and proceeds ahead. The indices of the inactive paths will be hold by the inactivePathIndices stack. An array implementation of stack is described where push as well as pop operations are taken into consideration and the complexity is *O(1)* time *O(L)* space is taken by a stack of size L. We describe an array called as activePath array which is a Boolean array. The activePath array contains a path called activePath[*l*] and this is true if and only if path *l* is active. Here one thing to be noted is that, the two inactivePathIndices and activePath store the identical information. Each layer can be denoted by λ and for each layer λ, a "bank" of *L* probability pair arrays is present for being used by the paths that are active. At any given instant several paths may use these arrays, while some may not be used by any of the arrays. We consider arrayPointer_P and every array of that kind is pointed by an element of the arrayPointer_P. Similarly, we will consider a bitpair array's bank and arrayPointer_C will be pointing to a bank of bitpair arrays.

Another pointer is the pathIndexToArrayIndex array and it will be used as follows. For a given path index *l* and given layer λ and, the two probabilities i.e. probability-pair array and bit-pair array matching to layer λ of path *l* will be pointed by

arrayPointer_P[λ][pathIndexToArrayIndex[λ][*l*]]

and

arrayPointer_C[λ][pathIndexToArrayIndex[λ][*l*]] , respectively.

Algorithm 2 : `assignInitialPath()`

Output: index ℓ of initial path

```
1  ℓ ← pop(inactivePathIndices)
2  activePath[ℓ] ← true
   // Associate arrays with path index
3  for λ = 0, 1, ..., m do
4  |  s ← pop(inactiveArrayIndices[λ])
5  |  pathIndexToArrayIndex[λ][ℓ] ← s
6  |  arrayReferenceCount[λ][s] ← 1
7  return ℓ [4]
```

Algorithm 3 : `clonePath(`ℓ`)`

Input: index ℓ of path to clone
Output: index ℓ' of copy

```
1  ℓ' ← pop(inactivePathIndices)
2  activePath[ℓ'] ← true
   // Make ℓ' reference same arrays as ℓ
3  for λ = 0, 1, ..., m do
4  |  s ← pathIndexToArrayIndex[λ][ℓ]
5  |  pathIndexToArrayIndex[λ][ℓ'] ← s
6  |  arrayReferenceCount[λ][s]++
7  return ℓ' [4]
```

Algorithm 4 : `killPath(`ℓ`)`

Input: index ℓ of path to kill

```
   // Mark the path index ℓ as inactive
1  activePath[ℓ] ← false
2  push(inactivePathIndices, ℓ)
   // Disassociate arrays with path index
3  for λ = 0, 1, ..., m do
4  |  s ← pathIndexToArrayIndex[λ][ℓ]
5  |  arrayReferenceCount[λ][s]−−
6  |  if arrayReferenceCount[λ][s] = 0 then
7  |  |  push(inactiveArrayIndices[λ], s) [4]
```

Algorithm 5 : `getArrayPointer_P(`λ, ℓ`)`

Input: layer λ and path index ℓ
Output: pointer to corresponding probability pair array

```
   // getArrayPointer_C(λ, ℓ) is defined
      identically, up to the obvious changes
      in lines 6 and 10
1  s ← pathIndexToArrayIndex[λ][ℓ]
2  if arrayReferenceCount[λ][s] = 1 then
3  |  s' ← s
4  else
5  |  s' ← pop(inactiveArrayIndices[λ])
6  |  copy the contents of the array pointed to by
   |  arrayPointer_P[λ][s] into that pointed to by
   |  arrayPointer_P[λ][s']
7  |  arrayReferenceCount[λ][s]−−
8  |  arrayReferenceCount[λ][s'] ← 1
9  |  pathIndexToArrayIndex[λ][ℓ] ← s'
10 return arrayPointer_P[λ][s'] [4]
```

Until now we have described the procedures and algorithms to show data-structures are set and used, now we can build our decoder on the basis of these data structures and discourse the low level functions which make the paths active and inactive. Algorithm2 is known by the function name assigninitialPath() where in the initial path of the algorithm is assigned and allocated i.e. we select a path index *l* which is not presently in use, and mark it as already have being used. After this algorithm, each of the layer λ, we mark an index *s* (through pathIndexToArrayIndex) such that both pointers i.e. both arrayPointer_P[λ][*s*] and arrayPointer_C[λ][*s*] are assigned to the present path. After Algorithm2 we describe Algorithm3 which is basically used to clone the path. This happens to be the final step before completely dividing or splitting the particular path into two paths. The logic is very similar to that of Algorithm 2, but now we make the two paths share bit-arrays and probability arrays. Algorithm 4 is a function known as *killpath(l)* and it is used to kill a path or terminate a path, and this is attained by marking the path as an inactive path. After this is done, the arrays marked as associated with the path must be dealt with as follows.

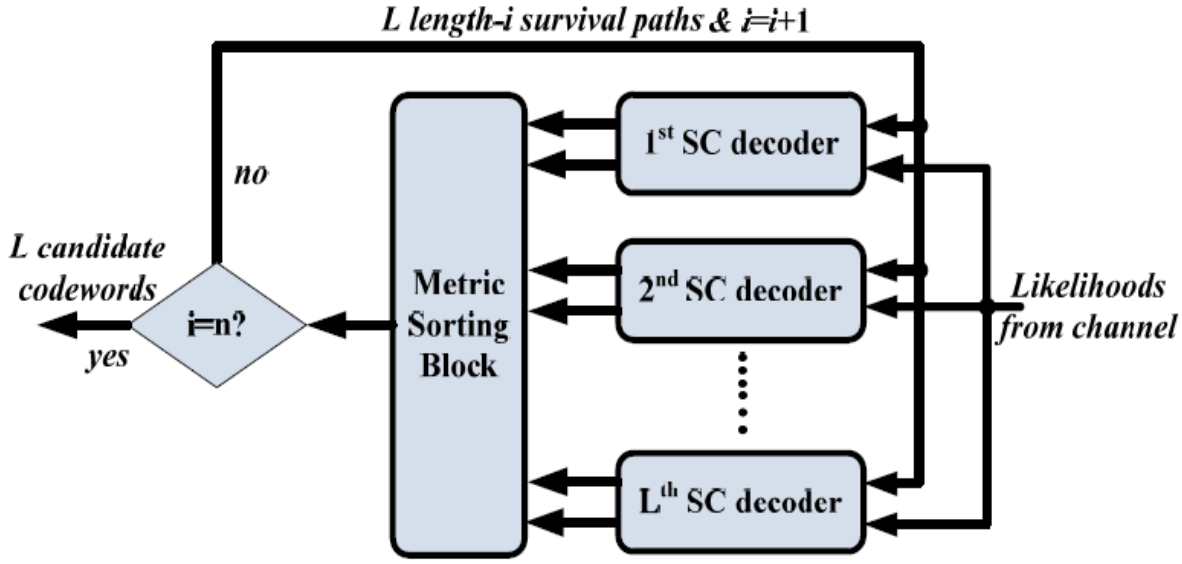


Figure 16: Block diagram of L-Size SCL decoder[44]

If the paths are not active we think them as not having any arrays associated with them. If this is true then all the arrays which are earlier known to have been related along the paths should keep its own reference count decremented by 1. The main aim of each of the earlier debated lowlevel functions is basically to allow the abstraction realized using functional pointers which are named as getArrayPointer_P and getArrayPointer_C. Whenever there is a need for the higherlevel functions to contact (for writing or reading any one of the following reasons) the array's probability-pair related with a some or the other layer λ and some or the other path *l*, getArrayPointer_P is the function is that is going to be called. Algorithm 5 describes the implementation of getArrayPointer_P. Here two cases can be well thought-out: The array can be associated with more than one path or it cannot be associated with more than one path. If it is not associated with more than one path, then we need not do anything but just an array. On the other hand, if the array is common or shared, a private copy for path *l* is created, also a pointer to that copy is returned. By doing so, we can make sure that a single array is not written by two paths. The functional pointer getArrayPointer_C is used in a similar

way for bitpair arrays, and it exactly does have a similar employment, to the changes that are going to be noticeable. Midlevel functions are introduced in the next section and they are described by the 6 and 7 Algorithm for the successive cancellation list decoding. Therefore, in order to guarantee about the steadiness of calculations is conserved, in spite of numerous paths sharing information, the new execution takes the usage of the functions getArrayPointer_C and getArrayPointer_P. Also, the code to normalize probabilities in contained in Algorithm6. This is done to achieve a particular goal i.e. to avoid the underflow of floating point. An important point to be noted here is that the "fresh pointer" condition we have imposed on ourselves holds true. Algorithm 10 is a good example for this to be considered. One important point to be noted here is that from any of the algorithms either *killpath(l)* or *clonepath(l)* is not called and algorithm 7 also holds true for this relation and so we can say that the "fresh pointer" condition is satisfied. Let us now jump to the lines from line number 20 of Algorithm10. One fact to be recalled is that arbitrarily small positive reals cannot be holded by floating point variables, and even if we go ahead and try to this the very small values will be rounded off to 0 and may go wrong and we underflow is the word used to define this scenario. By now we can say that all our earlier executions were susceptible to to "underflow" and these were the successive cancellation list decoders. To understand this well we can consider line 2of algorithm1. We represent the random vectors that are matching to **Y** and **U** by the respective random vectors matching to **y** and **u**, respectively. For $b \in \{0, 1\}$ we have the equation

$$\begin{aligned} & W_m^{(\varphi)}(\mathbf{y}_0^{n-1}, \hat{\mathbf{u}}_0^{\varphi-1} | b) \\ & \quad = 2 \cdot \mathbb{P}(\mathbf{Y}_0^{n-1} = \mathbf{y}_0^{n-1}, \mathbf{U}_0^{\varphi-1} = \hat{\mathbf{u}}_0^{\varphi-1}, \mathsf{U}_\varphi = b) \\ & \quad \leq 2 \cdot \mathbb{P}(\mathbf{U}_0^{\varphi-1} = \hat{\mathbf{u}}_0^{\varphi-1}, \mathsf{U}_\varphi = b) = 2^{-\varphi}. \end{aligned}$$

Algorithm 6 : recursivelyCalcP(λ, φ)[4] *list version*

Input: layer λ and phase φ

```
1  if λ = 0 then return // Stopping condition
2  set ψ ← ⌊φ/2⌋
   // Recurse first, if needed
3  if φ mod 2 = 0 then recursivelyCalcP(λ − 1, ψ)
   // Perform the calculation
4  σ ← 0
5  for ℓ = 0, 1, ..., L − 1 do
6      if pathIndexInactive(ℓ) then
7          continue
8      P_λ ← getArrayPointer_P(λ, ℓ)
9      P_{λ−1} ← getArrayPointer_P(λ − 1, ℓ)
10     C_λ ← getArrayPointer_C(λ, ℓ)
11     for β = 0, 1, ..., 2^{m−λ} − 1 do
12         if φ mod 2 = 0 then
                                   // apply Equation (4)
13             for u' ∈ {0, 1} do
14                 P_λ[β][u'] ←
                   Σ_{u''} (1/2) P_{λ−1}[2β][u' ⊕ u''] · P_{λ−1}[2β + 1][u'']
15                 σ ← max(σ, P_λ[β][u'])
16         else                    // apply Equation (5)
17             set u' ← C_λ[β][0]
18             for u'' ∈ {0, 1} do
19                 P_λ[β][u''] ←
                   (1/2) P_{λ−1}[2β][u' ⊕ u''] · P_{λ−1}[2β + 1][u'']
20                 σ ← max(σ, P_λ[β][u''])
   // normalize probabilities
21 for ℓ = 0, 1, ..., L − 1 do
22     if pathIndexInactive(ℓ) then
23         continue
24     P_λ ← getArrayPointer_P(λ, ℓ)
25     for β = 0, 1, ..., 2^{m−λ} − 1 do
26         for u ∈ {0, 1} do
27             P_λ[β][u] ← P_λ[β][u]/σ
```

Algorithm 7 : recursivelyUpdateC(λ, φ) *List Version*

Input: layer λ and phase φ
Require : φ is odd

```
1  set ψ ← ⌊φ/2⌋
2  for ℓ = 0, 1, ..., L − 1 do
3      if activePath[ℓ] = false then continue
4      set C_λ ← getArrayPointer_C(λ, ℓ)
5      set C_{λ−1} ← getArrayPointer_C(λ − 1, ℓ)
6      for β = 0, 1, ..., 2^{m−λ} − 1 do
7          C_{λ−1}[2β][ψ mod 2] ← C_λ[β][0] ⊕ C_λ[β][1]
8          C_{λ−1}[2β + 1][ψ mod 2] ← C_λ[β][1]
9  if ψ mod 2 = 1 then
10     recursivelyUpdateC(λ − 1, ψ)[4]
```

Algorithm 8 : SCL Decoder, Main Loop

Input: the received vector $\mathbf{y}$ and a list size L as a global
Output: a decoded codeword $\hat{\mathbf{c}}$

// Initialization
initializeDataStructures()
$\ell \leftarrow$ assignInitialPath()
$P_0 \leftarrow$ getArrayPointer_P$(0, \ell)$
for $\beta = 0, 1, \ldots, n-1$ **do**
  set $P_0[\beta][0] \leftarrow W(\mathsf{y}_\beta|0)$, $P_0[\beta][1] \leftarrow W(\mathsf{y}_\beta|1)$
// Main loop
for $\varphi = 0, 1, \ldots, n-1$ **do**
  recursivelyCalcP(m, φ)
  if u_φ *is frozen* **then**
    for $\ell = 0, 1, \ldots, L-1$ **do**
      if activePath$[\ell] =$ **false then continue**
      $C_m \leftarrow$ getArrayPointer_C(m, ℓ)
      set $C_m[0][\varphi \mod 2]$ to the frozen value of u_φ
  else
    continuePaths_UnfrozenBit(φ)
  if $\varphi \mod 2 = 1$ **then**
    recursivelyUpdateC (m, φ)
// Return the best codeword in the list
$\ell' \leftarrow 0$, $p' \leftarrow 0$
for $\ell = 0, 1, \ldots, L-1$ **do**
  if activePath$[\ell] =$ **false then continue**
  $C_m \leftarrow$ getArrayPointer_C(m, ℓ)
  $P_m \leftarrow$ getArrayPointer_P(m, ℓ)
  if $p' < P_m[0][C_m[0][1]]$ **then**
    $\ell' \leftarrow \ell$, $p' \leftarrow P_m[0][C_m[0][1]]$
set $C_0 \leftarrow$ getArrayPointer_C$(0, \ell')$
return $\hat{\mathbf{c}} = (C_0[\beta][0])_{\beta=0}^{n-1}$ [4]

3.3 CRC Aided Successive Cancellation List Decoding

3.3.1 Introduction to CRC Decoding

Cyclic redundancy check (CRC) is the most widely used technique in error detection and error correction in the space of information theory and coding. It is also widely used in cellular standards. For instance, the 3rd generation partnership project incorporates CRC widely in most of its radio access technologies like UMTS, LTE etc. If we consider an input block of K bit of the error correcting encoder and information bits of length k and the length of the CRC sequence as m-bit, i.e. K = k+m. CRC bits can be looked at as part of source bits for the error correcting code, the code rate R would then be defined as R=K/N. This section tries to present a combination of both the SCL decoder which is aided by CRC that improves the performance of the polarcode further we propose a combination of SCL (SCS) decoder and CRC detector to further improve the performance of polar codes. As shown in figure 16, at the receiver, the SCL (SCS) decoder outputs the candidate sequences into CRC detector and the latter feeds the check results back to help the codeword determination. We refer to such a decoding scheme as CRC aided SCL/SCS (CA-SCL/SCS). The performance of CASCL/ SCS is substantially improved and even outperforms that of turbo codes.

Stopping criterion is one of the most important thing to be considered when we talk about CRC aided decoding. When employing iterative decoding process a stopping criterion is provided by the CRC results or any retransmission requests are started. Traditionally, CRC results provide a stopping criterion for the iterative decoding process or start retransmission requests. In this section, we try to dig on the checking information which is provided by a CRC detector. We essentially make use of the CRC detector and CRC detector in a codeword selection mechanism provides a checking information. The main goal of this section is to provide details on the fact that polarcode performance can be improved further if it is aided by a cyclic redundancy check along with using a polar SCL decoder.

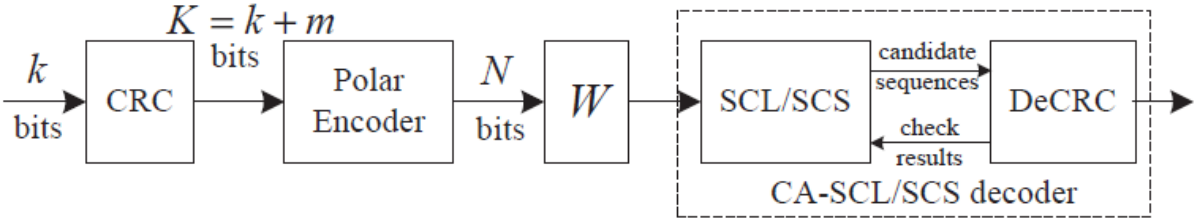


Figure 17: CRC aided polar code decoding [6]

3.2.2 CRC-Aided Successive Cancellation List Decoding [4][6]

The performance of the SC algorithm can be improved by using SCL decoding algorithm, which tracks L best decoding paths together. The performance can be further improved by introducing CRC to SCL decoding by selecting a CRC valid path among L best decoding paths at the end of decoding. However, SCL decoding algorithm suffers from long latency and low throughput due to high complexity, *O(L N log N)* calculations as L and N increases. The throughput of the SCL can be improved by using an adaptive decoder, which provides the SC throughput with the SCL performance.

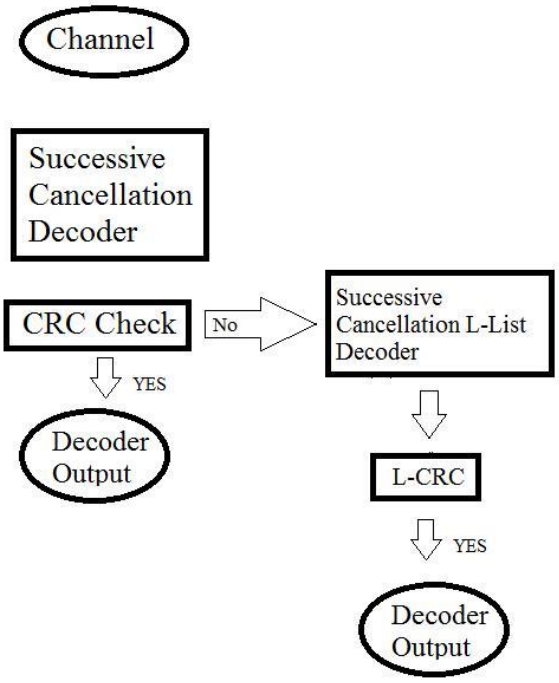


Figure 18: Data flow of the CRC aided decoder [4][32]

The decoder has three main components, SC, SCL and CRC decoders. Initially, the SC decoder is activated and a hard decision estimate vector is calculated. After that, the CRC decoder controls whether the hard decision vector is correct. If the CRC is valid, the hard decision vector is quite likely to be the correct information vector. In this case, the adaptive decoder is immediately terminated without the activation of the SCL decoder. In other case, when the CRC is invalid, the SCL decoder is activated and L information vector candidates are generated. Among these candidates, the CRC decoder selects am candidate, which has a valid CRC vector. If more than one CRC vector candidate is valid, the most probable one among these candidates is selected. Lastly, when none of the candidates has a valid CRC vector, the CRC decoder selects the most probable decision estimation vector to reduce BER.

Chapter 4 – Adaptive Polar Decoders

4.1 Introduction to Adaptive Polar Decoder

The simulation and construction of Adaptive SCL[5][6] (adaptive successive cancellation list decoder with cyclic redundancy check for polar codes is described here in the chapter. In the earlier chapter, we saw that CRC was implemented to successive cancellation list decoder. The main criterion here is that at least one path should pass CRC. Adaptive SCL makes use of iteration. By this the list size increases iteratively until at least anyone survival path passes cyclic redundancy check. Simulation using Matlab or C predicts about the adaptive successive cancellation list decoder gives enormous complexity reduction. Complexity reduction is one of the main tasks in any of the decoder. Simulations using Matlab or C show that polar code with N=2048 and k=1024 implemented using a 24bit cyclic redundancy check decoded utilizing an adaptive SCL decoder will attain an FER $\leq 10^{-4}$ at SNR of 1.1db with very large maximum size, that is like 0.2dB from the Shannon limit [5].

For improving the quality of performance of the Polarcode, the Polar code concatenation with CRC was proposed earlier. The main idea here is that the decoder does SCL decoding and then carries out a cyclic redundancy check on all the paths that survive which remain on list of decoders at the last step of SCL decoding. By doing so the redundancy check knocks out the invalid or wrong paths.

The fact can be well understood how the concatenation of polarcode with redundancy check returns a gain in performance, the following example can be considered. If we consider information sequence of different lengths i.e. a sequence of length of information sequences of 11648 that can produce a weight16 codewords and length of information sequences of 215040 that can produce the weight24 code words it is found that for the polarcode, and 16bit redundancy checks on them. It is found that for the five of the six CRC's, all the 16weight and 24weight codewords are knocked out by the redundancy check [3]. So, the 16bit redundancy checks significantly upgrade the min Hamming distance for polarcode N=2048 and k=1024 all the way from 16 through up to 32. By this we can get to know that very high enhancement is achieved. In fact, simulation depicts the concatenation of polarcode with N=2048 and k=1024 with a 16bit redundancy check using the SCL decoder that seems to perform also even much better than more better than the performance given by the non-concatenated polarcode.

4.2 Adaptive Polar Decoder Algorithm

The Algorithm for Polar Codes Adaptive Decoder is as follows [5]

Step 1. Initialize $L = 1$ for the SC-list decoder.
Step 2. Perform the SC-List decoding, and then perform CRC on each survival path at the end of SC-List decoding.
Step 3. If there is one or more than one paths passing CRC, output the path passing CRC and with highest probability, and exit decoding; otherwise go to 4;
Step 4. Update L to $2\times L$. If $L \leq L\max$, go to 2; otherwise output the path with the highest probability and exit decoding;

Even though the successive cancellation list decoder using list size *L* being 32 is able to reach Maximum likelihood performance of a polarcode having N=2048 and also k=1024 without the redundancy check, it seems that, due to some reason to be explained later, with redundancy check, to exploit the benefit of increased distance a big L is needed. Infact, simulations explain us about the performance of the successive cancellation list decoding of the polarcode using a N=2048 and a k=1024 using the 16bit cyclic redundancy check is not up to the maximum likelihood performance bound although when *list size L* is of the order of about a million. Therefore, for achieving Maximum likelihood performance, the polar code N=2048 and k=1024 using the 16bit cyclic redundancy check has to have a larger list size, that which gets us to the point of a very huge complexity of the decoder. For almost all of the frames that are received, the polar SCL decoder that which has a small list size *L* can be able to decode the information bits successfully, and in doing so a very few frames are there that may be needing big list size for achieving a successful decoding. Therefore, for bringing down the difficulty of complexity of decoding, an adaptive successive cancellation list decoder for polar codes with cyclic redundancy check. The adaptive successive cancellation list decoder firstly makes use of a small list size L, and then gradually tries to increase list size iteratively (if no paths pass the redundancy check), until the list size *L* goes up to a predefined number maximum list size i.e. *L*max.

We consider a BPSK modulation technique and the channel considered is additive white Gaussian noise(AWGN). As we can see form the table below [3] at FER = 10^{-3} there is almost 0.4dB gain of maximum list size *L*max = 8190 over *L*max = 32. As the list mostly has the weight16 and weight24 code words, and codewords generated by the information sequences cannot go through the redundancy check, the FER is in fact very much influenced by the probability that list doesn't contain the survival path. When we increase the list size L, we are doing nothing but just increasing the probability that List contains the correct path and therefore we can get better performance results.

Table 1 Adaptive successive Cancellation decoder with mean [5]

THE MEAN OF L OF THE ADAPTIVE SC-LIST DECODER

E_b/N_o(dB)	1.0	1.2	1.4	1.6	1.8	2.0
L_{max}=32	16.64	8.03	3.86	2.04	1.39	1.14
L_{max}=128	35.31	12.16	4.52	2.17	1.41	
L_{max}=512	70.41	19.14	5.45	2.27		
L_{max}=2048	133.40	30.80	6.64	2.36		
L_{max}=8192	271.07	52.59	7.88	2.47		

Now coming to the comparison of adaptive successive cancellation decoder's error probability with the max list size of L_{max} along with the error probability of an nonadoptive decoder with constant maximum list size of *L*max. If the list contains the correct path that is generated from an adaptive successive cancellation list decoder with a list size of *L* accurately being lesser than maximum list size *L*max, it is very much likely that the list generated by the

non-adaptive decoder with a bigger list size of *L*max contains correct path, so the two decoders can be decoded successfully. If we consider on the other hand i.e. if there is no correct survival path in the adaptive decoder's list, it will keep on increasing the list size *L* until *L* = *L*max i.e. the list size equals the maximum list size, and this thus leads to both the decoders that will use the similar *L* = *L*max and thus the two decoders perform in a similar fashion. Therefore, the two decoders perform similarly and the performance of the adaptive SC-List decoder with maximum list size *L*max = 32 and maximum list size *L*max = 8192 is almost to be very near to the nonadoptive successive cancellation decoder with constant list size *L* = 32 and list size *L* = 8192, respectively.

The mean of various list size *L* and for different maximum list size of *L*max is shown in Table1. As the Eb/N0 increases, the adaptive polar SCL decoder has more probability to decode the frames that are received using a similar list size *L*, so thus the mean of the respective list size *L* gets lesser for the adaptive successive cancellation list decoder. Because the implementation complexity of the successive cancellation list decoder is linear with respect to the L, the successive cancellation list decoder using a fixed list size *L* has a complexity of the order of O(*L N* log *N*) and the adaptive SCL decoder has on an average complexity of O(*L N* log *N)* i.e. both are almost same. It can be seen that under the maximum list size *L*max equals 32, *L* = 2.04 is the mean of List Size L for *SNR* = 1.6dB; It comes to about sixteen times more complexity getting lesser but with along with the same performance as when compared to the constant list size *L* being 32. *L* = 2.47 is the mean of the list size L *under* maximum list size *L*max = 8192 for *SNR* = 1.6dB; It comes to almost 3311 times complexity getting lesser except that the same performance range compared to the constant list size of *L* = 8192. Thus, It can be noted that the adaptive successive cancellation list decoder with a very bigger *L*max = 262144 for the polarcode of N=2048 and k=1024 with a 24bit redundancy check, the mean list size *L* = 819[5]. Frame Error Rates$\leq 10^{-3}$ at *SNR's* of 1.1dB can be obtained using the concatenated code outlined here. It is noticeable about the fact i.e. for a large number of polarcode channels, the formula for maximum rate which is attained at a block length *N* and frame error rate can be approximated using[5]:

$$R = C - \sqrt{\frac{V}{N}} Q^{-1}(\epsilon)$$

Here the channel capacity is *C* and the channel dispersion is a quantity called *V*. This channel dispersion can be calculated using the stats of the channel, by the equation [5]:

$$V = \mathrm{Var}\left[\log \frac{f(Y|X)}{f(Y)}\right]$$

Here conditional probability density for a channel output given inputs can be described as f(y|x), *f*(*y*) can be described as the probability density of the o/p given the condition the i/p are distributed as the distribution which is capacity achieving, and the random variable *X* is to be distributed as the distribution achieving capacity with Y as being an output of the channel respectively. For the binary input AWGN channel with input *X* is 1 or −1 and variance of noise σ^2,

$$f(y|x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(y-x)^2}{2\sigma^2}\right]$$

and along this capacity attaining distribution being either +1 or −1 as equally probable, the density output can be described using [5]

$$f(y) = \frac{1}{2}\left[f(y|1) + f(y|-1)\right]$$

So, we can calculate that to get a rate R = (1024 − 24)/2048 = 0.488, the minimum *Eb/No* required is said to be 0.85 dB. Hence the system here is about to be 0.20 dB from the information theoretic boundary. Thus, this performance is even most tough to be reproduced using successive cancellation list decoder with an almost constant list size of L being 262144.

Chapter 5 Results and Conclusion

5.1 Results

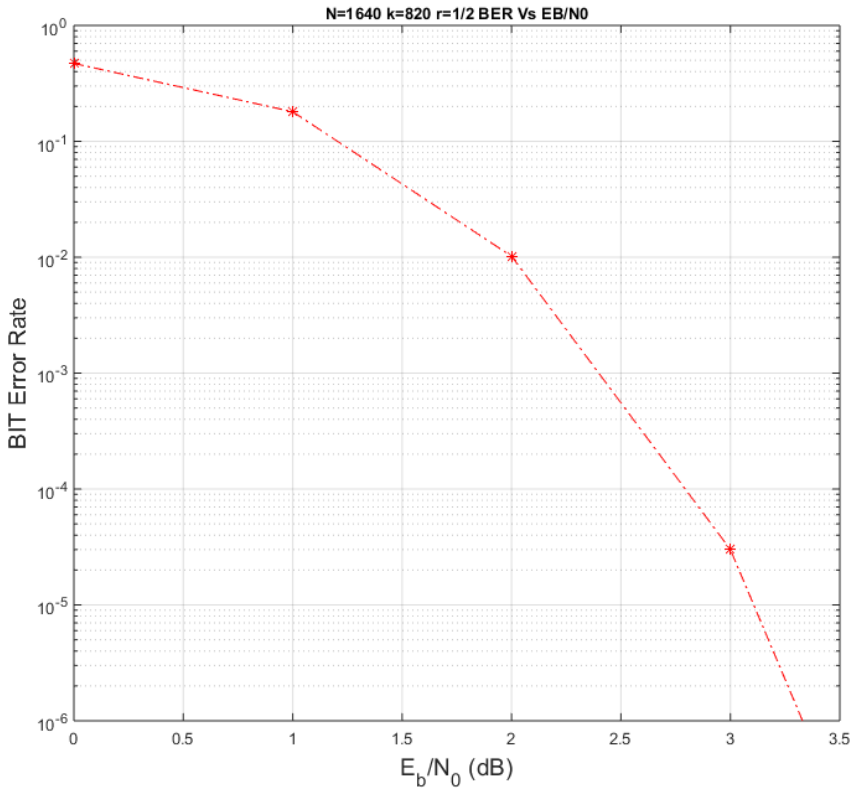


Figure 19 Successive Cancellation for R=1/4

Table 2 Successive Cancellation decoder for k=256 R=1/4

EB/No	berrors	bits	BER	ferrors	frames	FER	ACS
0	434	2560	0.169531	190	512	0.371094	15.5602
1	410	2560	0.160156	178	512	0.347656	15.1887
2	272	2560	0.10625	129	512	0.251953	14.3465
3	235	2560	0.091797	106	512	0.207031	13.7742
4	194	2560	0.075781	89	512	0.173828	13.5629
5	160	2560	0.0625	69	512	0.134766	13.1906
6	99	2560	0.038672	47	512	0.091797	12.993
7	51	2560	0.019922	24	512	0.046875	12.6941
8	51	4530	0.011258	27	906	0.029801	12.5854
9	52	4905	0.010601	27	981	0.027523	12.4997
10	50	7275	0.006873	25	1455	0.017182	12.359

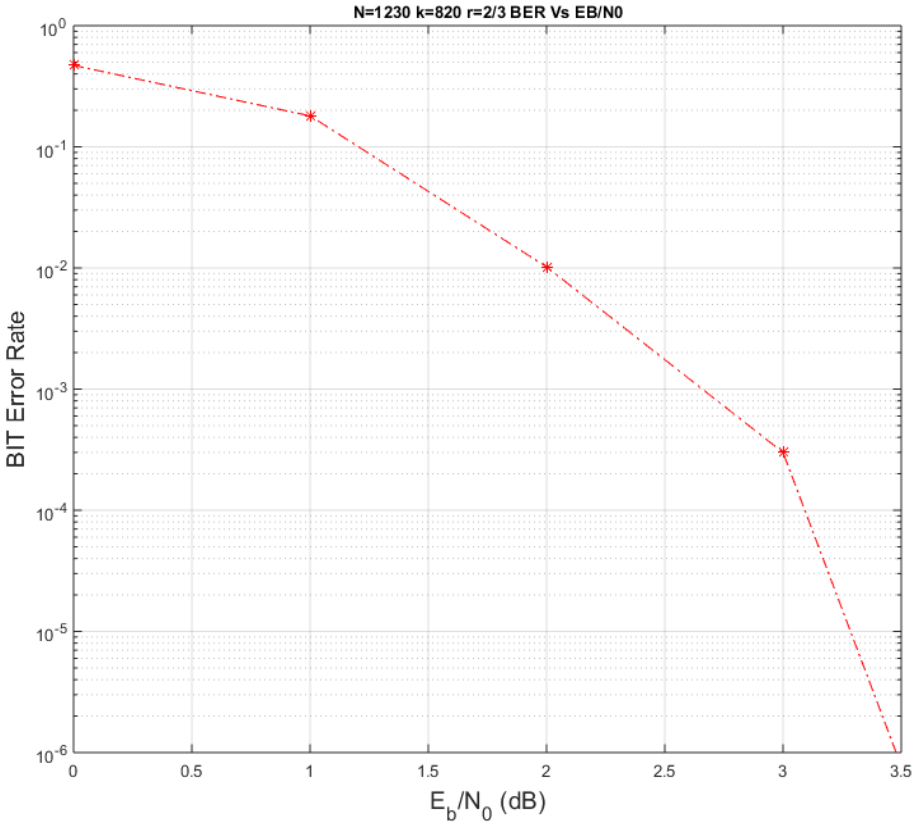


Figure 20Successive Cancellation for R=1/2

Table 3 Successive Cancellation decoder for k=512 R=1/2

EB/No	berrors	bits	BER	ferrors	frames	FER	ACS
0	849	5120	0.16582	375	1024	0.366211	15.6453
1	774	5120	0.151172	346	1024	0.337891	15.1029
2	551	5120	0.107617	250	1024	0.244141	14.4008
3	479	5120	0.093555	221	1024	0.21582	13.915
4	374	5120	0.073047	161	1024	0.157227	13.5791
5	269	5120	0.052539	120	1024	0.117188	13.1209
6	159	5120	0.031055	76	1024	0.074219	12.959
7	101	5415	0.018652	51	1083	0.047091	12.7505
8	100	6770	0.014771	45	1354	0.033235	12.613
9	101	7580	0.013325	48	1516	0.031662	12.4763
10	101	15040	0.006715	49	3008	0.01629	12.3941

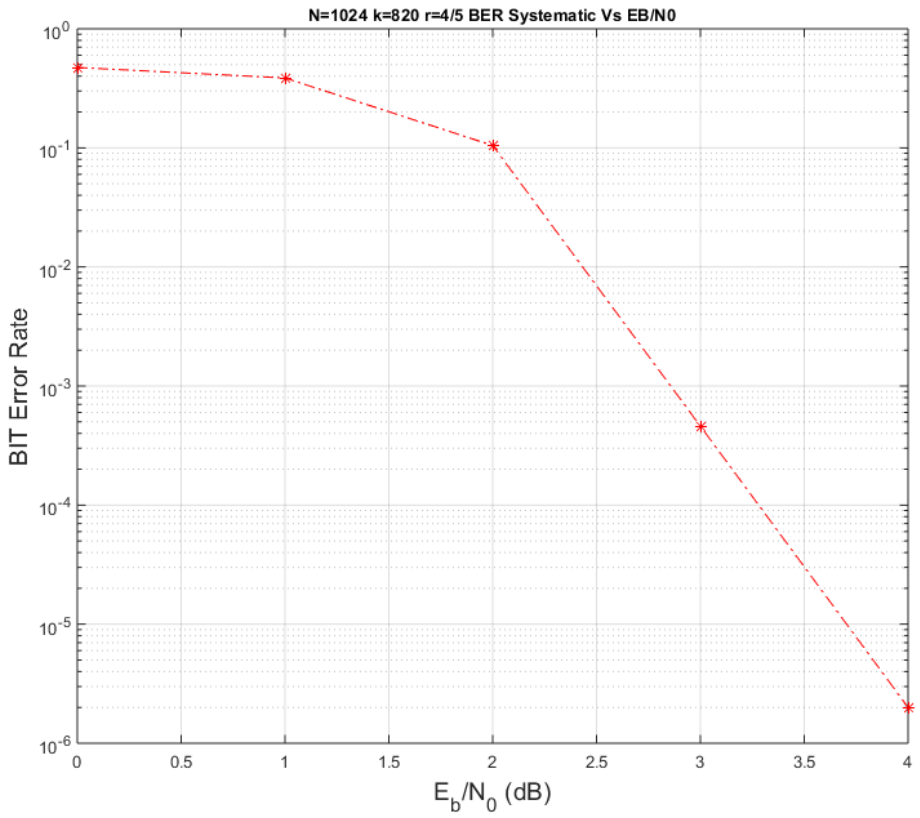


Figure 21 Successive Cancellation for R=3/4

Table 4 : Successive Cancellation decoder for k=768 R=3/4

EB/No	berrors	bits	BER	ferrors	frames	FER	ACS
0	928	5120	0.18125	412	1024	0.402344	15.6299
1	785	5120	0.15332	345	1024	0.336914	15.0498
2	524	5120	0.102344	247	1024	0.241211	14.3779
3	473	5120	0.092383	213	1024	0.208008	13.9822
4	306	5120	0.059766	137	1024	0.133789	13.5279
5	282	5120	0.055078	129	1024	0.125977	13.2113
6	201	5120	0.039258	85	1024	0.083008	12.9604
7	137	5120	0.026758	62	1024	0.060547	12.6883
8	101	6160	0.016396	46	1232	0.037338	12.5813
9	100	14015	0.007135	51	2803	0.018195	12.4859
10	100	13020	0.00768	48	2604	0.018433	12.3889

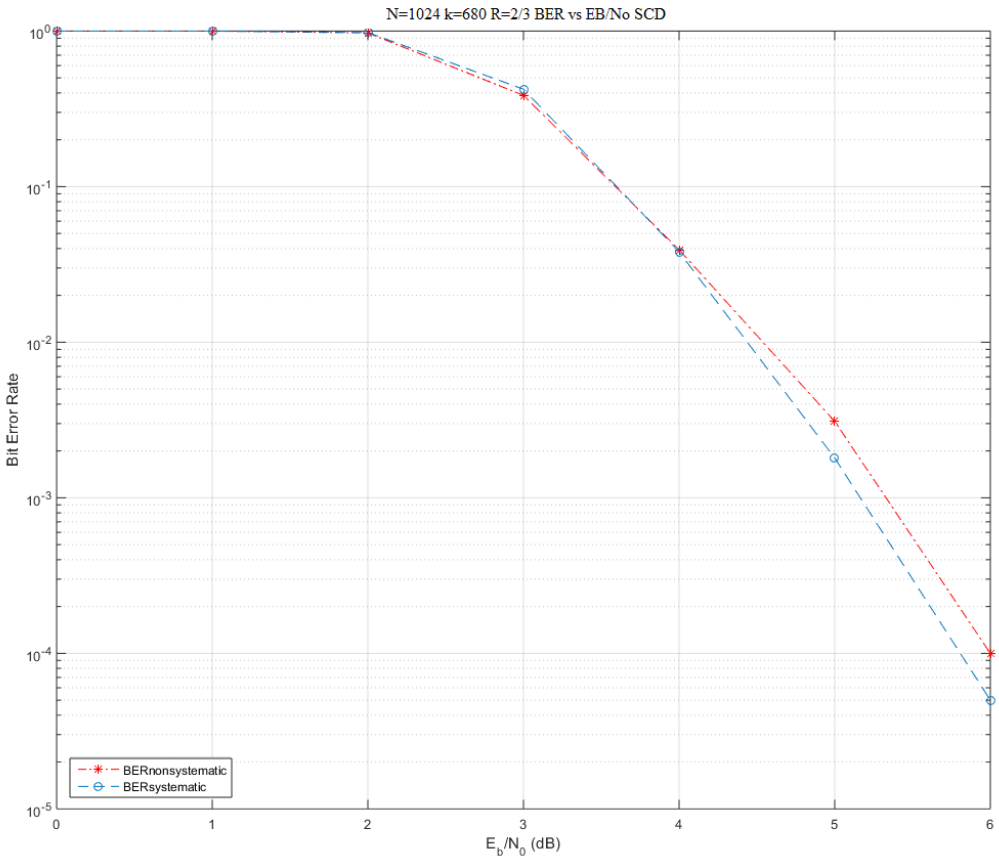


Figure 22BER Vs EB/No R=2/3 N=1024

Table 5: BER and FER recorded for N=1024 k=687 R=2/3 non-systematic

Eb/N0 (dB)	BER	FER
0	0.473	1
1	0.3862	0.992
2	0.1047	0.505
3	0.0046	0.0414
4	0.0001	0.0016
5	0.000032	0.00054

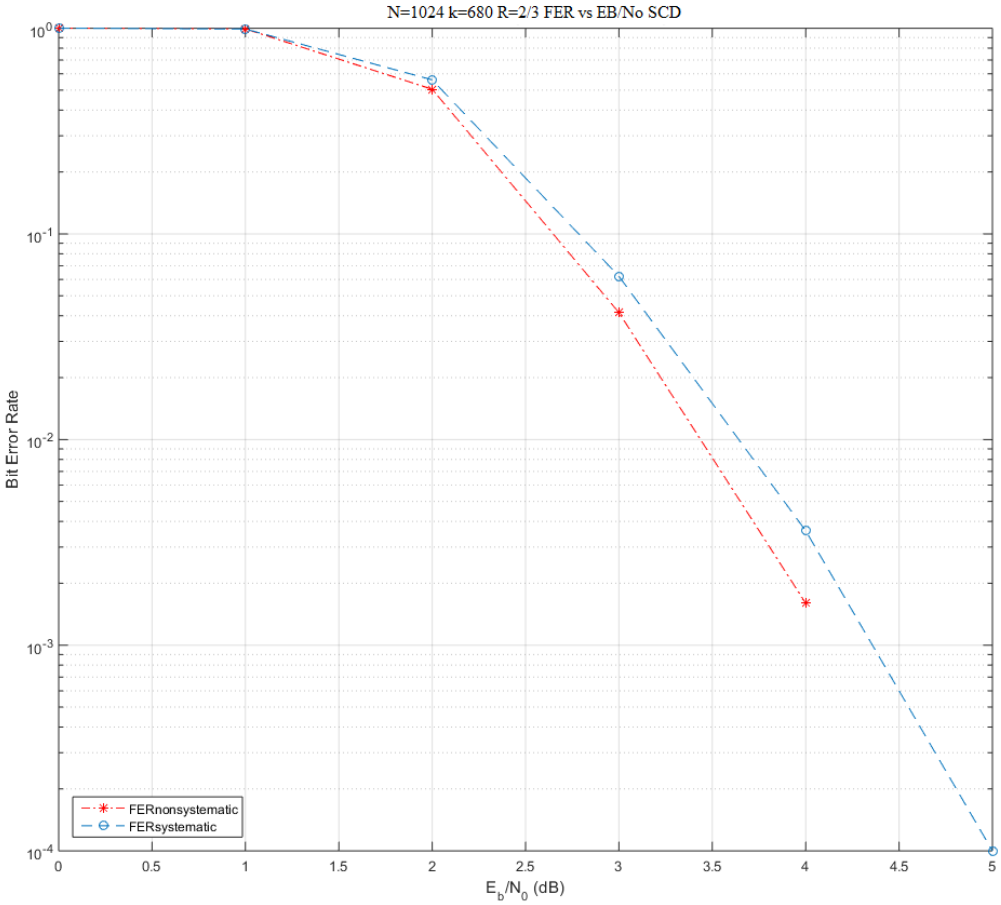


Figure 23FER Vs EB/No R=2/3 N=1024

Table 6 BER and FER recorded for N=1024 k=687 R=2/3 systematic

Eb/N0 (dB)	BER	FER
0	0.1575	1
1	0.0972	0.991
2	0.0192	0.561
3	0.0008	0.0622
4	0.00042	0.0036
5	0.000016	0.0001

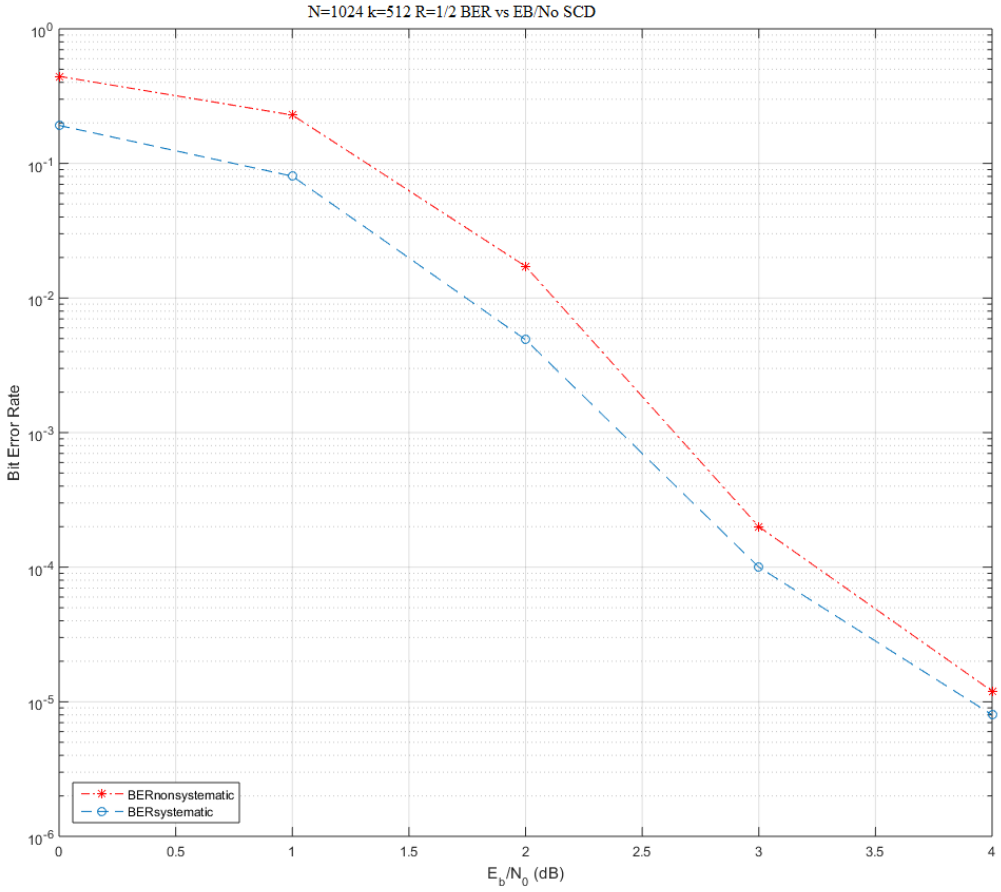


Figure 24BER Vs EB/No R=1/2 N=1024

Table 7BER and FER recorded for n=1024 k=512 R=1/2 nonsystematic

Eb/N0 (dB)	BER	FER
0	0.4438	0.9990
1	0.2288	0.7150
2	0.0172	0.0831
3	0.0002	0.0014
4	0.00036	0.0007
5	0.000060	0.0003

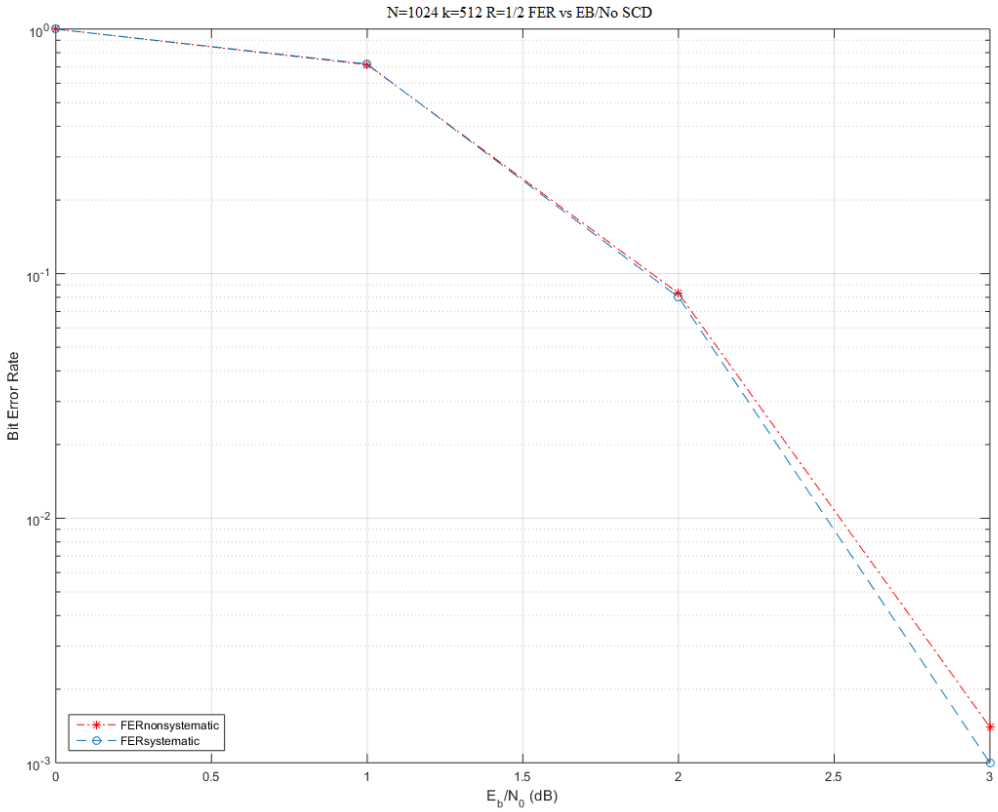


Figure 25FER Vs EB/No R=2/3 N=1024

Table 8: BER and FER recorded for n=1024 k=512 R=1/2 systematic

Eb/N0 (dB)	BER	FER
0	0.1916	0.9990
1	0.0804	0.7190
2	0.0049	0.0801
3	0.0003	0.0010
4	0.00026	0.00055
5	0	0

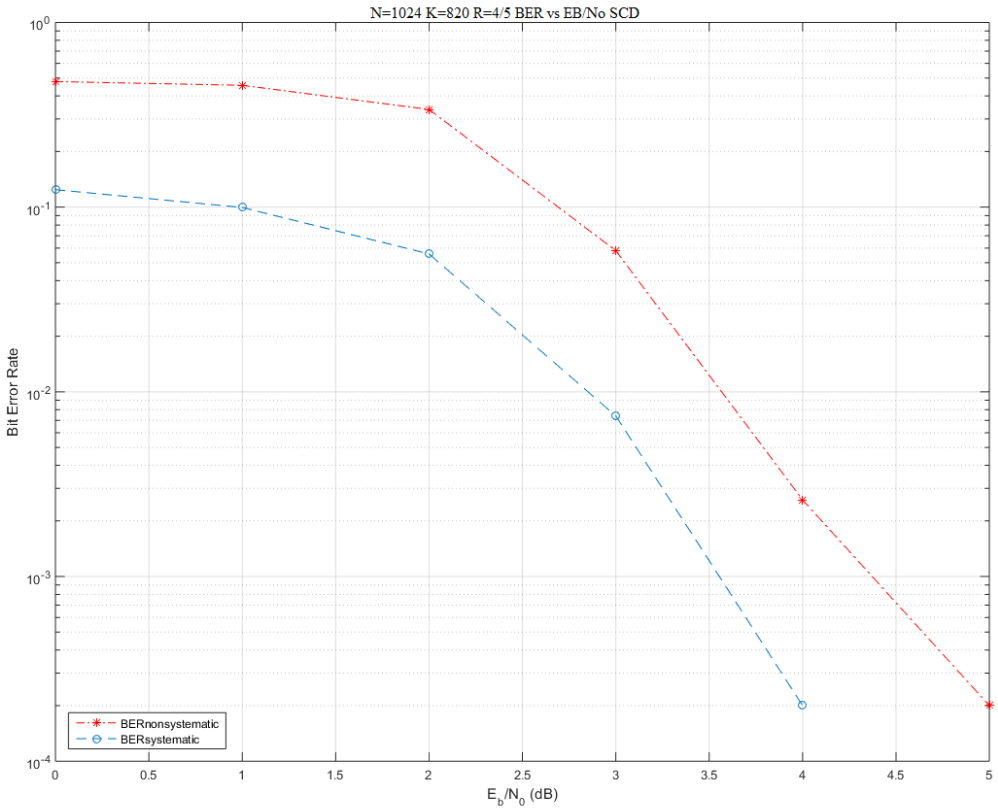


Figure 26BER Vs EB/No R=4/5 N=1024

Table 9: BER and FER recorded for n=1024 k=820 R=4/5 nonsystematic

Eb/N0 (dB)	BER	FER
0	0.4784	1.000
1	0.4566	1.000
2	0.3381	0.9730
3	0.0581	0.3680
4	0.00026	0.0393
5	0.0002	0.0031

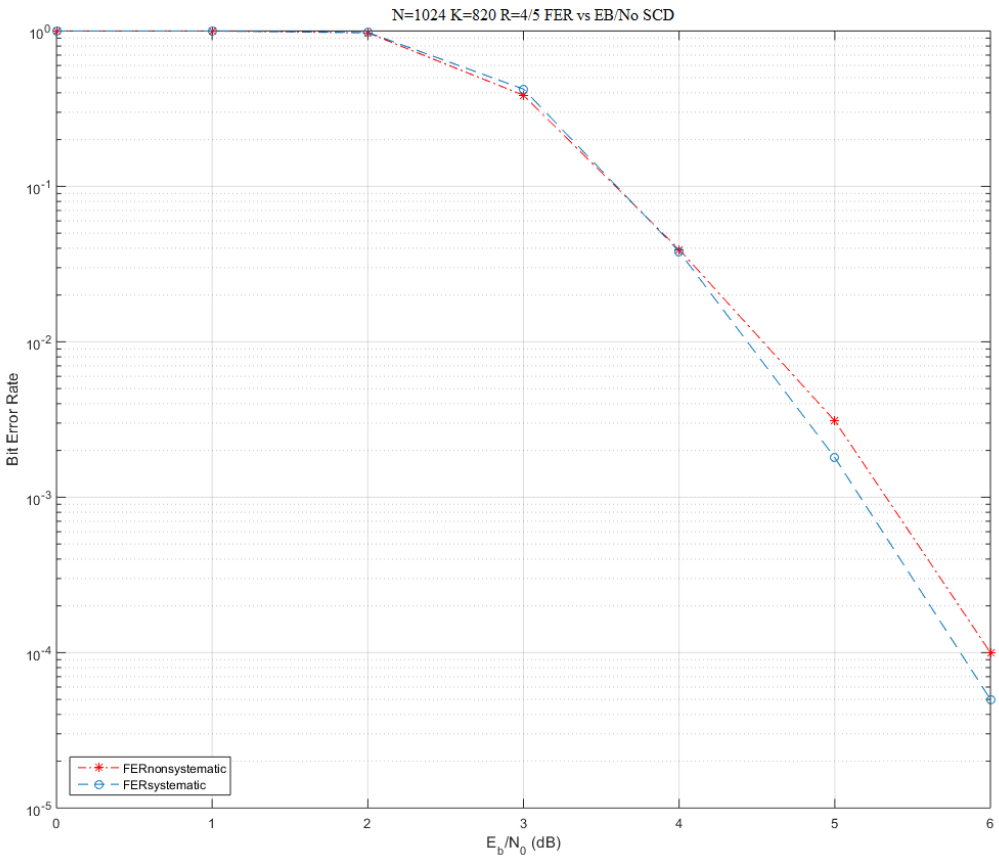


Figure 27FER Vs EB/No R=4/5 N=1024

Table 10 BER and FER recorded for n=1024 k=820 R=4/5 systematic

Eb/N0 (dB)	BER	FER
0	0.1243	1.000
1	0.0996	1.000
2	0.0558	0.9790
3	0.0074	0.4230
4	0.0018	0.0381
5	0.0002	0.0018

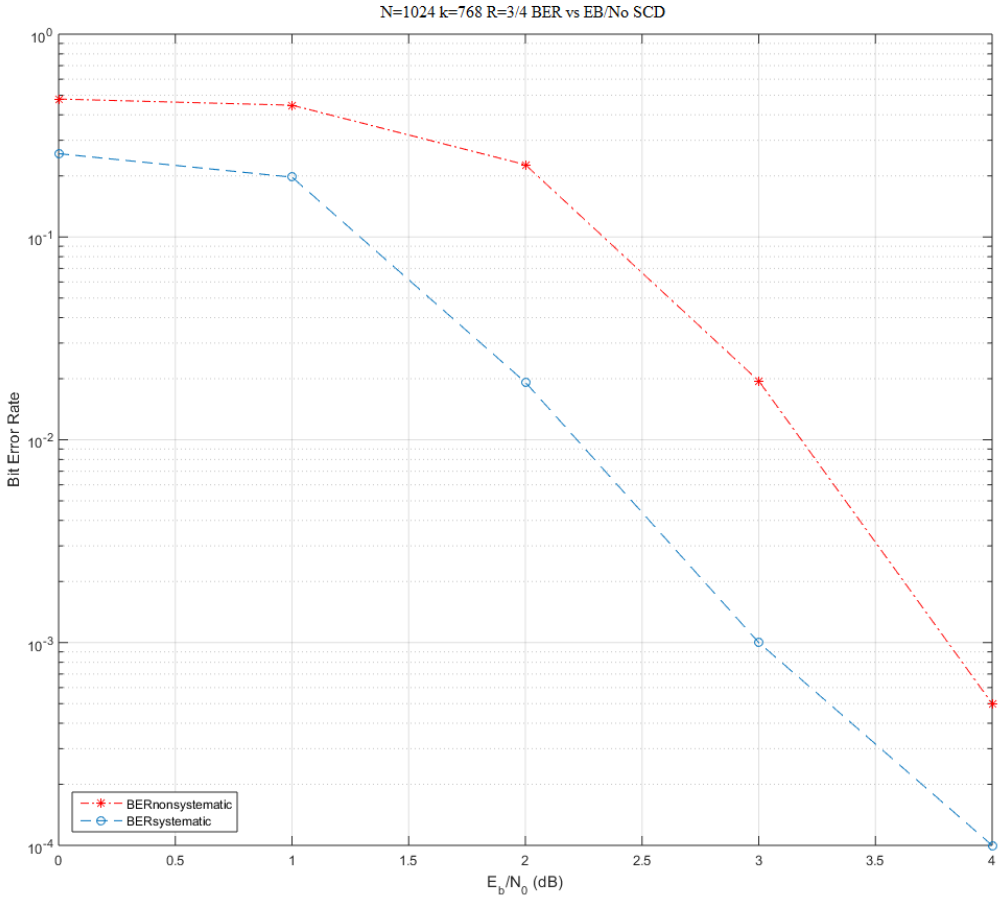


Figure 28BER Vs EB/No R=3/4 N=1024

Table 11 BER and FER recorded for N=1024 k=768 R=3/4 nonsystematic

Eb/N0 (dB)	BER	FER
0	0.5575	1
1	0.4972	0.991
2	0.3192	0.9610
3	0.0208	0.1622
4	0.00062	0.0360
5	0.00006	0.0014

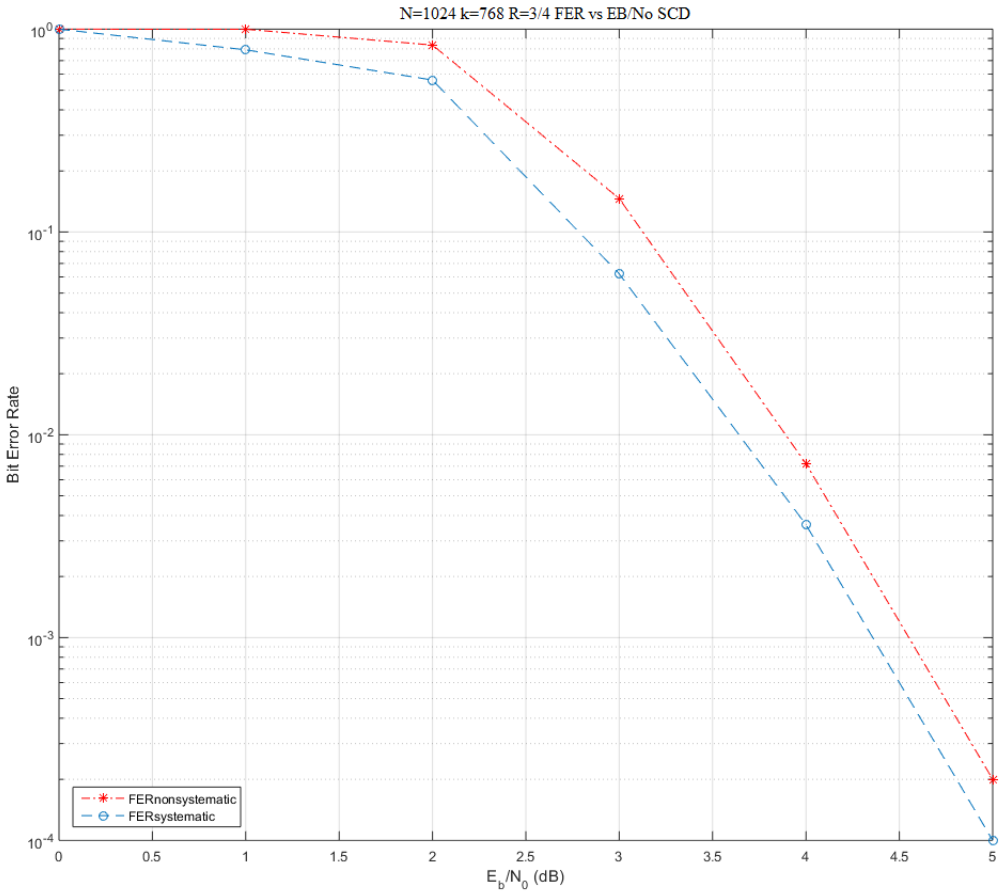


Figure 29FER Vs EB/No R=3/4 N=1024

Table 12BER and FER recorded for n=1024 k=768 R=3/4 systematic

Eb/N0 (dB)	BER	FER
0	0.4788	1
1	0.4463	0.990
2	0.2272	0.8350
3	0.0195	0.1461
4	0.0005	0.0072
5	0.00001	0.0003

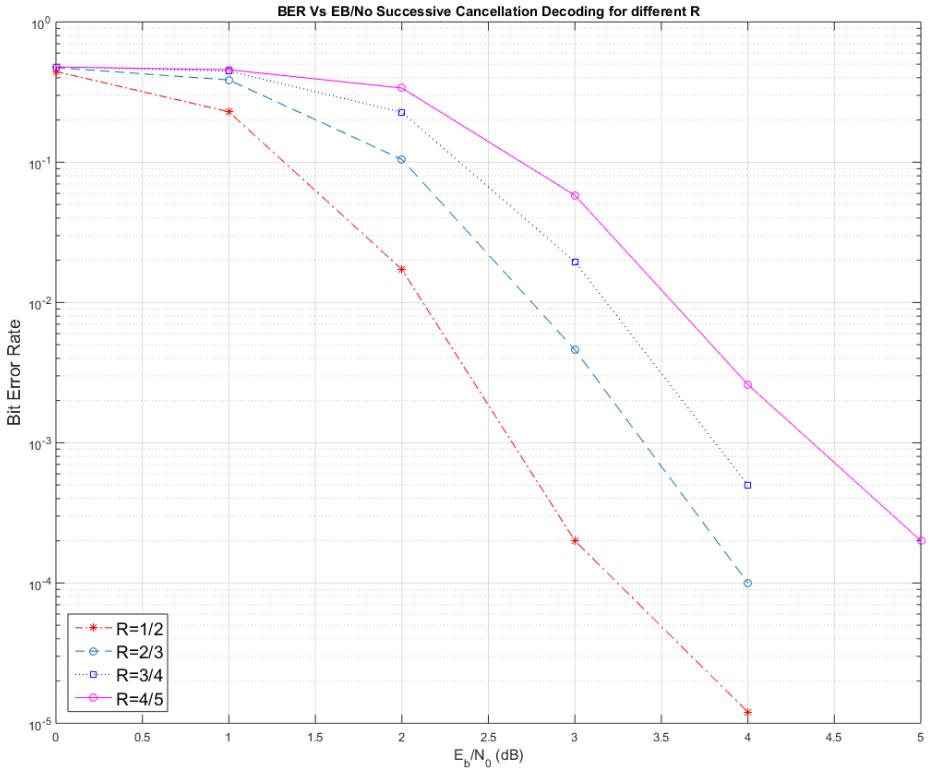


Figure 30 BER vs EB/No SCD for different R

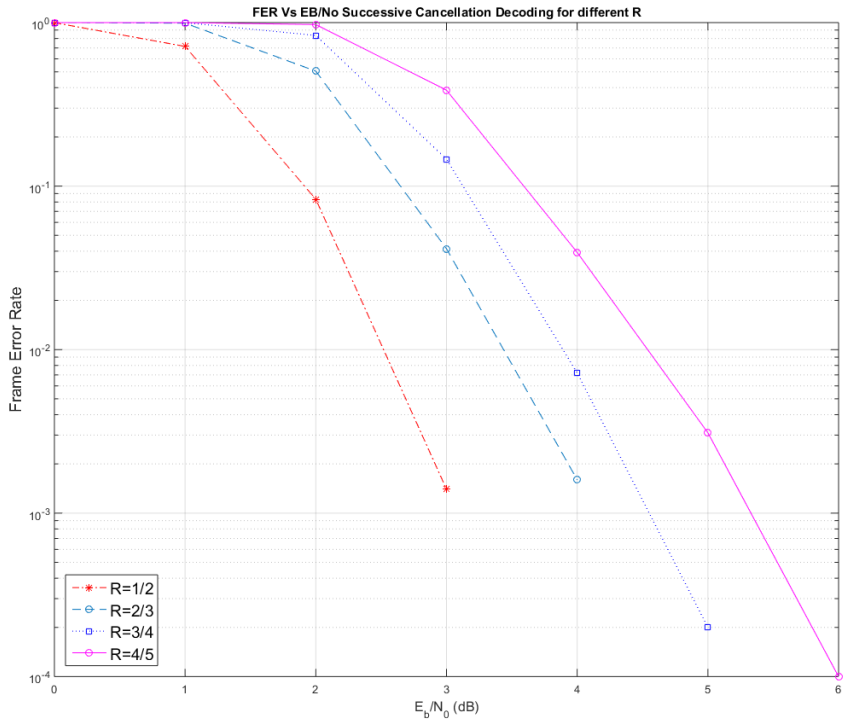


Figure 31FER Vs EB/No SCD for different R

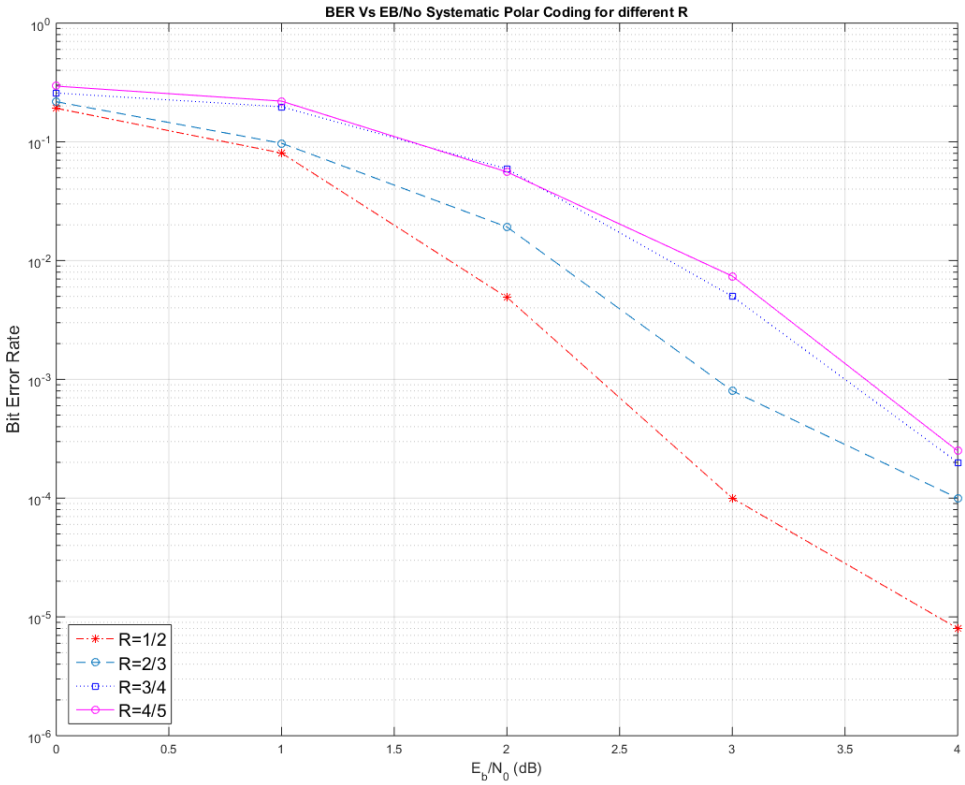


Figure 32 SCD for different values of R and N=1024

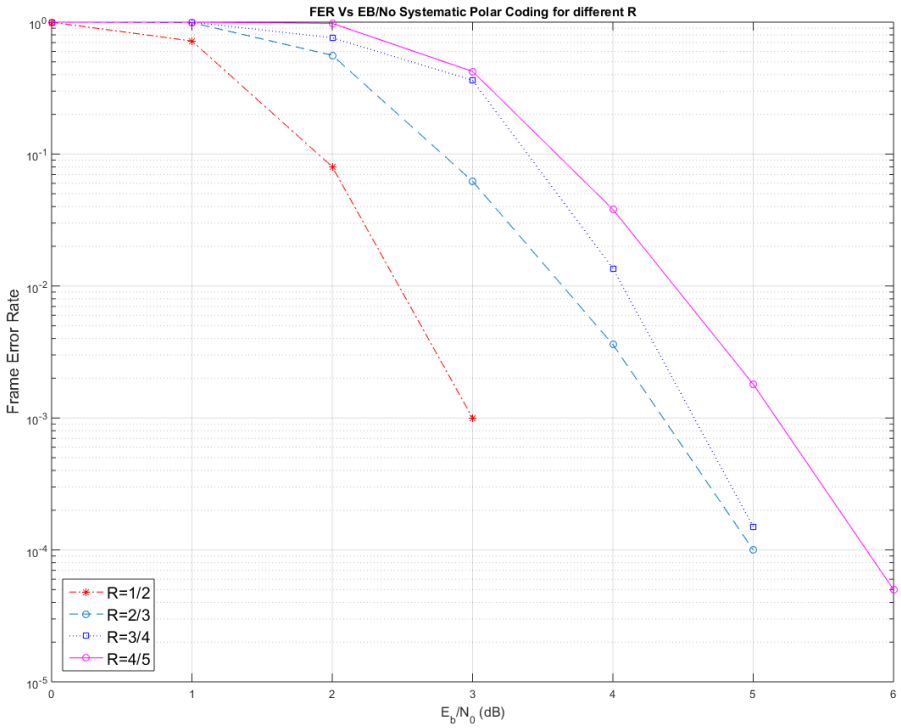


Figure 33SCD for different values of R and N=1024

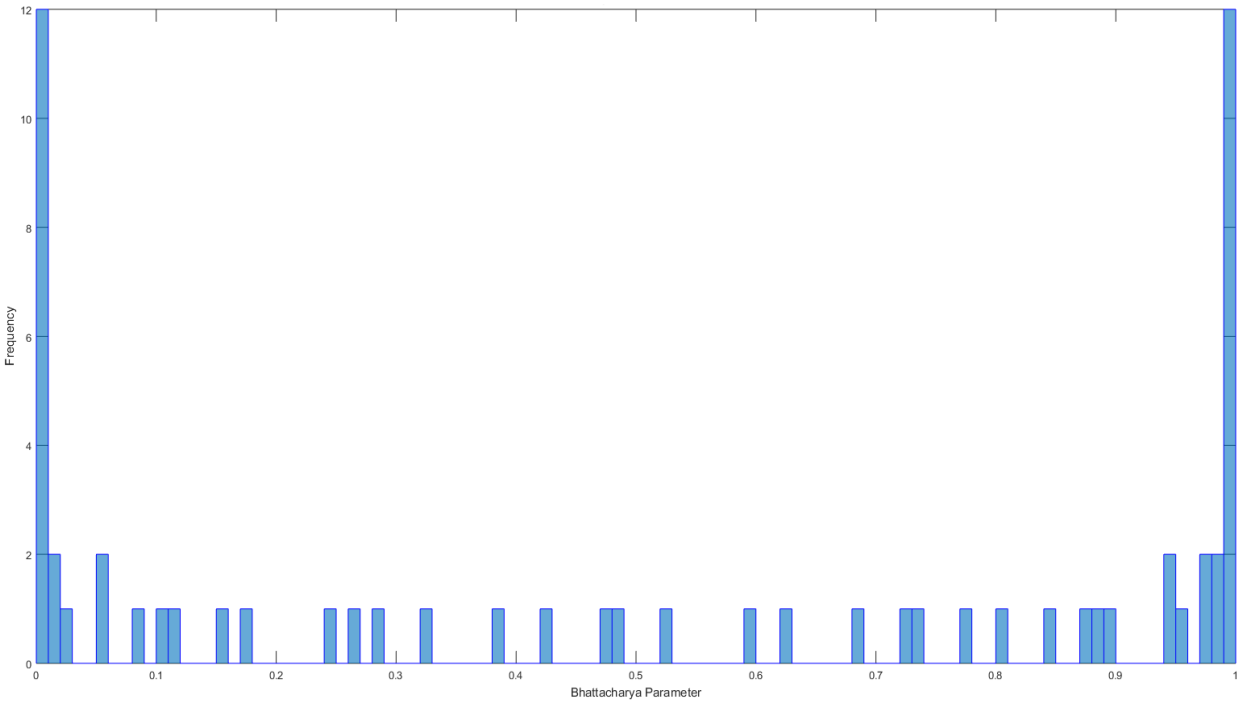


Figure 34 Histogram for Bhattacharya Parameter N=64

Table 13Histogram statistics for Bhattacharya Parameter N=64

Bhattacharya parameter	Frequency
0.95<Z(w)<=1.00	18
0.85<Z(w)<=0.95	5
0.75<Z(w)<=0.85	3
0.75<Z(w)<=0.65	3
0.65<Z(w)<=0.55	2
0.55<Z(w)<=0.45	3
0.45<Z(w)<=0.35	2
0.35<Z(w)<=0.25	3
0.25<Z(w)<=0.15	3
0.15<Z(w)<=0.05	3
0.05<Z(w)<=0.00	19

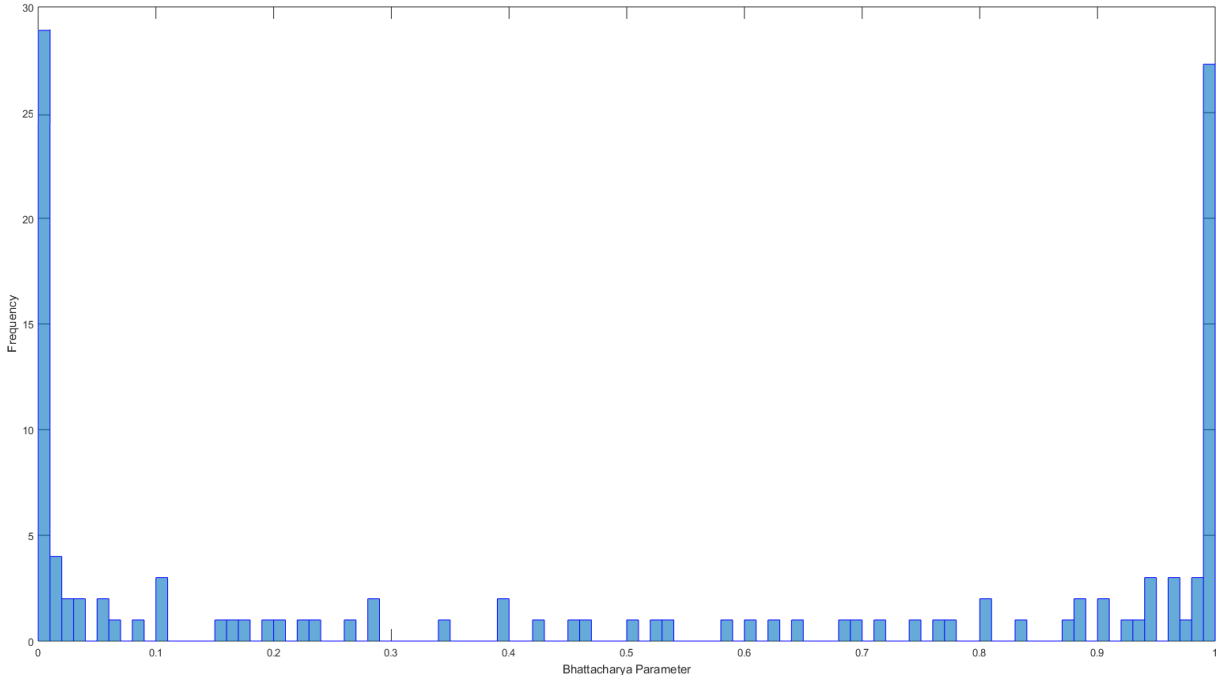


Figure 35 Histogram for Bhattacharya Parameter N=128

Table 14Histogram statistics for Bhattacharya Parameter N=128

Bhattacharya parameter	Frequency
0.95<Z(w)<=1.00	35
0.85<Z(w)<=0.95	8
0.75<Z(w)<=0.85	7
0.75<Z(w)<=0.65	5
0.65<Z(w)<=0.55	5
0.55<Z(w)<=0.45	6
0.45<Z(w)<=0.35	4
0.35<Z(w)<=0.25	4
0.25<Z(w)<=0.15	7
0.15<Z(w)<=0.05	6
0.05<Z(w)<=0.00	41

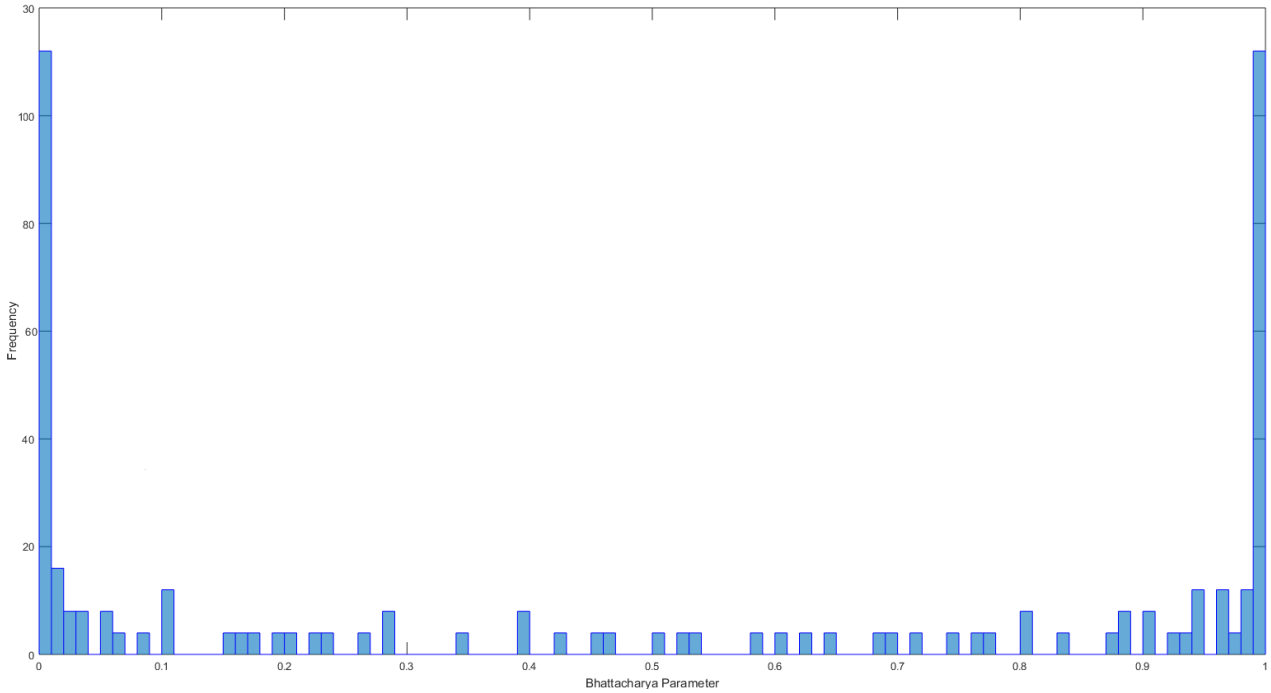


Figure 36Histogram Statistics for Bhattacharya Parameter N=256

Table 15Histogram Statistics for Bhattacharya parameter N=256

Bhattacharya parameter	Frequency
0.95<Z(w)<=1.00	110
0.85<Z(w)<=0.95	5
0.75<Z(w)<=0.85	6
0.75<Z(w)<=0.65	3
0.65<Z(w)<=0.55	4
0.55<Z(w)<=0.45	4
0.45<Z(w)<=0.35	3
0.35<Z(w)<=0.25	3
0.25<Z(w)<=0.15	4
0.15<Z(w)<=0.05	5
0.05<Z(w)<=0.00	110

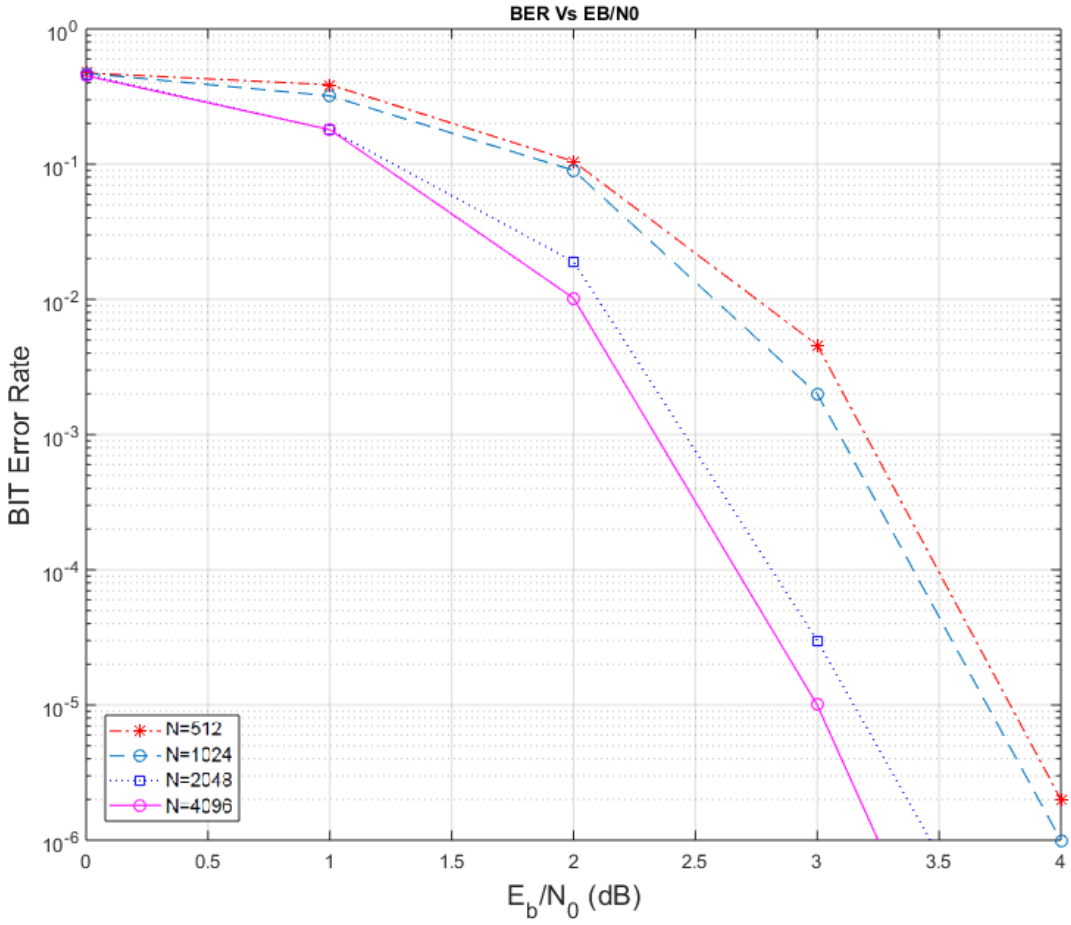


Figure 37 BER of SC decoder for different values of N

Figure 41 shows the plot of BER vs EB/No for different values of N=512,1024,2048 AND 4096 at a fixed coding rate R=1/2.It can be clearly seen that as N increases the performance improves by about 0.7 dB from N=512 to N=2048

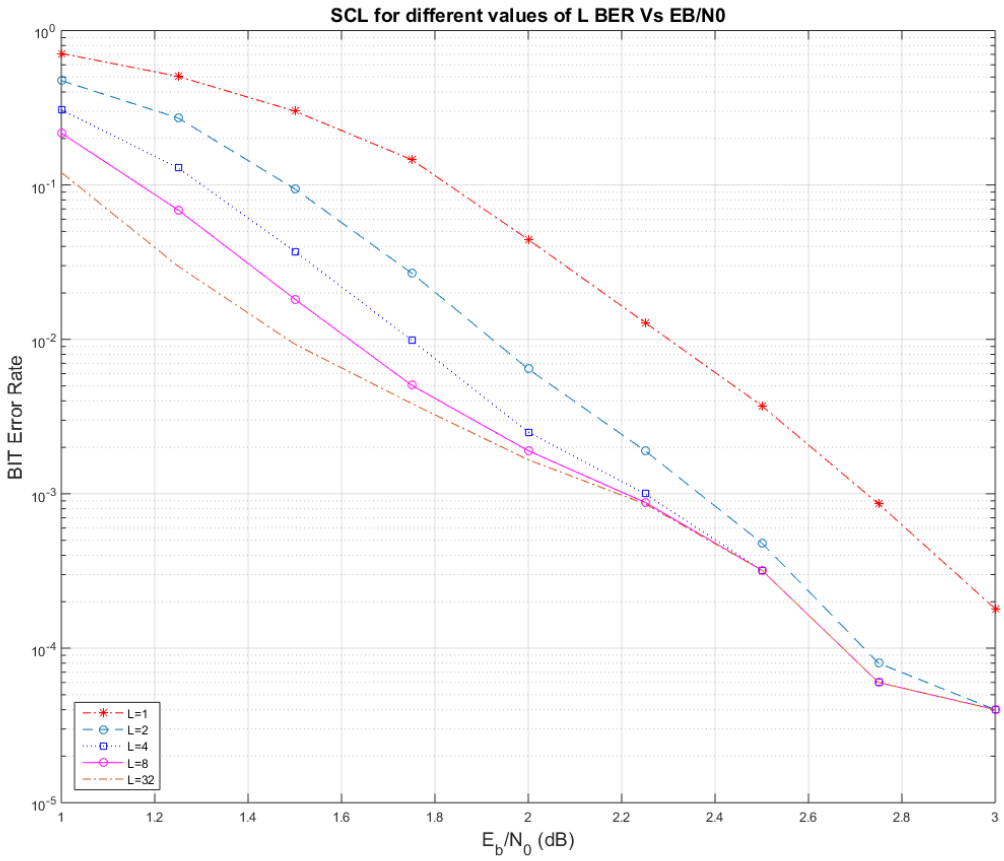


Figure 38 BER SC Vs. SCL Decoding for different L, N=1024 K=512

In case of List Decoding, the most likely codeword selected out of the L paths shows that the resulting performance is very close to that of ML decoder even for moderate values of L
A gain of 0.5 dB can be seen for BER of 10^{-5} with respect to the successive cancellation
Based on 100k runs for L=1 to 32 .L=1 is the basic SCD Performance approaches 2.5 dB for L=32 with 500k runs

Table 16BER values for SCL decoding

EB/No	BIT ERROR RATES FOR DIFFERENT L				
1	0.711268	0.474178	0.306991	0.216738	0.120669
1.25	0.505	0.271505	0.128827	0.068942	0.029758
1.5	0.300595	0.094216	0.037132	0.018215	0.009316
1.75	0.145954	0.026912	0.009894	0.005048	0.003844
2	0.044298	0.006427	0.002507	0.0019	0.00166
2.25	0.012847	0.0019	0.001	0.00088	0.00086
2.5	0.003719	0.00048	0.00032	0.00032	0.00032
2.75	0.00086	0.00008	0.00006	0.00006	0.00006
3	0.00018	0.00004	0.00004	0.00004	0.00004

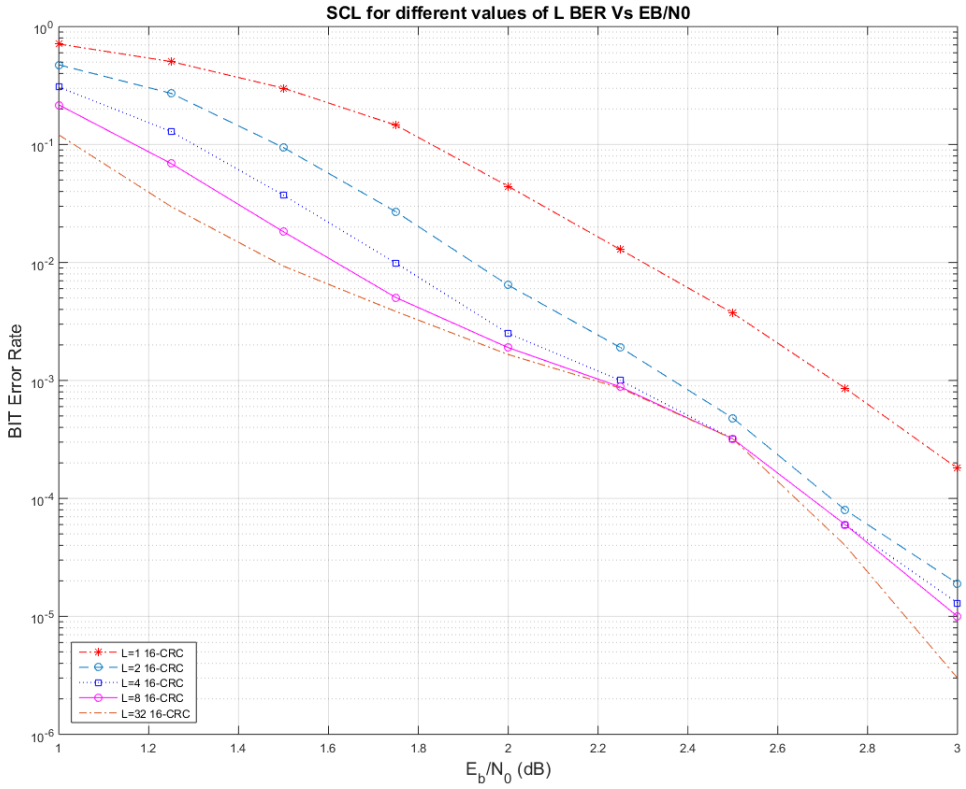


Figure 39CA-SCL for different values of L 16-CRC

In case of CRC Aided List Decoding, the most likely codeword selected out of the L paths shows that the resulting performance is very close to that of ML decoder even for moderate values of L

A gain of 0.5 dB can be seen for BER of 10^{-6} with respect to the successive cancellation Based on 100k runs for L=1 to 32. L=1 with 16-CRC is the basic SCD Performance approaches 2.5 dB for L=32 with 500k runs

Table 17BER values for CA SCL decoding with 16-CRC

EB/No	BIT ERROR RATES FOR DIFFERENT L WITH 16-CRC				
1	0.711268	0.474178	0.306991	0.216738	0.120669
1.25	0.505	0.271505	0.128827	0.068942	0.029758
1.5	0.300595	0.094216	0.037132	0.018215	0.009316
1.75	0.145954	0.026912	0.009894	0.005048	0.003844
2	0.044298	0.006427	0.002507	0.0019	0.00166
2.25	0.012847	0.0019	0.001	0.00088	0.00086
2.5	0.003719	0.00048	0.00032	0.00032	0.00032
2.75	0.00086	0.00008	0.00006	0.00006	0.00004
3	0.00018	0.000019	0.000013	0.00001	0.000003

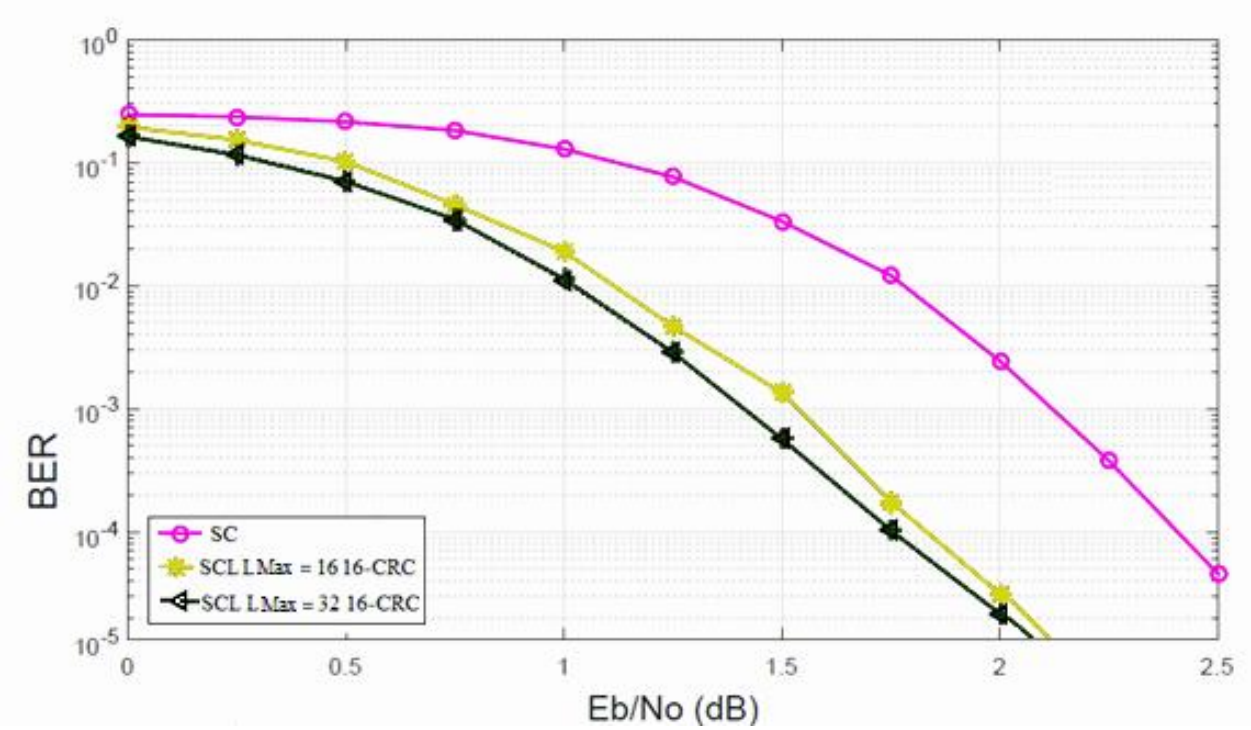


Figure 40 BER for SC vs Adaptive Decoder for N=1024 k=512

- Adaptive Decoders for basic SC and SCL-adaptive with a 16-bit CRC can be concluded that adaptive decoders outperform the basic SC and SCL decoders with about 1 dB crossing BER of 10^{-6}.
- It can be seen that as the List Size Increases in case of adaptive decoders it outperforms the basic SC and SCL decoders for LMax=16 and 32.

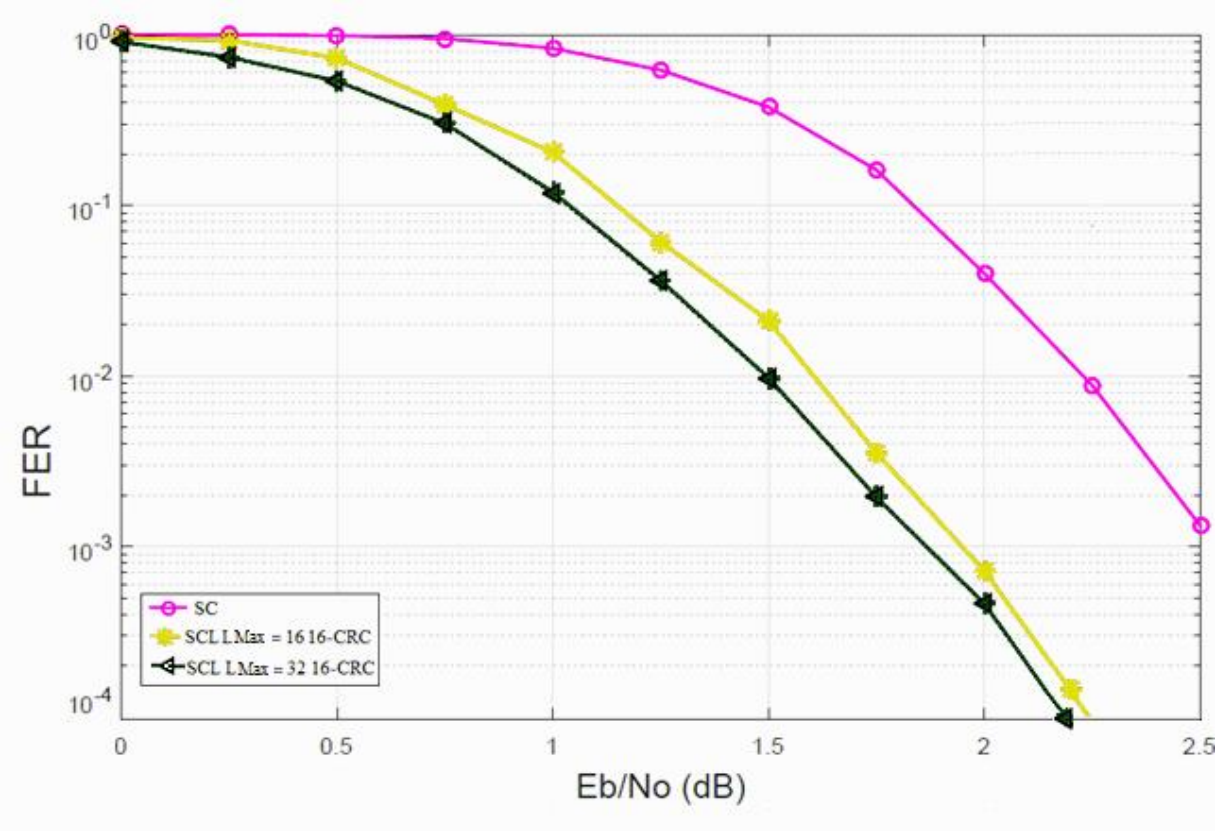


Figure 41FER for SC vs Adaptive decoder for N=1024 k=512

- Adaptive Decoders for basic SC and SCL-adaptive with a 16-bit CRC can be concluded that adaptive decoders outperform the basic SC and SCL decoders with about 1 dB crossing FER of 10^{-6}.
- It can be seen that as the List Size Increases in case of adaptive decoders it outperforms the basic SC and SCL decoders for LMax=16 and 32.

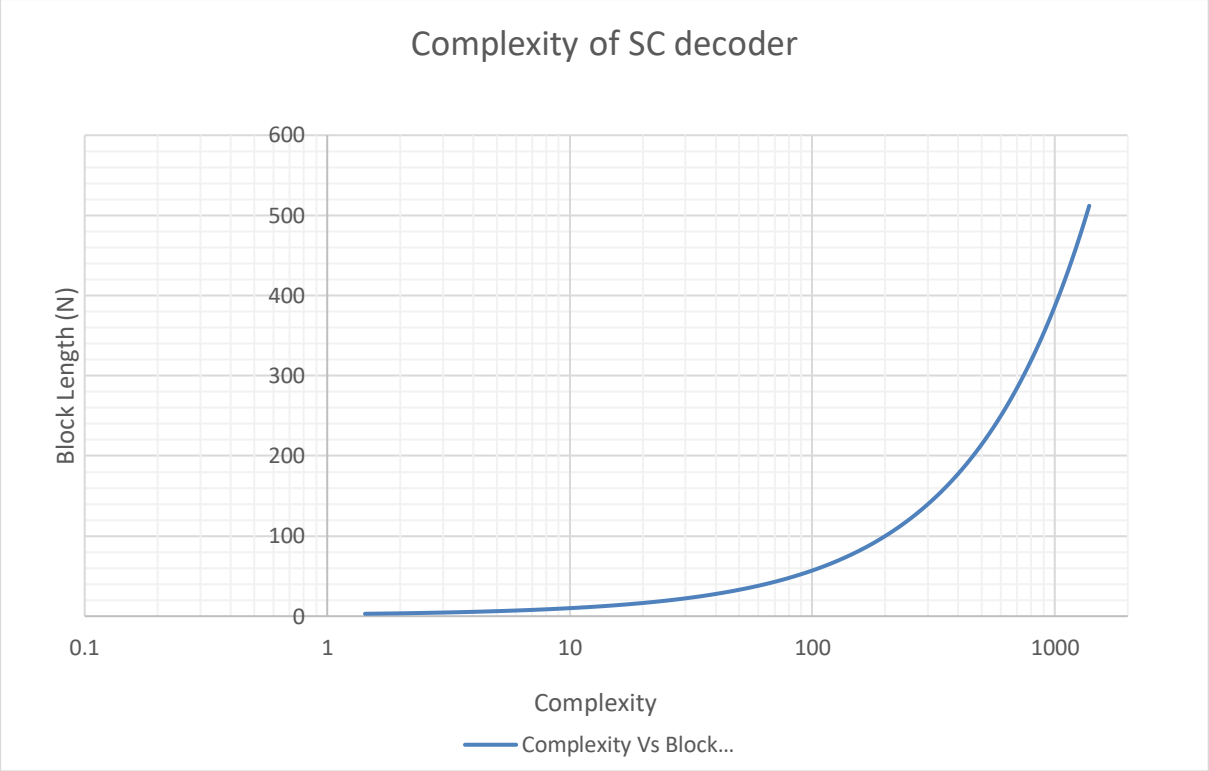


Figure 42Complexity of SC decoder

Figure 43Complexity of SC-L decoder for different values of L

5.2 Conclusion

Successive Cancellation decoder performs well for large block lengths and can achieve an Eb/No of 4.25 dB for coding rate of 0.8 and 3.25 dB for coding rate of 1/2 which is about a 1db difference for BER up to 10^{-5}. When the coding rate is fixed to R=1/2 and when the block length is increased for a fixed K, The performances improves from 4.8 dB to 3.5dB which is about 1.3dB gain at BER OF 10^{-6}. SC decoder compared to CTC a performance degradation can be seen as CTC outperforms Basic SC decoder by about 0.8 dB with coding rates of 105 and 0.75 for different values of N and K. Since FER can be derived from the BER, BER performance of CTC is best compared to Polar Codes The average number of Add-Compare-Select operations performed decrease as the number of bit errors decreases with respect to using the IT++ suite. This is the decoders computational complexity and can be seen for different coding rates. The results for the polarization Martingale indicate the number of channels that are distributed between the upper and lower bounds of the Bhattacharya parameter. As the block length increases the channels start settling either at 0 or 1(δ or 1-δ) which are shown through the histogram statistics.

The most likely codeword selected out of the L paths shows that the resulting performance is very close to that of ML decoder even for moderate values of L. A gain of 0.5 dB can be seen for BER of 10^{-6} with respect to the successive cancellation List decoder against the basic successive cancellation decoder. Also as the List size L keeps increasing, the performance improves at low SNRs, but eventually the performances of all the List sizes converge at the same Eb/No an has a gain of 0.5 dB than the basic successive cancellation decoder. CRC aided SCL decoder has a performance improvement of 0.5 dB over the SCL decoder as the CRC schemes widely used in 3GPP schemes are known to improve the resulting performance than the basic schemes. Adaptive Decoders for basic SC and SCL-adaptive with a 16-bit CRC can be concluded that adaptive decoders outperform the basic SC and SCL decoders with about 1 dB crossing BER of 10^{-6}.

Chapter 6 Future Work

There is a lot of research going on in the field of Channel Coding to improve the data rates in many of the Radio Access Technologies including the upcoming 5G technology. Basic Successive cancellation decoders degrade in the performance in small block lengths and this is an area which can be worked to improve even at small block lengths. Many other procedures can aid the SC or the SCL decoders to further improve the performance like the CRC aided ones. Adaptive decoders almost near to the information theoretic capacity and they can be further improved to achieve a break through in the field of channel coding and wireless communication. Complexity and Latency are the two major areas where polar codes have a scope for improvement

Polar Codes have a lot of scope for improvement especially in the area of Decoding.Decoder complexities are a major area of scope for research and development.The other areas that can be explored are the effect of Bhattacharya Parameter Z(W), the symmetric Channel Capacity I(W), Reliability of channels and the effect of Polarization on the reliability etc. Decoding methods is a hot topic currently undertaken by many researchers for improving them.In doing so many methods such as the List Decoding, Belid=ef Propogation decoding have come into existence.Improving the BERs effortlessy can be done with the help of decoders whose complexity and Latency have a tradeoff in terms of the number of XOR operations that are involved in this process.

Appendix A

Software Test Configurations

The software test suite that I used for experimental purposes is IT++ and Polar Code 1.1.0. PolarCode 1.1.0 provides functions for polar code training, encoding and decoding. Like turbo and LDPC codes, polar codes facilitate near-capacity operation. However, unlike turbo and LDPC codes, polar codes do not require an iterative decoder. The operating system that I used to carry out the testing is Linux Ubuntu 14.04 LTS. PolarCode 1.1.0 is tested with IT++ 4.0.2 and g++ 4.2.1. The version of GCC compiler that I used to carry out testing is gcc version 5.4.0.

This software suite is free software. It can have redistributed and/or modified under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful for purposed of channel coding experiments. See the GNU General Public License for more details.

The channel that we make use of here is the Binary discrete memoryless channel(B-DMC) AWGN channel and we transmit data over the BPSK modulated AWGN channel having specified SNR. Simulations will be done using IT++. IT++ is a C++ library of classes and functions for linear algebra, numerical optimization, signal processing, communications, and statistics. It is currently gaining usage amongst users and researchers in the area of information theory. The main reason for the use of IT++ is simply because of the speed of compiling and generating output for your program.10000 bursts are run for each simulation

Running Simulations

Example1:/mainTraining=1 N=1024 ThresholdMI=0.999 SNR=0 FrameCount=10000\encode

This will train the information set \f$\mathcal{A}\f$ and automatically choose an appropriate information block length \f$K\f$ for a polar code having an encoded block length of \f$N\f$ used for transmission over a BPSK-modulated AWGN channel having the specified SNR. During this training process, the mutual information offered by each of the \f$N\f$ channels will be compared with ThresholdMI. If the mutual information is greater than ThresholdMI, then the channel will be admitted to the information set \f$\mathcal{A}\f$ and \f$K\f$ will be incremented. The length and accuracy of the simulation depends on the value you use for FrameLength. The resultant parameters \f$(N,K,\mathcal{A})\f$ will be written into a configuration file having a filename of the form polar_N_K_SNR.

Example 2: ./main Training=1 N=1024 K=512 SNR=0 FrameCount=10000\endcode

This will train the information set \f$\mathcal{A}\f$ of an \f$(N,K)\f$ polar code for transmission over a BPSK-modulated AWGN channel having the specified SNR. The length and accuracy of the simulation depends on the value you use for FrameLength. The resultant parameters \f$(N,K,\mathcal{A})\f$ will be written into a configuration file having a filename of the form polar_N_K_SNR.

Example 3: ./main Training=1 N=1024 K=512 SNR=0 FrameCount=10000\endcode

This will read the polar code parameters \f$(N,K,\mathcal{A})\f$ from the specified configuration file and simulate its transmission over a BPSK-modulated AWGN channel having the SNRs [SNRStart, SNRStart+SNRDelta, SNRStart+2SNRDelta, SNRStart+3SNRDelta, ...]. The simulation of each SNR will continue until at least TargetBitCount number of bits have been transmitted and TargetErrorCount number of errors have been observed. The results are written into a file having a filename of the form ConfigFile_SNRStart_ber. For each SNR, the results file provides the number of transmitted bits, the number of observed errors, the BER and the average number of Add, Compare and Select (ACS) operations performed per bit.

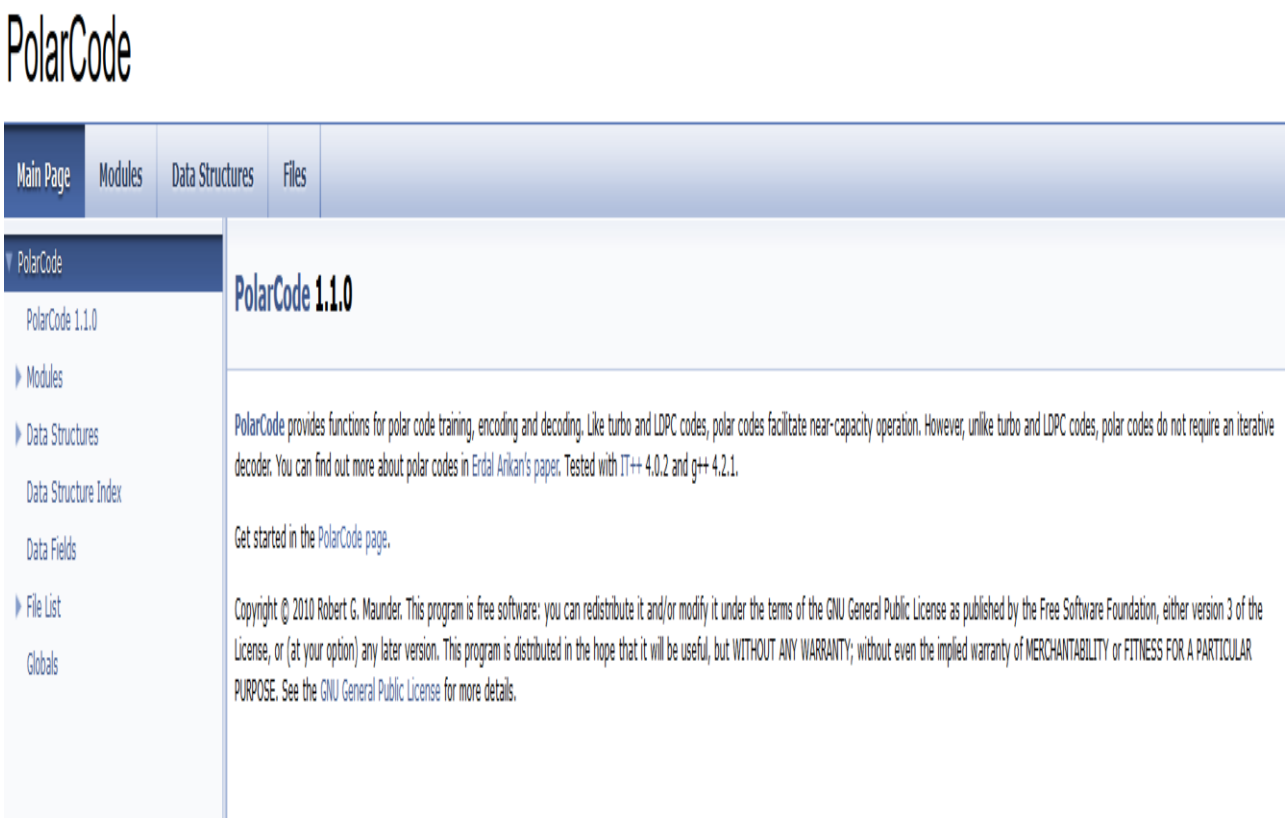


Figure 44Polar Codes Main Page (a)

Figure 45Polar Codes Main Page (b)

Figure 46: Modules in Software suite of Polar Code1.1.0

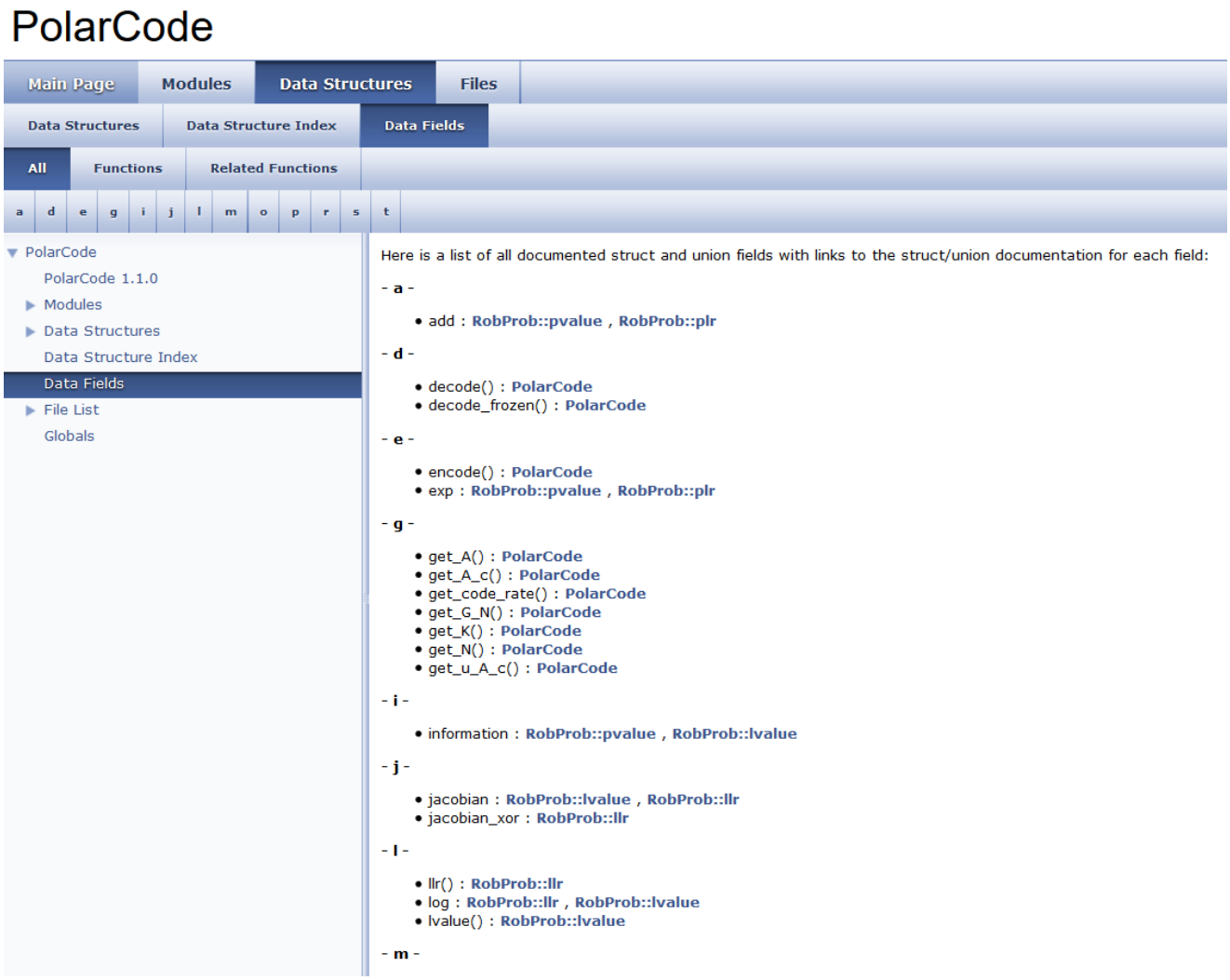


Figure 47 Data Fields in Software Suite of PolarCodes1.1.0

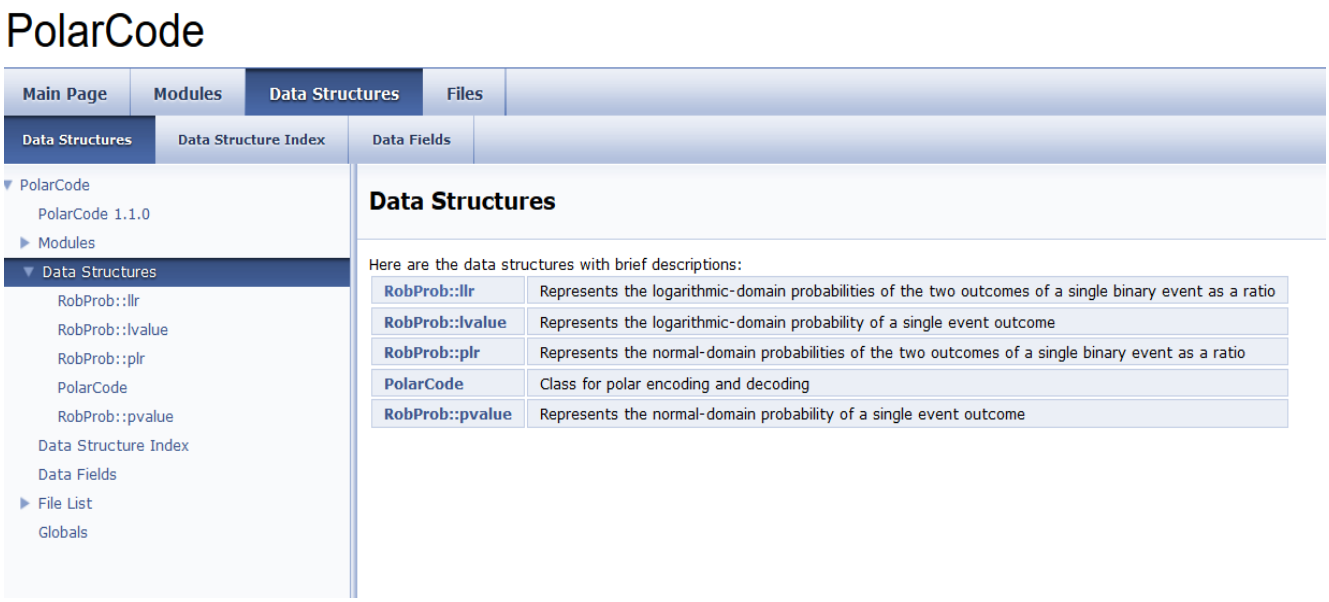


Figure 48 Data Structures in Software suite PolarCode1.1.0

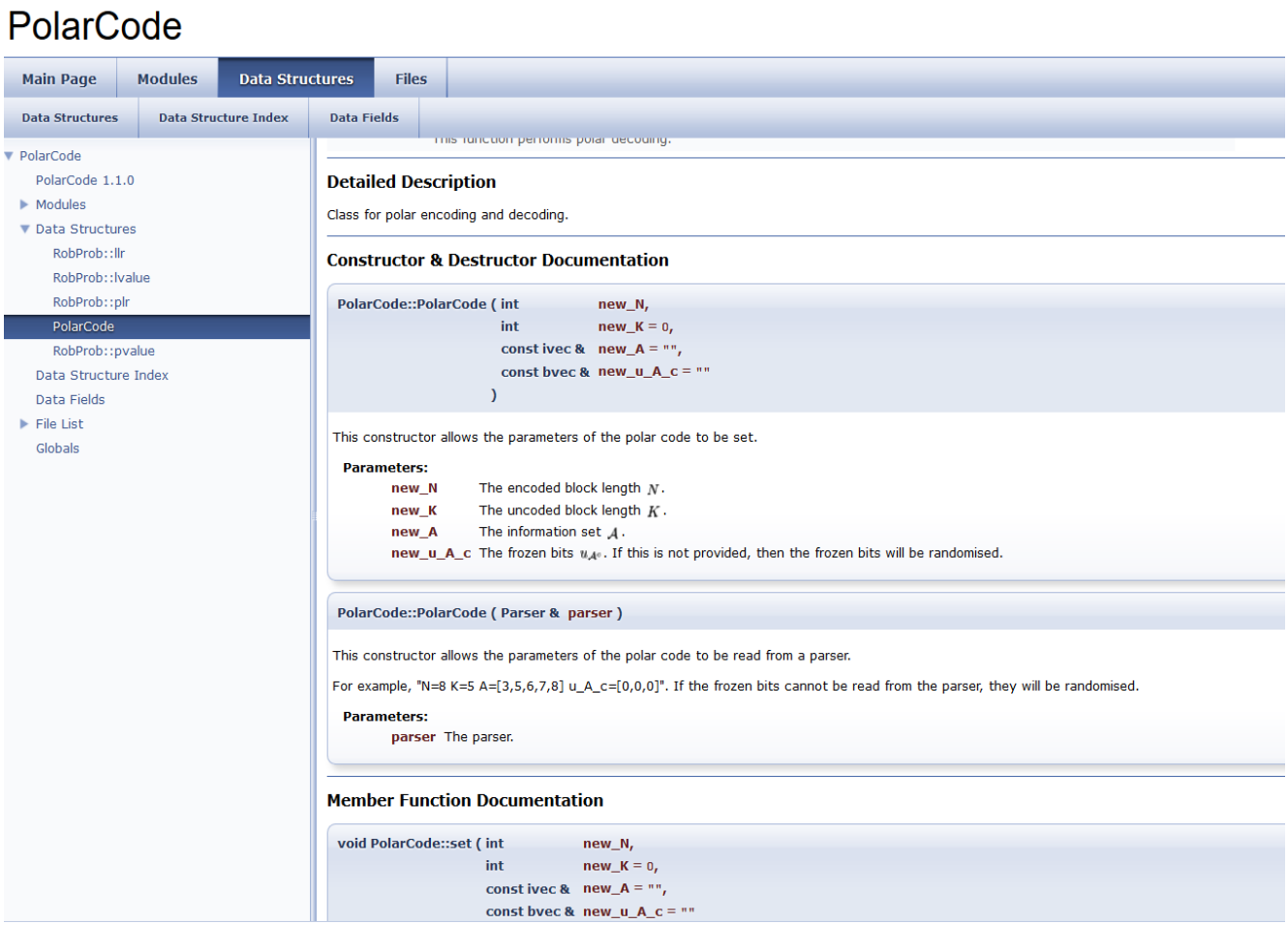


Figure 49:Discription of Classes for Polar Encoding and Decoding

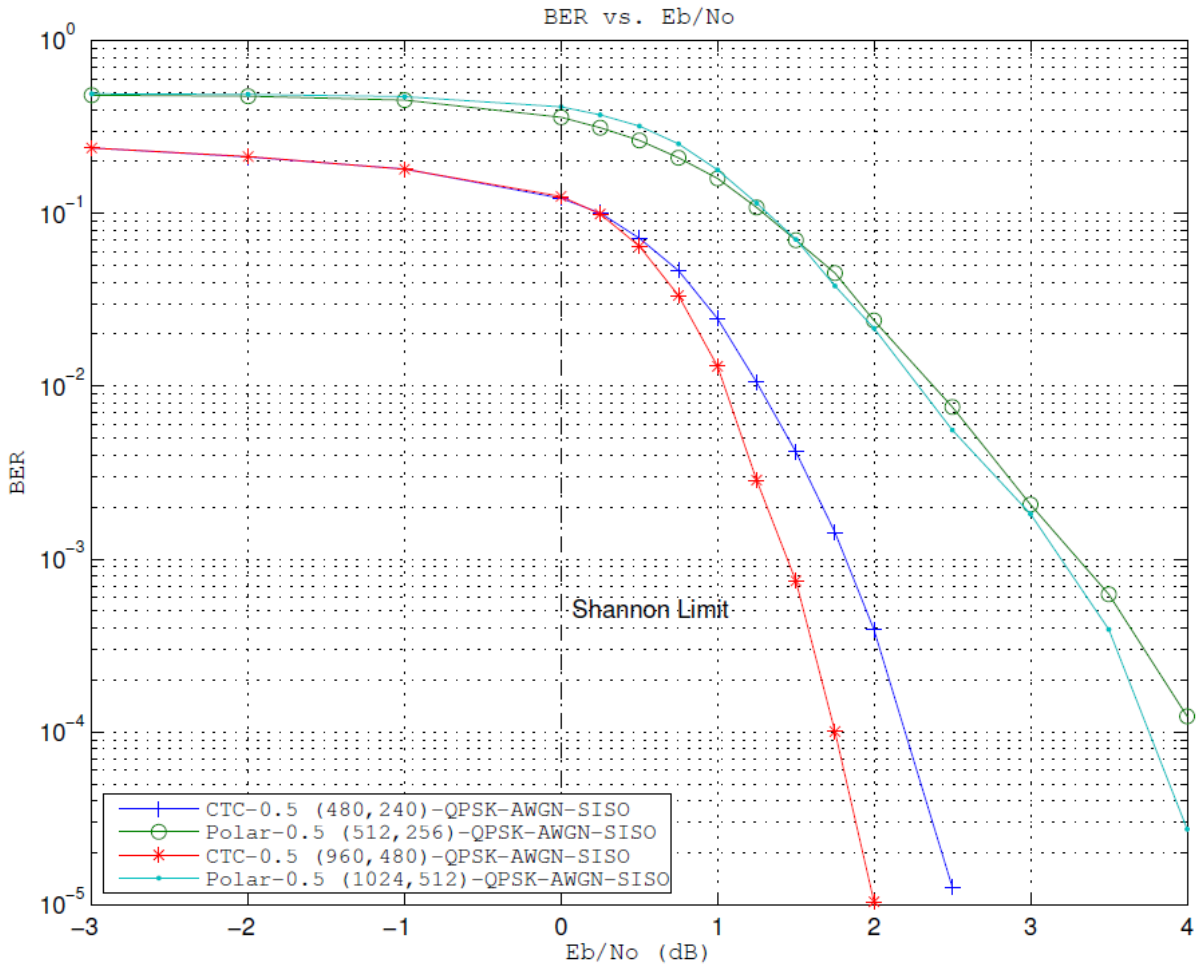


Figure 50 BER Curve for Turbo and Polar Code R=1/2[45]

Table 18Parameters for CTC vs Polar Codes R=1/2[45]

Code Type	Code Rate	Information Length (bits)	Coded Block Length (bits)
CTC	1/2	240	480
CTC	1/2	480	960
Polar	1/2	256	512
Polar	1/2	512	1024

- SC decoder compared to CTC a performance degradation can be seen as CTC outperforms Basic SC decoder by about 0.8 dB with coding rates of 0.5 and different block lengths
- This tells us that basic SC decoder cannot be a good fit to be used in the upcoming 5G systems

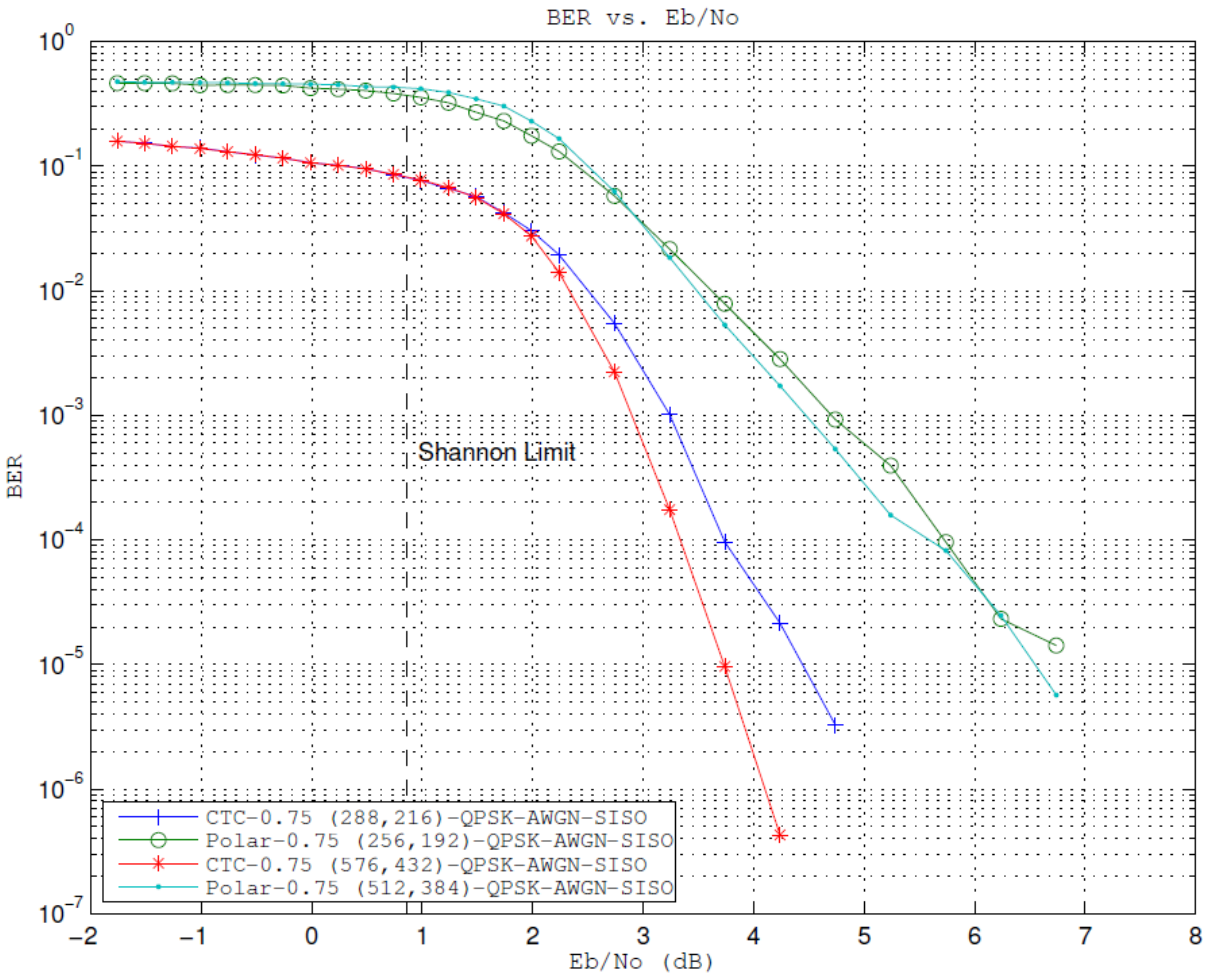


Figure 51 BER Curve for Turbo and Polar Code R=3/4[45]

Table 19Table for CTC vs Polar Codes R=3/4[45]

Code Type	Code Rate	Information Length (bits)	Coded Block Length (bits)
CTC	3/4	216	288
CTC	3/4	432	576
Polar	3/4	192	256
Polar	3/4	384	512

- SC decoder compared to CTC a performance degradation can be seen as CTC outperforms Basic SC decoder by about 0.8 dB with coding rates of 0.75 and different block lengths
- This tells us that basic SC decoder cannot be a good fit to be used in the upcoming 5G systems

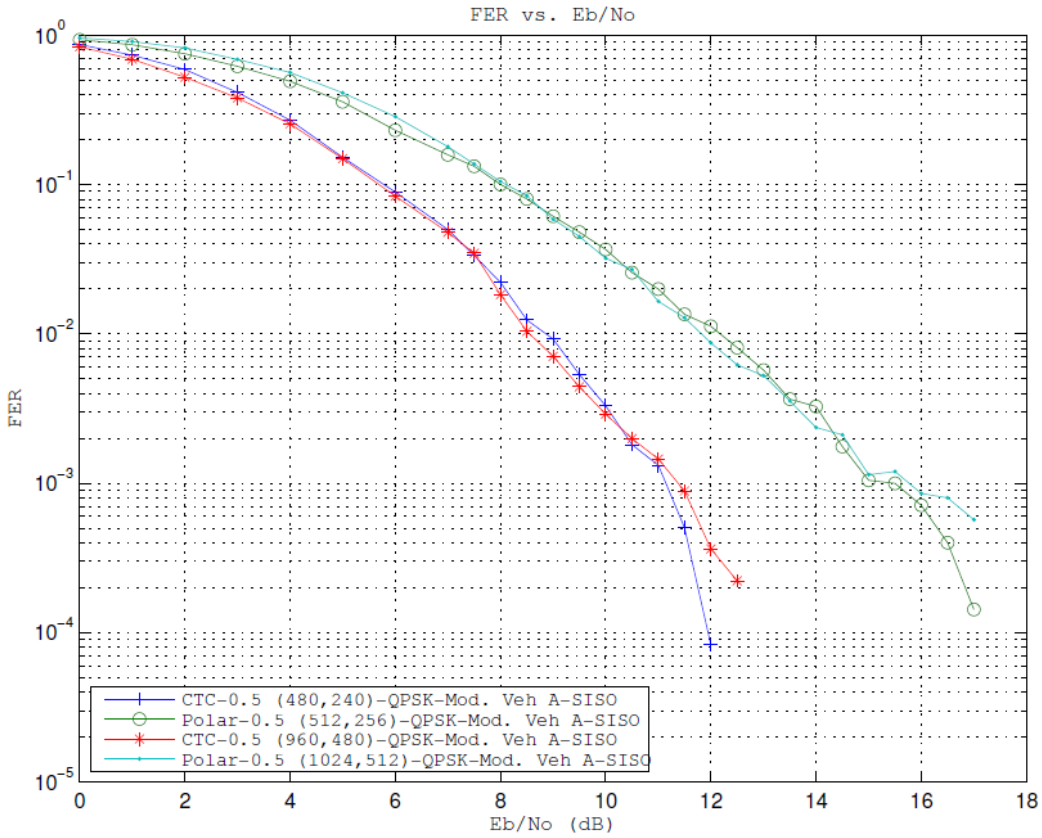


Figure 52 FER for Turbo and Polar Code R=1/2[45]

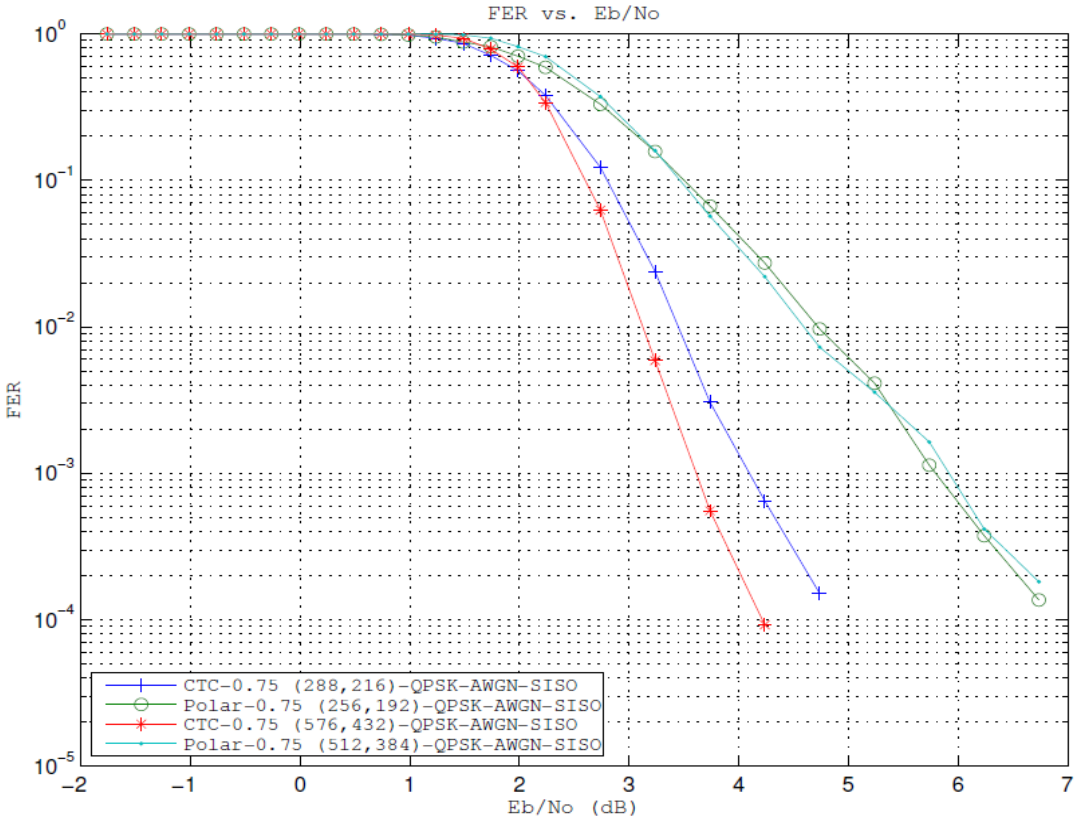


Figure 53FER for Turbo and Polar Code R=3/4[45]

```
/*
 PolarCode provides functions for polar code training, encoding and decoder.
 Copyright (C) 2010  Robert G. Maunder

 This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the Li
 This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU Ge
 The GNU General Public License can be seen at http://www.gnu.org/licenses/.
*/

/// \file main.cpp
/// \author Robert G Maunder
/// \date 06/08/2010
/// \version 1.1.0
/// \brief Trains polar codes and simulates their operation.

#include <cfloat>
#include "itpp/itcomm.h"
#include "PolarCode.h"

/// \ingroup polar
/// Trains polar codes and simulates their operation.
/// Parameters are read from the command line and/or a configuration file. For example...

/// \code./executable Training=1 N=1024 K=512 SNR=0 FrameCount=10000 \endcode
/// This will train the information set \f$\mathcal{A}\f$ of an \f$(N,K)\f$ polar code for transmission over a BPSK-modulated AWGN channel having the specified SNR. The length and accuracy of the

/// \code./executable Training=1 N=1024 ThresholdMT=0.999 SNR=0 FrameCount=10000 \endcode
/// This will train the information set \f$\mathcal{A}\f$ and automatically choose an appropriate information block length \f$K\f$ for a polar code having an encoded block length of \f$N\f$ used

/// \code./executable ConfigFile=polar_1024_512_+0.00_SNRStart=-10_SNRDelta=0.5_TargetBitCount=5120_TargetErrorCount=100\endcode
/// This will read the polar code parameters \f$(N,K,\mathcal{A})\f$ from the specified configuration file and simulate its transmission over a BPSK-modulated AWGN channel having the SNRs [SNRSta
int main(int argc, char* argv[])
{
    // Create a parser and read parameters from the command line and the configuration file, if it exists
    Parser parser;
    parser.init(argc, argv);

    // Should the polar code be trained or characterised?
    bool training = 0;
```

Figure 54:Code Snippet depicting 10000 Bursts

References

[1] Arikan, Erdal. "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels." *IEEE Transactions on Information Theory* 55, no. 7 (2009): 3051-3073.

[2] IIT Kharagpur Lecture Series on Digital Communication by Department of Electrical Engineering.http://nptel.ac.in/courses/Webcoursecontents/IIT%20Kharagpur/Digi%20Comm/pdf-m-6/m6l33.pdf

[3] Arikan, Erdal. "Systematic polar coding." *IEEE communications letters* 15, no. 8 (2011): 860-862.

[4] Tal, Ido, and Alexander Vardy. "List decoding of polar codes." In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pp. 1-5. IEEE, 2011

[5] Li, Bin, Hui Shen, and David Tse. "An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check." *IEEE Communications Letters* 16, no. 12 (2012): 2044-2047

[6] Niu, Kai, and Kai Chen. "CRC-aided decoding of polar codes." *IEEE Communications Letters* 16, no. 10 (2012): 1668-1671.

[7] Dizdar, Onur, and Erdal Arıkan. "A high-throughput energy-efficient implementation of successive cancellation decoder for polar codes using combinational logic." *IEEE Transactions on Circuits and Systems I: Regular Papers* 63, no. 3 (2016): 436-447.

[8] Pamuk, Alptekin. "An FPGA implementation architecture for decoding of polar codes." In *Wireless Communication Systems (ISWCS), 2011 8th International Symposium on*, pp. 437-441. IEEE, 2011.

[9] Eslami, Ali, and Hossein Pishro-Nik. "A practical approach to polar codes." In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pp. 16-20. IEEE, 2011.

[10] Lin, Jun, and Zhiyuan Yan. "An efficient list decoder architecture for polar codes." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 23, no. 11 (2015): 2508-2518.

[11] Leroux, Camille, Alexandre J. Raymond, Gabi Sarkis, and Warren J. Gross. "A semi-parallel successive-cancellation decoder for polar codes." *IEEE Transactions on Signal Processing* 61, no. 2 (2013): 289-299.

[12] Trifonov, Peter. "Efficient design and decoding of polar codes." *IEEE Transactions on Communications* 60, no. 11 (2012): 3221-3227.

[13] Tal, Ido, and Alexander Vardy. "How to construct polar codes." *IEEE Transactions on Information Theory* 59, no. 10 (2013): 6562-6582.

[14] Coding Theory Wikipedia https://en.wikipedia.org/wiki/Coding_theory

[15] Forward Error Correction Wikipedia
http://en.wikipedia.org/wiki/Forward_error_correction.

[16] Lecture Notes on Channel Coding Schemes by *Prof. Dr. Qilian Liang EE6367 University of Texas Arlington.*

[17] Polar Codes 1.1.0 C++ implementation of modules for Polar Code - http://users.ecs.soton.ac.uk/rm/wp-content/PolarCode/index.html.

[18] Vangala, Harish, Yi Hong, and Emanuele Viterbo. "Efficient algorithms for systematic polar encoding." *IEEE communications letters* 20, no. 1 (2016): 17-20.

[19] Vangala, Harish, Emanuele Viterbo, and Yi Hong. "A comparative study of polar code constructions for the AWGN channel." *arXiv preprint arXiv:1501.02473* (2015).

[20] Leroux, Camille, Ido Tal, Alexander Vardy, and Warren J. Gross. "Hardware architectures for successive cancellation decoding of polar codes." In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pp. 1665-1668. IEEE, 2011.

[21] Balatsoukas-Stimming, Alexios, Mani Bastani Parizi, and Andreas Burg. "LLR-based successive cancellation list decoding of polar codes." *IEEE Transactions on Signal Processing* 63, no. 19 (2015): 5165-5179.

[22] Pedarsani, Ramtin, S. Hamed Hassani, Ido Tal, and Emre Telatar. "On the construction of polar codes." In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pp. 11-15. IEEE, 2011.

[23] Wu, Dongsheng, Aijun Liu, Yingxian Zhang, and Qingshuang Zhang. "Parallel concatenated systematic polar codes." *Electronics Letters* 52, no. 1 (2015): 43-45.

[24] Niu, Kai, Kai Chen, Jiaru Lin, and Q. T. Zhang. "Polar codes: Primary concepts and practical decoding algorithms." *IEEE Communications magazine* 52, no. 7 (2014): 192-203.

[25] IIT Bombay Lecture series on Channel Coding by Prof. Bikash. Kumar https://www.youtube.com/watch?v=G2Xk9fvCN0.

[26] Bravo-Santos, Angel. "Polar codes for the Rayleigh fading channel." *IEEE Communications Letters* 17, no. 12 (2013): 2352-2355.

[27] 5G Wikipedia https://en.wikipedia.org/wiki/5G

[28] Massive MIMO and channel modelling for millimeter wave *Gustavo Fedrich, Departmento DE Communicaoes, Unicamp.*

[29] 5G Virtualization Early trends in 5G technology https://knect365.com/5g-virtualisation/article/1a7b1504-12db-47ba-9ecf-aacad96c3582/early-trends-in-5g-technology-leadership

[30] Anderson, John B., and Seshadri Mohan. *Source and channel coding: an algorithmic approach*. Vol. 150. Springer Science & Business Media, 2012.

[31] Valenti, Matthew C., and Jian Sun. "The UMTS turbo code and an efficient decoder implementation suitable for software-defined radios." *International journal of wireless information networks* 8, no. 4 (2001): 203-215.

[32] LDPC Codes Wikipedia https://en.wikipedia.org/wiki/Low-density_parity-check_code.

[33] Cisco Support Community on LDPC Codes
https://supportforums.cisco.com/document/12526916/low-density-parity-check-ldpc-codes

[34] Wadayama, Tadashi, Keisuke Nakamura, Masayuki Yagita, Yuuki Funahashi, Shogo Usami, and Ichi Takumi. "Gradient descent bit flipping algorithms for decoding LDPC codes." In *Information Theory and Its Applications, 2008. ISITA 2008. International Symposium on*, pp. 1-6. IEEE, 2008.

[35] R.G.Gallager, "*Low-Density Parity-Check Codes*", in Research Mono-graph series. Cambridge, MIT Press,1963.

[36] M.Jiang, C.Zhao, Z.Shi, and Y.Chen, "An improvement on the modified weighted bit flipping decoding algorithm for LDPC codes," IEEE Communications Letters, vol.9, no.9, pp.814–816, 2005.

[37] S.Boyd and L. Vandenberghe, *"Convex optimization,"* Cambridge University Press, 2004.

[38] *Massive MIMO and Channel Modeling for Millimeter* Wave Gustavo Fraidenraich Engenharia Elétrica Departamento de Comunicações Unicamp1

[39] *early-trends-in-5g-technology-leadership* https://knect365.com/5g-virtualisation/article/1a7b1504-12db-47ba-9ecf-aacad96c3582/early-trends-in-5g-technology-leadership

[40] *Polar Codes Wikipedia* https://en.wikipedia.org/wiki/Polar_code_(coding_theory)

[41] *Polar Codes tutorial by Erdal Arikan at UC berkely*
https://simons.berkeley.edu/sites/default/files/docs/2691/slidesarikan.pdf

[42] Niu, Kai, Kai Chen, Jiaru Lin, and Q. T. Zhang. "Polar codes: Primary concepts and practical decoding algorithms." *IEEE Communications magazine* 52, no. 7 (2014): 192-203.

[43*] Polar codes: construction and performance improvement* Li, Huijun, Electrical Engineering & Telecommunications, Faculty of Engineering, UNSW

[44] Yuan, Bo. (2015). Algorithm and VLSI Architecture for Polar Codes Decoder. Retrieved from the University of Minnesota Digital Conservancy, http://hdl.handle.net/11299/175384.

[45] Özgür, Üstün *"A Performance comparison of polar codes with convolutional turbo codes"* Department of Electrical and Electronic Engineering and the Institute of Engineering and Sciences of Bilkent University.