https://doi.org/10.26552/com.C.2013.2A.167-173

Jakub Sedy – Pavel Silhavy – Ondrej Krajsa – Ondrej Hrouza *

PERFORMANCE ANALYSIS OF TURBO CODES

The article focuses on the performance analyses of Turbo Codes. These codes belong to the group of error correction codes. By their use, it is possible to achieve high system performance. The performance analysis is based on simulations for different code parameters that affect the code gain, bit error rate and computational complexity. Furthermore, it presents the basic structure of the encoder and decoder and the principle of encoding and decoding, which uses the Viterbi algorithm with soft input and soft output.

Keywords: *Turbo encoder, turbo decoder, iterative decoding, SOVA, BER.*

1. Introduction

The basic elements of turbo codes are convolutional codes [1, 2] and decoding algorithms that use soft input and soft output [3, 4, 5, 6]. The input bit sequence is encoded by two encoders, between which is stored interleaver to ensure that the encoded sequences are mutually independent. RSC (Recursive Systematic Convolutional) encoders [1, 2] are often used where each RSC encoder produces a systematic output, which is equivalent to input information, and produces parity bits. Both parity sequences can be punctured before they are transferred with systematic bits to the decoder. Via puncturing it is possible to reduce the number of parity bits to one half and thereby also increase the information rate to 1/2.

For decoding, special algorithms must be used which use soft input and soft output [4, 5, 6]. These soft inputs and outputs do not determine only whether the decoded bit has the logical value 0 or 1 but the likelihood ratio which determines the probability of whether the bit was correctly decoded. The turbo decoder operates iteratively. The first iteration of the first decoder gives an estimate of the original data sequence, based on the soft output channel. It also provides an extrinsic output. Extrinsic output for a given bit is not dependent on the value of the transmission channel for this bit but on the information for the surrounding bits. This extrinsic output from the first decoder is used as a-priori information for the second decoder together with input information from the channel.

The second decoder will give us again extrinsic information and soft output. In the second iteration the extrinsic information from the second decoder in the first iteration is used as a-priori information for the first decoder. Thus the decoder achieves a more accurate estimate of the decoded bits than was the case in the first iteration. This cycle is continuously repeated. In each iteration of the two decoders soft output and extrinsic information are calculated based on the input sequence and a-priori information obtained from extrinsic information of the previous decoder. After each iteration, the BER (Bit Error Rate) decreases.

2. Turbo encoder

The block diagram of turbo encoder [5, 7] is shown in Fig. 1. Two identical encoders are used here, usually RSC, which are separated by interleaver. It is possible to use a structure with more than two encoders, but in this chapter we will deal with the classical structure of two RSC encoders [1, 2].

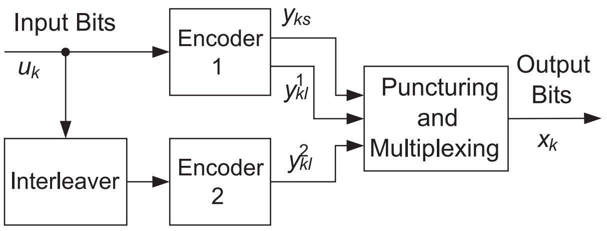

Fig. 1 Turbo encoder block diagram

The input bit sequence is fed to the input of the first encoder where it is encoded. The output of the first encoder is formed by systematic and parity bits. The input bits for the second encoder are interleaved and encoded in the second encoder. The input bits for the second encoder thus become independent of the input bits of the first encoder. Typically, pseudo-random or block interleaver is used. The second encoder produces parity bits only. The output from the two encoders is punctured and then multiplexed. Usually both RSC encoders have an information rate of 1/2 and give one systematic and one parity bit for each input bit. This means that the turbo encoder output sequence contains for each input bit one systematic and two parity bits, i.e. $y_{1s'}$, $y^1_{1l'}$, $y^2_{1l'}$, $y_{2s'}$, $y^1_{2l'}$, $y^2_{2l'}$, ...,

* **Jakub Sedy, Pavel Silhavy, Ondrej Krajsa, Ondrej Hrouza**
Faculty of Electrical Engineering and Communication, Brno University of Technology, Czech Republic, E-mail: jakub.sedy@phd.feec.vutbr.cz

$y_{ks'}$, $y^1_{kl'}$, $y^2_{kl'}$. For this output sequence the turbo encoder has an information rate of 1/3. For the total information rate to be 1/2, the output bits from the turbo encoder must be punctured. The output sequence is punctured so that all the systematic bits are preserved and only the parity bits are punctured. Puncturing of the systematic bits will degrade the code performance. After puncturing and multiplexing the turbo encoder output sequence x_{kl} would be $y_{1s'}, y^1_{1l'}, y^2_{1l'}, y_{2s'}, y^1_{2l'}, y^2_{2l'}, \ldots, y_{ks'}, y^1_{k+1s'}, y^2_{k+1l'}$.

3. Turbo decoder

A. Soft output Viterbi algorithm

The turbo decoder uses the Viterbi algorithm which is referred to as the SOVA (Soft-Output Viterbi Algorithm [3, 4, 5]. For decoding turbo codes, this algorithm has two modifications. The first modification adapts the path metric so that it takes into account a-priori information when selecting the maximum likelihood paths in the trellis diagram. The second modification of the algorithm consists in soft output in the form of a-posteriori LLR (Log Likelihood Ratio) $L\,(u_k \mid \underline{y})$ for each decoded bit.

The first modification considers the state sequence $\underline{s}_k^s$ which gives the states along the surviving paths at the state $S_k = s$ at stage k in the trellis diagram. The metric should be easy to compute via the recursive way where we go from stage k - 1 to the kth stage in the trellis diagram. A suitable metric for the path $\underline{s}_k^s$ is defined as [4, 5]:

$$M\big(\underline{s}_k^{s}\big) = M\big(\underline{s}_{k-1}^{\acute{s}}\big) + \frac{1}{2}u_k L\big(u_k\big) + \frac{L_c}{2}\sum_{l=1}^{n} y_{kl}x_{kl'} \tag{1}$$

where $M(\underline{\acute{s}}_k^s)$ is the metric of surviving path through the state S_{k-1} at stage $k-1$ in the trellis diagram, u_k is the encoder input bit, x_{kl} is the transmitted channel sequence (output from encoder) associated with a given transition, and y_{kl} is the received sequence from the transmission channel for that transition. Using the transmission channel with BPSK (Binary Phase Shift Keying) modulation and AWGN (Additive White Gaussian Noise), the channel reliability L_c is defined as follows [5]:

$$L_c = 4\alpha\frac{E_b}{2\sigma^2}, \tag{2}$$

where E_b is the transmitted energy per bit, α is the fading amplitude, and σ is the noise variance.

Now we will discuss the second modification of the algorithm which is the soft output. In a binary trellis diagram there will be two paths reaching the state $S_k = s$ at the stage k. The modified Viterbi algorithm takes a-priori information, calculates the metric of these two paths according to Equation (1) and discards the path with a lower metric. When both paths $\underline{s}_k^s$ and $\underline{\hat{s}}_k^s$ reaching state S_k have the metric $M(\underline{s}_k^s)$ and $M(\underline{\hat{s}}_k^s)$, respectively and the path with the higher metric $\underline{s}_k^s$ is selected as surviving, we define the difference metric Δ_k^s of these paths as [4, 5]:

$$\Delta_k^s = M\big(\underline{s}_k^s\big) - M\big(\underline{\hat{s}}_k^s\big) \geq 0, \tag{3}$$

where $M(\underline{s}_k^s)$ is the metric for the surviving path, and $M(\underline{\hat{s}}_k^s)$ is the metric for the discarded path.

When we reach the end of the trellis diagram and find the ML (Maximum Likelihood) path, it is necessary to find the LLR. This determines the reliability of deciding on the bits around the ML path. The Viterbi algorithm shows that all the surviving paths at the stage in the trellis diagram come from the same path a few steps before this stage. This previous stage may attain δ transitions before the stage k, where δ is usually set to five times the constraint length of the convolutional code. Therefore, the bit value u_k associated with the transition from the state $S_{k-1} = \acute{s}$ to the state $S_k = s$ on the ML path may be different when the Viterbi algorithm selects one path merged with the ML path instead of the ML path after the δ transitions, i.e. $k + \delta$ stage in the trellis diagram. If the algorithm selects one of the paths merged with the ML path, it will not affect the value u_k, because this path will differ from the ML path from the transition $S_{k-1} = \acute{s}$ to $S_k = s$. When we calculate the LLR for the bit u_k, SOVA has to take into account the probability of paths merging with the ML path at the stage k to stage $k + \delta$. By comparing the differences in the metric $\Delta_i^{s_i}$ for all states s_i along the ML path from the state $i = k$ to $i = k + \delta$. This LLR is defined as [4, 5]:

$$L\big(u_k \big| \underline{y}\big) \approx u_k \min_{i=k\cdots k+\delta\, u_k \neq u_k^i} \Delta_i^{s_i}, \tag{4}$$

where u_k is the bit value of the ML path, and u_k^i is the value of the bit of the path that merged with the ML path and was discarded in the state i. The minimization in Equation (4) is only used for paths merging with the ML path which gives a different value for the bit u_k when this path is selected as the surviving path. The paths that gave the same value u_k as the ML path do not affect the decision.

B. Implementation of the SOVA

SOVA is implemented as follows. In every state at every stage in the trellis diagram the metric $M(\underline{s}_k^s)$ is calculated for the two paths merging into the state using Equation (1). The path with the higher metric is chosen as the surviving for this state and the metric indicator stored as the Viterbi algorithm does it. However, in order to provide reliable decoded bits, it also stores the value of $L\,(u_k \mid \underline{y})$ calculated by using Equation (4). The metric differences between the surviving and the discarded path are stored together with the binary vector of $\delta + 1$ bits in length, which indicates the sequence of discarded path bits u_k from k back to $k - \delta$ to compare the differences with the surviving path. This series of bits is called the update sequence and is given by output modulo 2 between the previous $\delta + 1$ and the decoded bit along the surviving and the discarded paths. When SOVA identifies the ML path, the update sequences and metric differences along the path are stored and used to calculate the value of $L\,(u_k \mid \underline{y})$.

C. Iterative decoding

Now we will describe how iterative decoding works. Fig. 2 shows the schematic of turbo decoder, and it describes the inputs and outputs of individual blocks.

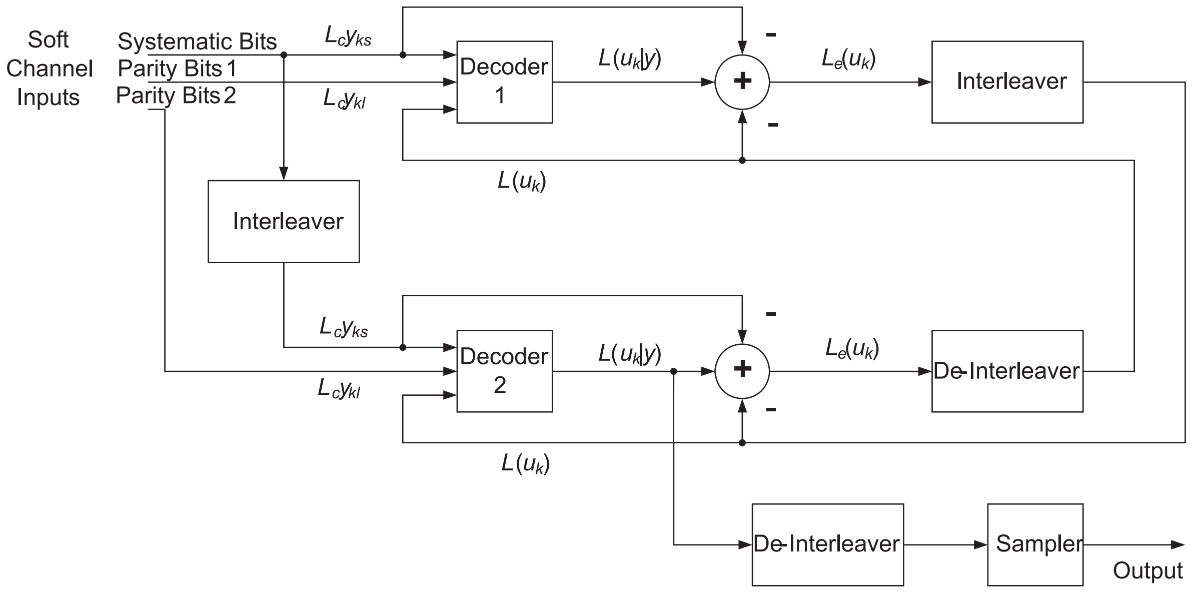

Fig. 2 Turbo decoder schematic

The first decoder in the first iteration receives a sequence $L_c \underline{y}^{(1)}$ from the transmission channel, which includes systematic bits $L_c y_{ks}$ and parity bits $L_c y_{kl}$ from the first encoder. Usually only half of the parity bits are received because these bits have been punctured in the transmitter. The decoder inserts zeros on the punctured places in the soft channel output $L_c y_{ks}$. The first decoder begins processing the soft input from the channel. The output of the first decoder is conditional LLR $L_{11}(u_k \mid \underline{y})$ of data bits u_k, where $k = 1, 2, \dots N$. The subscript of symbol $L_{11}(u_k \mid \underline{y})$ denotes a-posteriori LLR in the first iteration from the first decoder. In the first iteration the first decoder has no a-priori information about bits, therefore the value of $L(u_k) = 0$, which corresponds to an a-priori probability of 0.5. Now the second decoder begins to operate. It receives the sequence $L_c \underline{y}^{(2)}$ which contains systematic bits for the first decoder which passes through the interleaver and the parity bits from the second encoder. Furthermore, it receives a-priori LLRs $L(u_k)$ which is generated from the conditional LLR $L_{11}(u_k \mid \underline{y})$ from the first decoder. As can be seen from the figure, the extrinsic information $L_s(u_k)$ from the first decoder is adjusted by the interleaver to match with the sequence of input bits entering the second decoder. The second decoder uses this information and the received interleaved sequence $L_c \underline{y}^{(2)}$ to calculate the a-posteriori LLR $L_{12}(u_k \mid \underline{y})$. Now by the equation [4, 5]:

$$L_s(u_k) = L(u_k|\underline{y}) - L(u_k) - L_c y_{ks} \tag{5}$$

the systematic soft input $L_c y_{ks}$ and a-priori information $L(u_k)$ from the previous decoder are subtracted from the decoder output $L(u_k \mid \underline{y})$. The calculated value is the extrinsic information $L_s(u_k)$ and it is used as a-priori information for the first decoder in the second iteration. This ends the first iteration for both decoders.

In the second iteration the first decoder processes the received sequence $L_c \underline{y}^{(1)}$ again, but now it has available a-priori information which is de-interleaved extrinsic information $L_s(u_k)$ calculated by the second decoder in first iteration from the a-posteriori $L_{12}(u_k \mid \underline{y})$. Now, the first decoder can calculate a more accurate a-posteriori LLR $L_{21}(u_k \mid \underline{y})$. The second iteration continues in the second decoder. It uses the more accurate a-posteriori LLR $L_{21}(u_k \mid \underline{y})$ from the first decoder which calculated more accurate a-priori information $L(u_k)$ by using Equation (5). This information is used together with the received sequence $L_c \underline{y}^{(2)}$ to calculate $L_{22}(u_k \mid \underline{y})$ from which $L_s(u_k)$ is then calculated for the following (first) decoder.

When the series of iterations is completed, the turbo decoder output is given by de-interleaving the a-posteriori LLR $L_{12}(u_k \mid \underline{y})$ of the second decoder where i is the number of iterations used. The signs in a-posteriori sequences give the hard decision output, that is $+1$ or -1.

4. Performance analysis of turbo codes

In this chapter we will present simulations based on the effect of parameters on the performance of turbo codes. The parameters that were used in the simulation are shown in Table 1. Turbo encoder uses two parallel concatenated encoders. Selected as the code was the RSC with generator polynomials $G_0 = 37$, $G_1 = 21$ (octal) and constraint length of code $K = 5$. The interleaver chosen was the pseudo-random interleaver with length $L = 2048$ bits. Unless specified otherwise, puncturing the parity bits to one half will always be used, which will increase the information rate to $R = 1/2$. The decoder uses the SOVA algorithm; usually 8 iterations were used for decoding. The AWGN transmission channel with BPSK modulation is used in the simulation.

Parameters of turbo encoder and decoder Table 1

Channel	AWGN
Modulation	BPSK
Encoders	Two identical RSC
RSC Parameters	$n = 2$, $k = 1$, $K = 5$, $G_0 = 37$, $G_1 = 21$
Puncturing	Half parity bits from each encoder, information rate $R = 1/2$
Decoder	SOVA
Iterations	8
Interleaver	2048-bit pseudo-random interleaver

The performance of turbo codes can be influenced by many parameters. Some of these parameters are:

- The number of decoding iterations used.
- The use of puncturing in encoding.
- The generator polynomials of the codes.
- The frame lengths of input data.

A. Effect of the number of iterations

Fig. 3 shows the performance of turbo codes depending on the number of decoder iterations. Uncoded BER is shown for comparison. The performance after the first iteration of the turbo decoder should be theoretically comparable with the performance of the convolutional code [5]. As the number of iterations increases, the performance of the decoder increases too. For example, the improvement of the performance between the first and second iterations is about 1.2 dB at BER 10^{-4}. This performance increase continues up to the eighth iteration. Code gain between the eighth and fourteenth iteration is only 0.1 dB at BER 10^{-4}. From the figure it is possible to conclude that the increasing number of iterations increases not only the performance of the code but also the computational complexity in decoding; therefore it is recommended to use between 4 and 14 decoder iterations. For this reason, only 8 decoder iterations are used in the following simulations.

B. Effect of puncturing

As already described, the turbo encoder uses two or more encoders which produce parity bits. In these simulations the RSC encoders are used. This is the most common solution which is able to achieve an information rate of below 1/3. In order to achieve an information rate of 1/2, every second parity bit from each encoder must be punctured. It is also possible to use the code without puncturing and thus keep the information rate at 1/3. The performance of unpunctured code is shown in Fig. 4. Encoders use the same parameters as in the previous simulation, Fig. 3. The turbo encoder for unpunctured code has at BER 10^{-4} of a code gain which is 0.5 dB better than the turbo encoder which used puncturing. Very similar gains may also be achieved for different

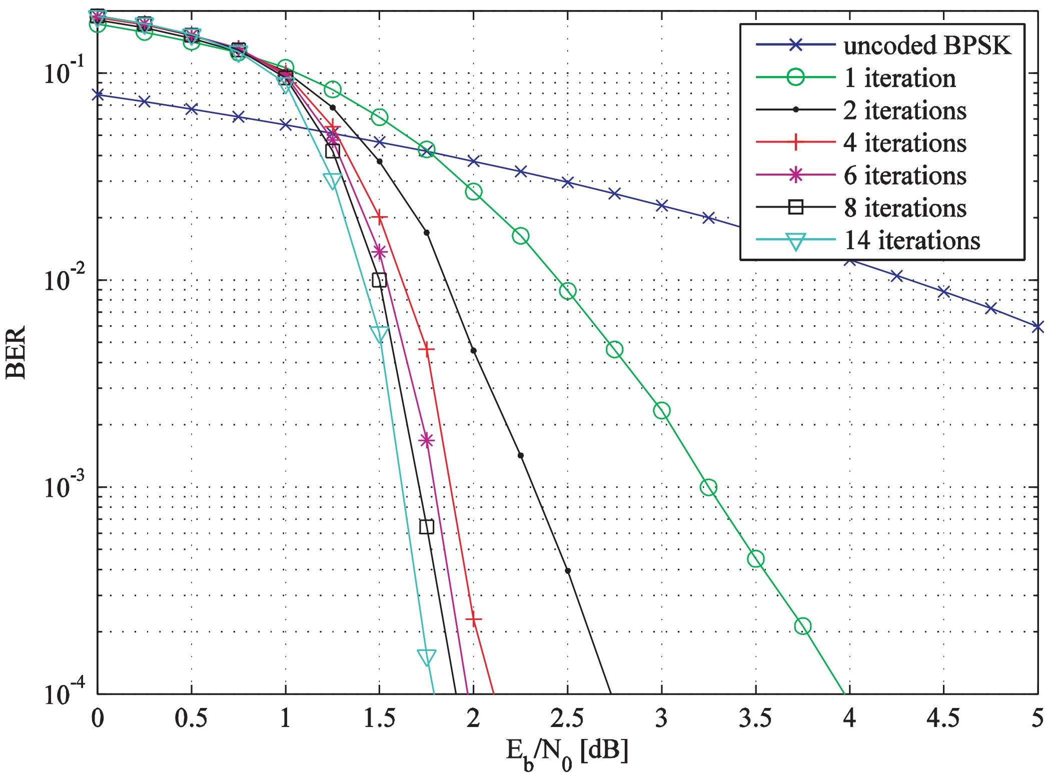

Fig. 3 Turbo coding BER performance using different numbers of iterations

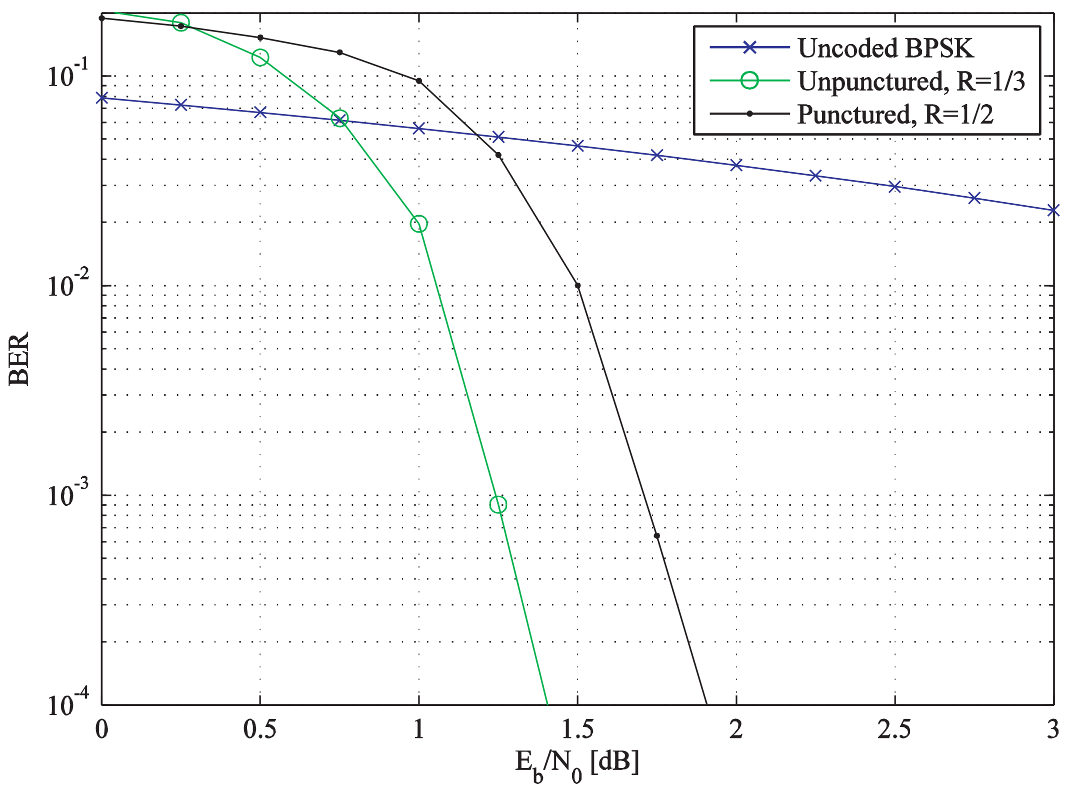

Fig. 4 Comparison of BER performance between punctured and unpunctured turbo codes

generator polynomials. From the figure it is possible to conclude that better results are obtained if puncturing is not used. But this is an improvement in the order of tenths of a decibel. Taking into account the lower number of transmitted bits in the case of puncturing, in some cases it may be considered preferable to use puncturing. Puncturing does not affect the computational complexity of encoding and decoding. It only reduces the number of bits transmitted by a transmission channel and thus reduces the transmission bandwidth.

C. Effect of generator polynomial

Fig. 5 shows the dependence of the performance of turbo convolutional code on the generator polynomial. The first code selected was the RSC code with generator polynomials $G_0 = 7$, $G_1 = 5$ and constraint length $K = 3$. The second code selected was $K = 4$, $G_0 = 17$, $G_1 = 15$. This code achieves performance that is about higher than that achieved by the code with constraint length $K =$ $= 3$ at BER of 10^{-4}. The third selected code, which was used for all simulations, has a constraint length $K = 5$ and generator polynomials $G_0 = 37$, $G_1 = 21$. Compared with the first code ($K = 3$), it reaches a performance that is about 0.3 dB higher at a BER of 10^{-4}; in comparison with the code $K = 4$, its performance increases by about 0.125 dB. With increasing constraint length of the code and with greater generator polynomials the performance of turbo codes increases, but what also increases is the size of trellis diagram and thus the computational complexity of decoding.

D. Effect of frame length

Fig. 6 shows the performance of turbo codes depending on the frame length. For many applications, such as applications using real-time transmission, a large frame length is absolutely unacceptable. Frames with a length of 256 bits are useful for voice transmission and 1024 to 2048 bits for video transmission. Systems with larger frame lengths can be used to transfer data and for applications that do not require real-time transmission. The best result in the simulation was reached by a turbo code with a frame length of 65536 bits. The turbo code with a frame length of 65536 bits has a code gain of 0.35 dB compared to turbo codes with a frame length of 2048 bits and 0.6 dB to turbo codes with a frame length of 1024 for BER of 10^{-4}. With growing frame length the performance of turbo convolutional codes increases but the delay gets affected and for shorter frames reaches lower values.

5. Conclusion

This article deals with the problem of turbo codes. It describes a basic structure of turbo encode using two identical RSC codes and turbo decoder which uses Viterbi algorithm. Furthermore, it also presents basic mathematical equations for the SOVA decoding algorithm and describes iterative decoding. Simulations were performed for different parameters of turbo codes. Based on these simulations, it is possible to conclude that the performance of

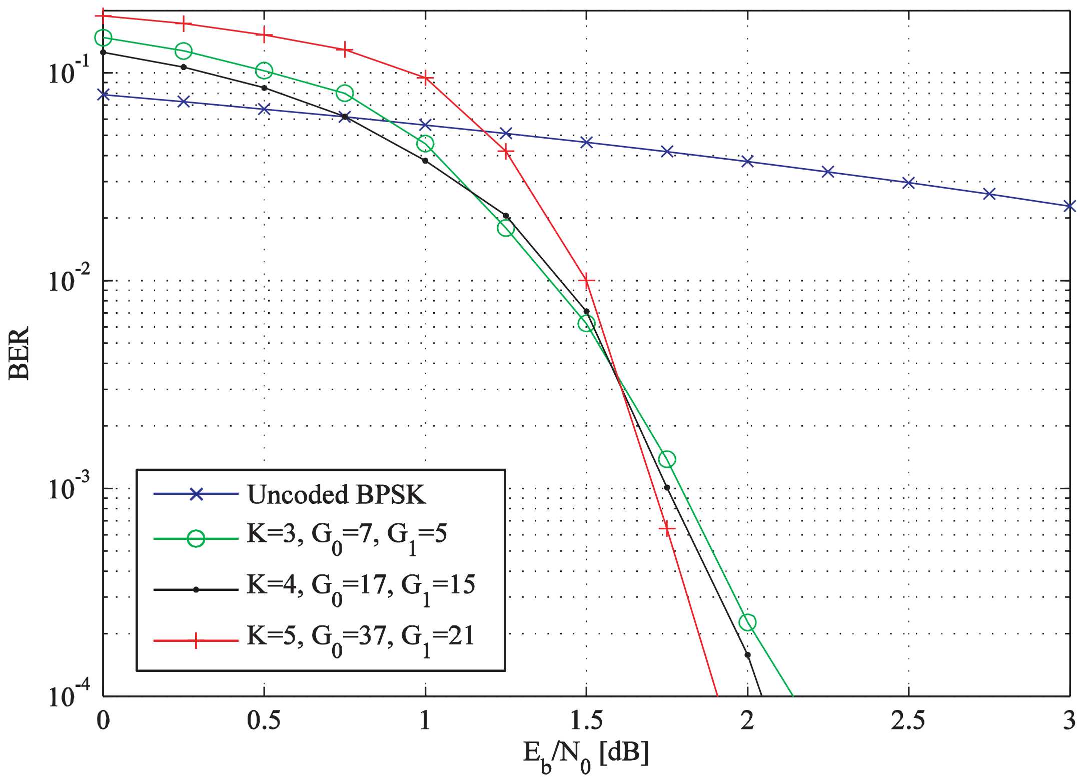

Fig. 5 Effect of constraint length and generator polynomial on the BER performance of turbo coding

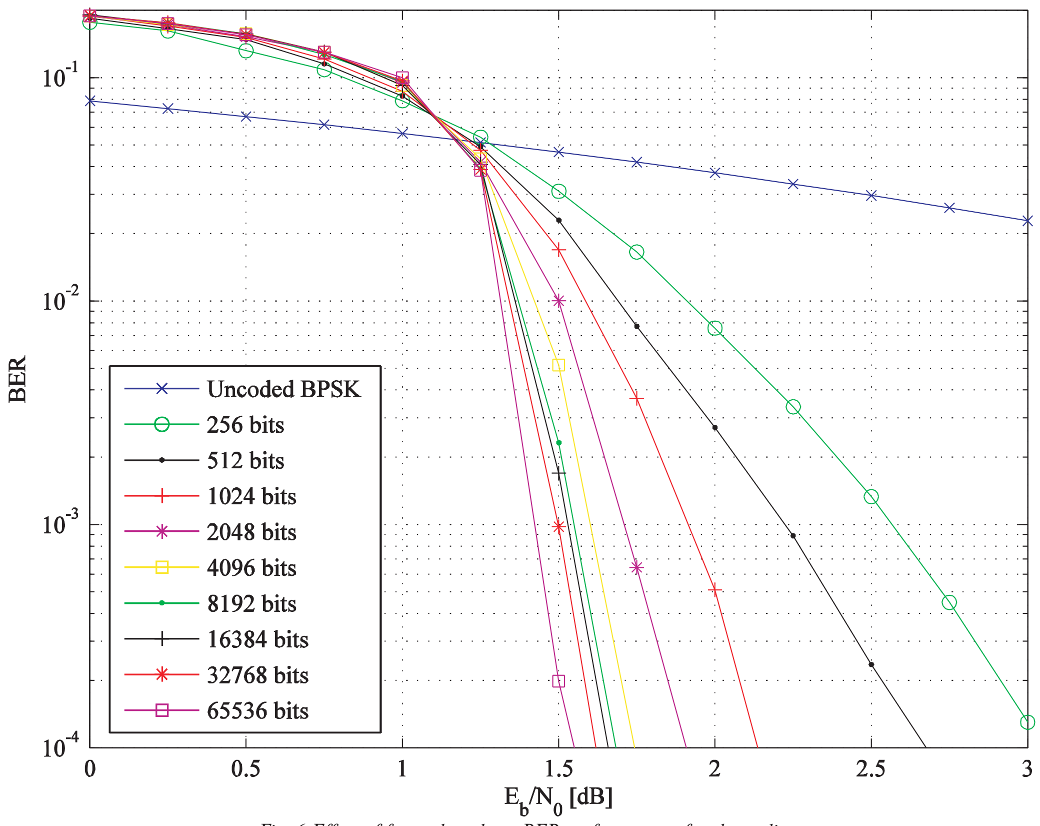

Fig. 6 Effect of frame length on BER performance of turbo coding

turbo codes decreases when puncturing is used. On the contrary, the performance of turbo codes increases with increasing number of the decoding iterations performed by an appropriate choice of the code (generator polynomial) or by changing the frame length. It is possible to implement a high performance codec.

Acknowledgement

The described research was performed in laboratories supported by the SIX project; the registration number CZ.1.05/2.1.00/03.0072, the operational program Research and Development for Innovation and the Technology Agency of Czech Republic project No. TA02020856.

References

[1] LEE, L. H. C.: *Convolutional Coding – Fundamentals and Applications.* Artec House, 1997, ISBN 0-89006-914-X.
[2] LIN, S., COSTELLO, D. J.: *Error Control Coding: Fundamentals and Applications, second edition.* Prentice Hall: Englewood Cliffs, NJ, 2005, ISBN: 0-13-042672-5.
[3] FRANEKOVA, M., NAGY, P.: Ciphering Systems Based on The Error Correcting Coding Techniques, *Communications – Scientific Letters of the University of Zilina,* No. 3, 1999, ISSN 1335-4205
[4] GLAVIEUX, A.: *Channel Coding in Communication Networks: From Theory to Turbo Codes.* Wiley-ISTE, 2007, ISBN: 978-1-90520-924-8.
[5] HANZO, L., LIEW, T. H., YEAP, B. L.: *Turbo Coding, Turbo Equalisation and Space-Time Coding for Transmission over Fading Channels.* John Wiley, 2002, ISBN: 0470847263.
[6] MOON, T. K.: *Error Correction Coding: Mathematical Methods and Algorithms.* Wiley-Interscience, 2005, ISBN-13: 978-0070010697.
[7] FARELL, P. G., MOREIRA, J. C.: *Essentials of Error-Control Coding.* John Wiley, 2006, ISBN-13 978-0-470-02920-6.