Differential properties of polar codes

Georgiy Shalin[1*], *Dmitriy* Pokamestov[1], *Yakov* Kryukov[1], *Artem* Shinkevich[1], *Andrey* Brovkin[1], and *Eugeniy* Rogozhnikov[1]

[1]Tomsk State University of Control and Radioelectronics, 47, Vershinina str., 634034 Tomsk, Russia

Abstract. This paper discusses the features of polar coding as a differential encoder for a binary erasure channel. One of the modern methods of error-correcting coding is polar codes, which has great prospects in the development of current and future wireless communication systems. At the moment, polar coding is used only in fifth-generation systems. In this paper, a description of polar coding is presented, a description of a polar decoding algorithm capable of restoring messages with an error probability close to 1 is presented, and a general model of a binary channel with erasures is described. The channel model represents a fixed number of introduced errors, depending on the size of the transmitted message. The result of the study is the dependence of the bit-error ratio (BER) on the error probability for low, medium and high code rates, which illustrate the possibility of the polar coding algorithm to restore a message with a partially or completely inverse data stream. Keywords: polar codes, block codes, differential encoder, generator matrix.

1 Introduction

Currently, wireless communication systems are rapidly enhancing together with new technologies and algorithms being developed. These innovations can be used to upgrade existing developments, enhance the quality and improve data rate. One of the methods for increasing system noise immunity is channel coding.

Modern communication systems employ several coding algorithms, including polar codes. Polar coding is a relatively new method of error-correcting coding proposed by Arikan in 2008 [1].

Polar codes hold great promise in the development of communication systems, due to their simplicity and ability to bring the data rate closer to the Shannon limit, that is, to the boundaries of error-free data transmission with noise interference.

Polar codes replaced convolutional codes and were applied in the 5G New Radio (NR) communication systems [2, 3], namely in channel coding for downlink and uplink communication channels in information control channels in the enchanted mobile broadband (eMBB), massive machine-type communications (mMTC), and ultra-reliable low latency communication (URLLC) scenarios [4]. In particular, they are used in the broadcast channel (BCH), as well as in the service information transmission channels: downlink control information (DCI) and uplink control information (UCI) [3].

* Corresponding author: shalingn1120@gmail.com

2 Constructing polar codes

Polar codes are non-systematic block codes with $N = 2^n$ (2, 4, 8, etc.) length of a codeword.

As in block coding, the process of constructing a codeword in polar coding is reduced to multiplying message vector u by generator matrix **G**.

$$d = uG, \tag{1}$$

where **u** is a sequence of bits arriving at the multiplier input.

Arikan matrix [5] is the basic generator matrix (2,2):

$$\mathbf{G} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}. \tag{2}$$

This is a generator matrix for polar coding, since higher-order matrices are generated from it. It should be noted that regardless of the order, all the formed matrices will be square, that is, having the equal number of columns and rows.

The coding procedure is performed as follows:

$$\mathbf{d} = [d_1 \ d_2] = [u_1 \ u_2] \cdot \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} = [u_1 + u_2 \ u_2]. \tag{3}$$

In this procedure, d_1 and d_2 are output channels, while u_1 and u_2 are input channels. Thus, one can see that information about channel u_2 is contained in both d_1 and d_2, while only channel d_1 contains information about u_1. It means that channel u_2 is more reliable than u_1, since there is the information about u_2 in both output channels, while the information about u_1 is only in one of them. Such an operation is called polar transformation, which means that after the channel is polarized, one channel becomes more noise-immune, and the other one — less. This procedure is shown in Figure 1

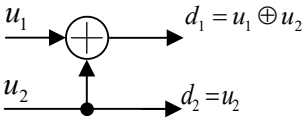

Fig. 1. Polar transformation.

A generalized data transmission channel in a polar-coded system is shown in Figure 2 Data stream u arrives at the multiplier block, where it is multiplied by generator matrix $\mathbf{G_N}$, after which codeword **d** is formed. Data stream $\mathbf{r} = [r_1, r_2, \ldots, r_N]$ arrives at the receiving side after modulation and passing the radio wave propagation channel. Vector **r** values are estimates of the log-likelihood ratio (LLR) [6]. LLR is a natural logarithm of the likelihood ratio of the fact that the received symbol corresponds to the bit zero value to the fact that the received symbol corresponds to a single value:

$$LLR = \ln \frac{P_r(r_i \mid u = 0)}{P_r(r_i \mid u = 1)}, \tag{4}$$

where P_r is the probability of occurrence of a certain bit, $\mathbf{r_i}$ are accepted symbol values and u are transmitted bits.

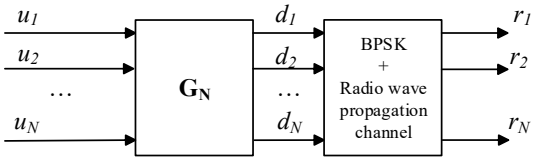

Fig. 2. Generalized communication channel diagram.

3 Polar coding

The reliability of a channel in a polar-coded system depends on the number of check bits formed with its help.

The idea behind polar coding is to freeze the least reliable channels. It means that we will transmit the so-called frozen bits through them, that is, bits with a constant value, for example, 0. If the codeword length is N, and the message length is K, then according to this principle, N-K bits will be frozen. The encoding procedure is described by the expression:

$$\mathbf{d} = [f_1 \ldots f_{N-K} \ m_1 \ldots m_K]\mathbf{G}, \tag{5}$$

where $f_i = 0$ are frozen bits and m_k are message bits.

Polar coding can be represented in three ways: using a generator matrix, using a code tree and as a block diagram. Each of the methods gives the same result and any of them can be used when encoding. All examples are given for N = 4 codeword length.

Figure 3 shows encoding algorithm thru code tree.

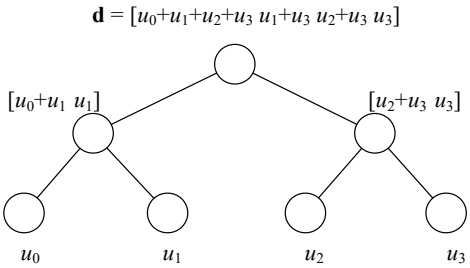

Fig. 3. Encoding thru code tree.

where $\mathbf{u_N}$ are transmitted bits, and **d** is a codeword.

The circles in the above diagram are called vertices, and the rows are called generations. The bottom row is the youngest generation; the younger vertices are connected to the older ones and are their descendants.

Figure 4 shows encoding algorithm using a block diagram.

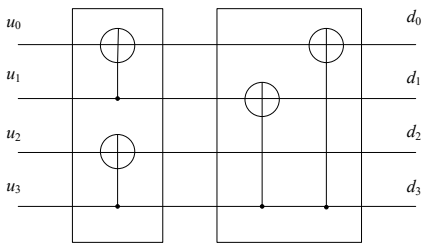

Fig. 4. Encoding with help of block diagram.

Encoding that employs a generator matrix is written as

$$\mathbf{d} = \mathbf{uG} = [u_0\ u_1\ u_2\ u_3]\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} =$$

$$= [u_0 + u_1 + u_2 + u_3\ \ u_1 + u_3\ \ u_2 + u_3\ \ u_3] \tag{6}$$

Typically, encoding thru code tree and encoding with help of a block diagram are used to visualize the polar coding algorithm, while encoding using a generator matrix shows the mathematical representation of codeword constructing. Each of the encoding options gives the same result at identical parameters. In practice, polar coding is performed using special high-speed algorithms.

4 Polar decoding

The polar code decoding algorithms are based on the sequential interference compensation SC (successive cancellation) approach [7]. The basic algorithm is the algorithm of the same name. In addition, there are SCL (successful cancellation list) [8] and SCS (successful cancellation stack) [9] algorithms that demonstrate higher efficiency [10]. All three algorithms are based on a similar principle: sequential tree traversal and compensation of bit operations.

In all three algorithms, the input data are "soft" estimates of the accepted symbols: log-likelihood ratio or LLR.

4.1 The SC algorithm

The simplest decoding algorithm is SC, because a single decision is made in each iteration in this algorithm. Decoding is performed over a number of iterations. Their number coincides with the number of code tree branches, excluding the younger generation of descendants. For each vertex, one performs the following actions:

1. Recalculating values for the left descendant (the left incoming branch of the vertex) based on the input values of L (LLR).
2. Decision-making for the right descendant.
3. Recovering the codeword.

Let us take a detailed look at each stage. Let $a_1 \ldots a_M$ be LLR values arriving for the left descendant, while $b_1 \ldots b_M$ — LLR values arriving from the right descendant. The value of M depends on the generation of branches under consideration. So, in the second generation M=1 (one value arrives at the input from each descendant), in the third M =2, etc.

Figure 5 provides an explanation of the first decoding stage.

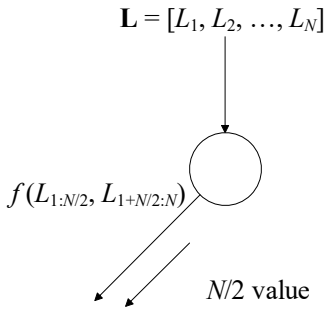

Fig. 5. Decision-making for the left descendant in the older generation.

The first stage reduces to making decisions for the left descendant of the *M*-th order. It can be represented as follows:

$$\text{minsum}(a_{1:M},b_{1:M})=[\text{minsum}(a_1,b_1), \text{minsum}(a_2,b_2), \ldots, \text{minsum}(a_M,b_M)], \quad (7)$$

where minsum is the operation of the minimum approximation sum [11].

In the following, let us denote the minsum function as *f*.

$$f(a,b) = \text{sign}(a)\cdot\text{sign}(b)\cdot\min(|a|,|b|), \quad (8)$$

where sign is the operator for number sign generation and min is the minimum value of a pair of numbers.

The second stage is the recalculation of values based on the right descendant. This procedure is shown in Figure 6.

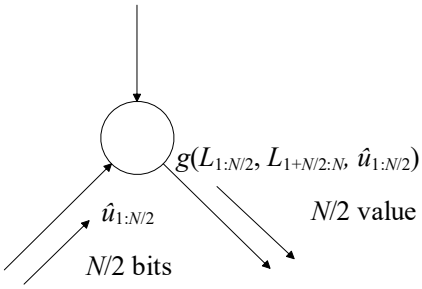

Fig. 6. Decision-making for the right descendant in the older generation.

The second stage, that is, making a decision for the right descendant in the older generation, can be described as follows:

$$g(a_{1:M},b_{1:M},\hat{u}_{1:N/2}) = [g(a_1,b_1,\hat{u}_1), g(a_2,b_2,\hat{u}_2), \ldots, g(a_M,b_M,\hat{u}_{N/2})], \quad (9)$$

where $\hat{u}$ is a "hard" solution deduced from LLRs recalculated in the left branch. $\hat{u}$ value can be deduced from the LLR sign: LLR>0 corresponds to the bit value equal to 0, and vice versa, and LLR<0 corresponds to the bit value equal to 1.

The calculation of each conditional sum of two LLRs and the bit estimated in the previous metric is performed using the function

$$g(a,b,c) = b + (1-2c)\cdot a, \quad (10)$$

where a is the LLR value of the left branch; b is the LLR value of the right branch; c is the bit value estimated by the left descendant.

Thus, if the values of bit $\hat{u}$ in the left branch are 0, then a and b add up; and if value $\hat{u}$=1, then a is subtracted from b.

The next step is the procedure of recovering a codeword shown in Figure 7. It is executed in all iterations except the last one. This stage is equivalent to polar transformation, that is, coding.

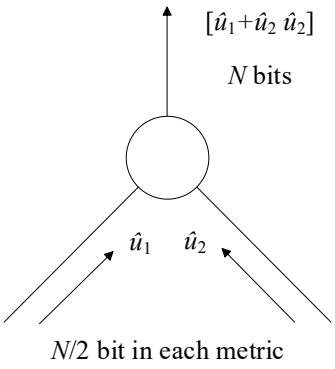

Fig. 7. Codeword recovering.

In the next iteration, the same steps are repeated for the next branch in traversal order of the graph.

The general code tree illustrating the decoding algorithm for a codeword of size $N = 4$ is shown in Figure 8, where numbers under the operations indicate the order of their execution.

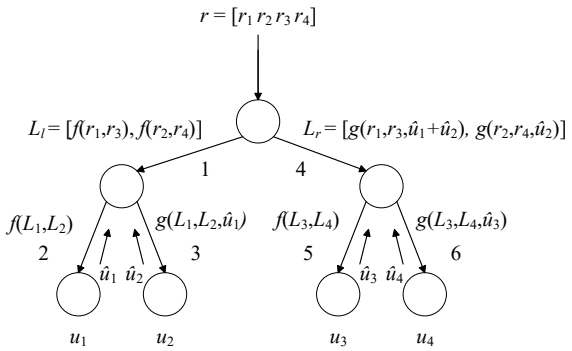

Fig. 8. SC decoding algorithm.

4.2 The SCL algorithm

In polar coding, the SCL decoding algorithm is used in conjunction with cyclic redundancy check (CRC) [9, 12, 13]. This is a kind of cyclic codes that makes it possible to detect errors, but not to correct them. CRC is used in communication systems to control the integrity of packets.

The transmission of CRC data packets consists of the original message and check bits that are added from the right. The check bits are obtained as the remainder of dividing the message polynomial by the generator code polynomial [14]. This operation is performed on a shifting register, which allows significantly reducing the computational complexity.

When constructing a communication channel model with polar coding, the SCL algorithm was used; this method employs two "hard" solutions for each path, while the path is the connection of two descendants, the younger and the older. Several solutions are selected both for the left and right descendant. As a result, we obtain a group of codewords; the decision on the correctness of a codeword is made by calculating the checksum. The CRC calculation is performed as follows: decoded codewords are checked by the generated checksum polynomial, if a parity bit obtained by the receiver does not match the one calculated by the message, then the next codeword is checked.

The SCL decoder itself is a set of the N-*th* number of SC decoders. The SCL decoder circuit is shown in Figure 9.

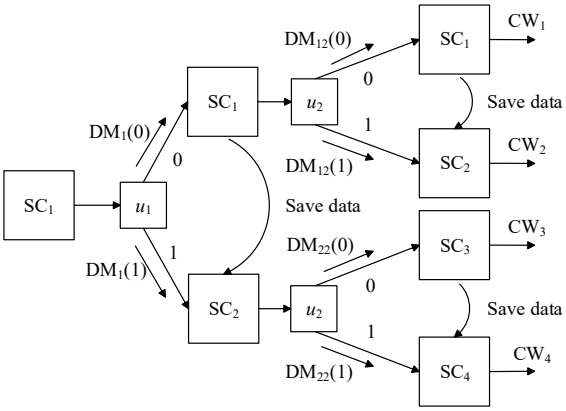

Fig. 9. SCL decoder.

In this algorithm, the LLR values as well as path metrics are calculated in each iteration. Two types of metrics are considered here: path metric (PM), that is, the total metric, and decision metric (DM).

The square blocks in Figure 9 represent SC decoders. After passing each SC decoder, two decisions are made: that the estimate of $\hat{u}_i$ is equal to 1 and that it is equal to 0. For each value, we should deduce DM and add current PM. DM is calculated by the following rules:

1. If bit estimate is $L_i \geq 0$, then $\hat{u}_i = 0$ has DM = 0 and $\hat{u}_i = 1$ has DM = $|L_i|$.
2. If bit estimate $L_i < 0$, then $\hat{u}_i = 1$ has DM = 0 and $\hat{u}_i = 0$ has DM = $|L_i|$.
3. If u_i is frozen, then DM = 0 at $L_i \geq 0$, and DM = $|L_i|$ at $L_i < 0$.

5 Distinctive features of polar coding as differential encoder

One of the coding methods is a differential encoder. In essence, it is modulo two addtion to all arriving values with a shift of one character. Let us assume that the input data stream to the encoder takes the form $\mathbf{u} = [u_1\ u_2\ u_3]$, then the encoded data stream is $\mathbf{d} = [u_1\ u_1+u_2\ u_1+u_2+u_3]$. The differential encoder circuit is shown in Figure 10.

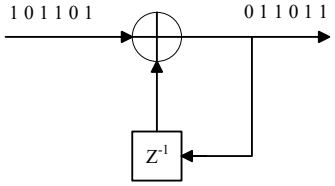

Fig. 10. Differential encoder circuit.

The differential decoder also adds modulo two to all the received data with a shift of one character. Let the data stream $\mathbf{x} = [u_1\ u_1+u_2\ u_1+u_2+u_3]$ arrive at the decoder input, then the decoded data stream takes the following values: $\mathbf{u} = [u_1\ u_1+u_2+u_1\ u_1+u_2+u_3+u_1+u_2]$. After performing modulo two addition to each element of sequence **u**, we get the following form of the decoded message: $\mathbf{u} = [u_1\ u_2\ u_3]$. The differential encoder circuit is shown in Figure 11.

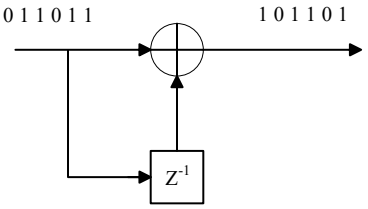

Fig. 11. Differential encoder circuit.

One important feature of polar coding is the ability to decode messages with a bit error probability close to 100%, that is, when almost completely or completely inverted bits arrive at the decoder. This is due to the encoding algorithm that has properties of differential encoding.

As an example, we take sequence $\mathbf{r} = [r_1, r_2, r_3, r_4]$. Let us perform decoding for the left descendant: this procedure is shown in Figure 12.

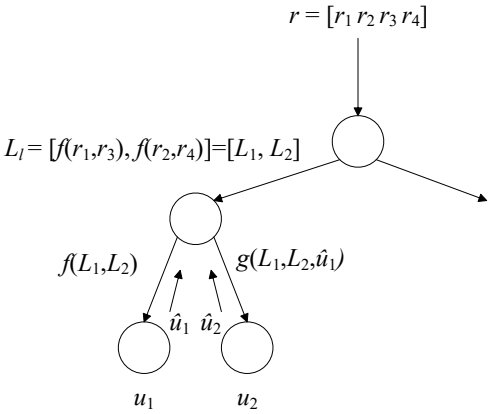

Fig. 12. Decoding by the left metric.

The first stage is the following estimation:

$$L_l = [f(r_1,r_3),\ f(r_2,r_4)] = [L_1, L_2], \tag{11}$$

where $f(r_1,r_3)$ is the minimum sum between r_1 and r_3;

$f(r_2,r_4)$ is the minimum sum between r_2 and r_4.

Next, we estimate the first bit by the youngest left descendant:

$$\hat{u}_1 = [f(L_1,L_2)]. \tag{12}$$

The final stage is to estimate the second bit by the youngest right descendant:

$$\hat{u}_2 = L_2 + (1-2\cdot \hat{u}_1)\cdot L_1 \tag{13}$$

If we represent the above operations as differential coding, given that $r_1 = u_1 + u_2 + u_3 + u_4$, $r_2 = u_2 + u_4$, $r_3 = u_3 + u_4$, $r_4 = u_4$, then the differential coding process takes the following form:

$$L_l = [r_1 + r_3,\ r_2 + r_4] = [u_1 + u_2 + u_3 + u_4 + u_3 + u_4,\ u_2 + u_4 + u_4] = [u_1 + u_2,\ u_2]$$

$$\hat{u}_1 = u_1 + u_2 + u_2$$

It can be noticed from the foregoing that the minimum sum matches the process of differential decoding.

Knowing the estimate of the previous bit, it is possible to recover the next one: in this case, by estimating $\hat{u}_1$, one can get the estimate of $\hat{u}_2$.

The bit inversion can be represented as follows: $\bar{d}_1 = u_1 + u_2 + u_3 + u_4 + 1$, $\bar{d}_2 = u_2 + u_4 + 1$, $\bar{d}_3 = u_3 + u_4 + 1$, $\bar{d}_4 = u_4 + 1$. Then the bit estimation process has the following form:

$$L_l = [\bar{r}_1 + \bar{r}_3,\ \bar{r}_2 + \bar{r}_4] = \ = [u_1 + u_2 + u_3 + u_4 + 1 + u_3 + u_4 + 1,\ u_2 + u_4 + 1 + u_4 + 1] = [u_1 + u_2,\ u_2]$$

$$\hat{u}_1 = u_1 + u_2 + u_2$$

It can be seen that the result of decoding inverted bits matches error-free bit decoding.

From the above example, it is apparent that the process of polar coding and decoding can recover the inverse data stream, but the last inverted bit is decoded incorrectly. The introduced redundancy and checksum checking process in the SCL algorithm allows overcoming this decoding error.

6 System model

The diagram of the communication channel simulation using polar coding is shown in Figure 13.

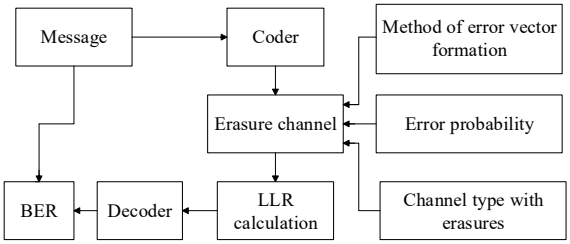

Fig. 13. Communication channel diagram.

The simulation was carried out in a binary erasure channel for the following code parameters: $(N,K) = (64,512)$, $(N,K) = (128,256)$, $(N,K) = (512,1024)$.

The error vector in the erasure channel is formed according to the Bernoulli distribution. Error probability p is used as the input data.

In the implemented simulation, we used an adaptive calculation of the number of errors for the erasure channel. The number of errors is equal to [15]:

$$N_{er} = N_{in} p\,, \tag{14}$$

where N_{in} is the size of the input data vector.

Figures 14, 15, 16 show the dependencies of BER on the error probability for different encoding rates. Since binary alphabet characters, that is, "1" and "0", are transmitted in the channel, the probability of erasure equal to 1 corresponds to a complete bit inversion.

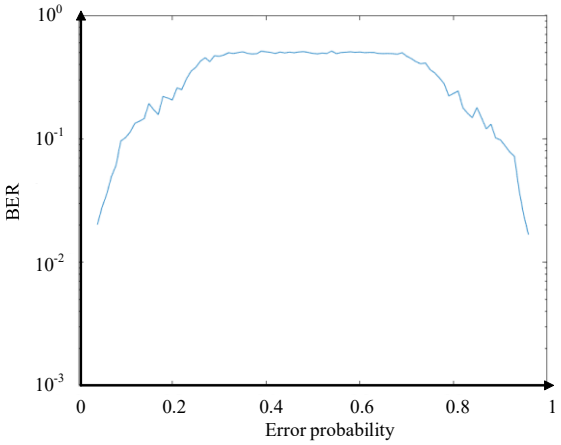

Fig. 14. Error probability at code rate of 64/512.

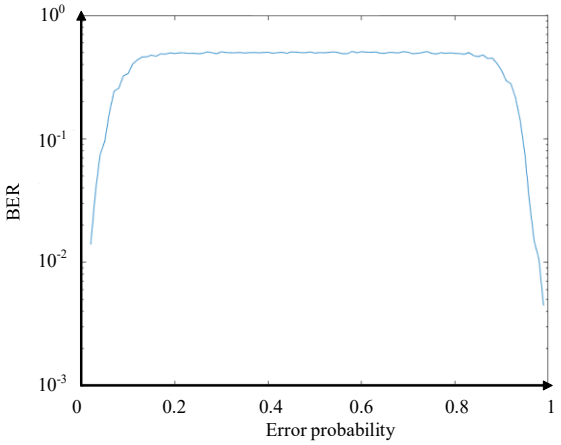

Fig. 15. Error probability at code rate of 128/256.

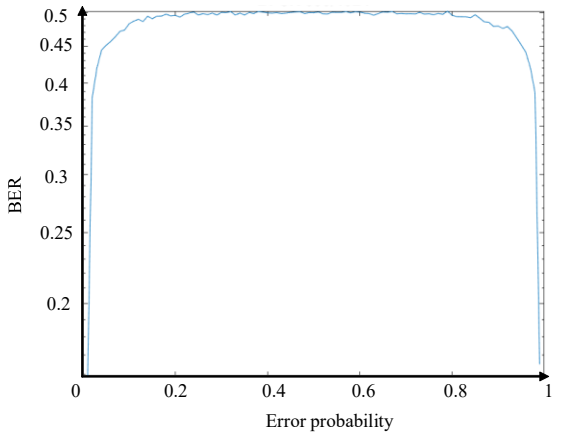

Fig. 16. Error probability at code rate of 896/1024.

The resulting graphs take the form of a "bell", so the dependencies are symmetric with respect to the central error probability of 0.5, that is, the error probability of 0.9 matches 0.1, which means that the decoder can decode a message with an error probability close to 1. This effect is associated with the feature of the differential polar code discussed in paragraph 5.

One can notice that the lower the code rate is, the narrower the graph becomes, and the edges take a flat appearance. This corresponds to the classical concept of the error correction capability: the lower the code rate is, the greater the number of bit errors that can be corrected, and vice versa. Thus, at a low code rate, the dependence tends to zero with a higher error probability at the input compared to the average and high code rates.

7 Conclusion

This paper presents polar coding as differential. The algorithms of polar and differential coding and data transmission through a binary erasure channel have been investigated. The comparison of two algorithms and the results obtained in Figures 14–16 make it clear that the distinctive feature of polar coding as a differential encoder allows decoding messages, even with the error probability close to 100%. Also, the error correction capability depends on the code rate: so, for a low code rate, the bit-error probability decline to zero begins at a higher value of the error probability compared to the average and high code rates.

Acknowledgment

The work is supported by the Russian Science Foundation grant. Project number 22-79-10148, https://rscf.ru/en/project/22-79-10148/.

References

1. E. Arikan, IEEE Communications Letters **12(6)**, 447–449 (2008)
2. *3GPP TS 36.212 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; NR; Multiplexing and channel coding V15.2.1* (2018) https://www.etsi.org/deliver/etsi_ts/136200_136299/136212/15.02.01_60/ts_136212v150201p.pdf
3. *3GPP TS 38.212 3rd Generation Partnership Project; Technical Specification Group Radio Access Network. Multiplexing and channel coding 16.0.0* (2019) URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3214
4. V. Bioglio, C. Condo, IEEE Communications Surveys & Tutorials, 29–40 (2020)
5. E. Arikan, IEEE Communications letters **15(8)**, 860–862 (2011)
6. Altug Sural, Goksu Sezer, Yigit Ertugrul, Orthan Arikan, Erdal Arikan, IEEE 30th International Symposium on Personal, Indoor and Mobile Radio Communications (PIRMC Workshops) **7** (2019)
7. R. Pedarsani, *Polar Codes: Construction and Performance Analysis* (Swiss Federal Institute of Technology (EPFL), 2011)
8. I. Tal, A. Vardy, IEEE Transactions on Information Theory May, 2213–2226 (2015)
9. K. Niu, K. Chen, Electronics Letters **48(12)**, 695–696 (2012)

10. A. Smeshko et al, *Decoding of polar codes: comparison of a list and stack decoding algorithms*, URL: https://sites.skoltech.ru/app/data/uploads/sites/52/2020/09/Proekt-Sochi-AA-draft.pdf
11. E. Arıkan, IEEE journal on Selected Areas in Communications **34(2)**, 209–223 (2015)
12. A. Balatsoukas-Stimming, M.B. Parizi, A. Burg, IEEE transactions on signal processing **63(19)**, 5165–5179 (2015)
13. K. Chen, K. Niu, J. Lin, IEEE Transactions on Communications **61(8)**, 3100–3107 (2013)
14. R. Morelos-Zaragoza, *The Art of Error Correcting Coding. Methods, algorithms, application* (Technosphere, Moscow, 2005)
15. M. Hazewinkel, *Encyclopedia of Mathematics, Springer* (2001) ISBN 978-1-55608-010-4