Available online at www.sciencedirect.com

SciVerse ScienceDirect

Journal of the Franklin Institute 349 (2012) 1721–1733

Journal of The Franklin Institute

www.elsevier.com/locate/jfranklin

Linear-constant log-map, a fast accurate algorithm for map decoding

Hamid Samadian*, Amir Mousavie Nia

Electrical and Computer Engineering Department, Khaje Nasiroddin Toosi University, Tehran, Iran

Received 10 July 2011; received in revised form 18 November 2011; accepted 3 February 2012
Available online 22 February 2012

Abstract

This paper proposes a new approximation to be used for the correction function in the turbo decoding algorithm, called Linear-Constant-log Map. Max-log Map, Linear-log Map and Constant-log Map are the well known simplified versions of Jacobi-log Map (Maximum a Posteriori) algorithm already in use but they cannot meet a proper performance in term of output BER and clock consumption of the CPU decoding encoded bits. The proposed algorithm first breaks the correction function domain of the Jacobi logarithm to three subsections by determining the border points between these sections and then uses a linear function and two constant values as an approximation of this function. Using an AWGN channel model, simulation results show that the new algorithm is almost more than six times faster than Jacobi-log Map algorithm with a Bit Error Rate (BER) very close to it.
© 2012 The Franklin Institute. Published by Elsevier Ltd. All rights reserved.

1. Introduction

The third generation (3G) portable wireless applications require higher data rates at lower channel SNR than ever before. This means that more advanced error correcting techniques are required to get reliable data transmission. Turbo code [1] introduced in 1993, provides near Shannon limit error correcting performance with acceptable decoding complexity, candidates it as a coding scheme for the 3GPP standard for the European 3G

*Corresponding author.
E-mail addresses: Samadian.hamid@gmail.com (H. Samadian), moosavie@eetd.kntu.ac.ir (A.M. Nia).

0016-0032/$32.00 © 2012 The Franklin Institute. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.jfranklin.2012.02.003

Universal Mobile Telecommunications System (UMTS) and cdma2000 third-generation cellular standards.

It should be said that Turbo code's issues can be divided into different categories, each focuses on some aspects of this coding algorithm that have led to introduction of new achievements. Ref. [2] presents a new method for the performance evaluation of bit decoding algorithms. In [3] Turbo code is proposed in optical OFDM multimode fiber communication system in order to decrease the bit error rate (BER) of the system. Ref. [4] introduces an efficient turbo encoding scheme which considers unpunctured trellis-coded modulation constituent encoders.

Fig. 1 presents a general simplified digital transmission block diagram, composed by: source encoder, channel encoder, digital modulator, transmitter, coding channel (AWGN), receiver, digital demodulator, channel decoder, source decoder and finally output information. In such a communicational system, Turbo algorithm can be served as one of the best and most reliable choices for channel encoder and decoder in transmitter and receiver respectively.

Fig. 2 shows the Turbo encoder scheme that has been used by UMTS. As Fig. 2 shows it is composed of two constraint length 4 Recursive Systematic Convolutional (RSC) encoders working in parallel [5]. The feed forward and the feedback generators are 15 and 13 in octal form consecutively. A block of K elements is supposed to enter to turbo encoder input, while the two used switches are held in the up position. Data is encoded by the RSC Encoder 1 in its natural order. Simultaneously it is first interleaved by the interleaver block using a predefined interleaving algorithm before applying to the RSC Encoder 2.

The interleaver uses a matrix which its rows and columns, depend on the size of the input block. Data is written into the interleaver in column wise from top to bottom and left to right. For interleaving intra-row permutations are performed on each row of the matrix in accordance with a rather complicated algorithm, which is fully described in [5]. Now for the next step, inter-row permutations are performed to change the ordering of the rows without changing the ordering of elements within each row.

After the intra-row and inter-row permutations, data is read from the interleaver in a row wise fashion from left to right and top to bottom.

When encoding has been finished the data bits are transmitted with the parity bits generated by the two parallel encoders, so the overall code rate of the encoder is $R=1/3$, not including the tail bits. The first 3 K output bits of the encoder are in the form: V_1, U_1,

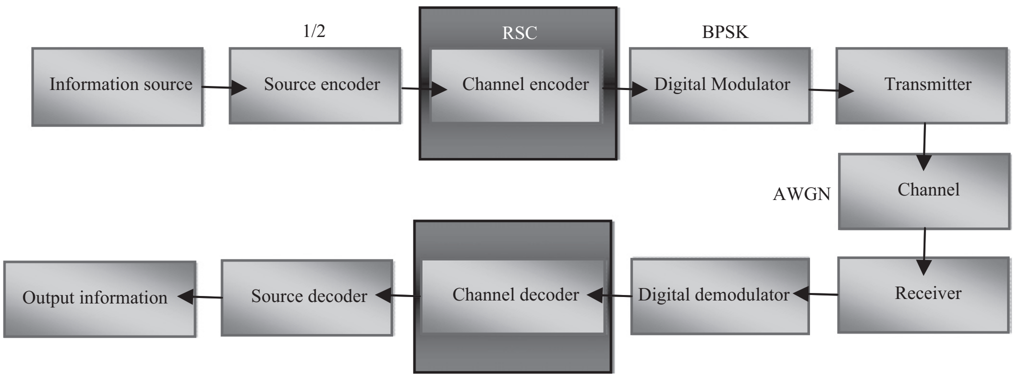

Fig. 1. An entire digital transmission overview.

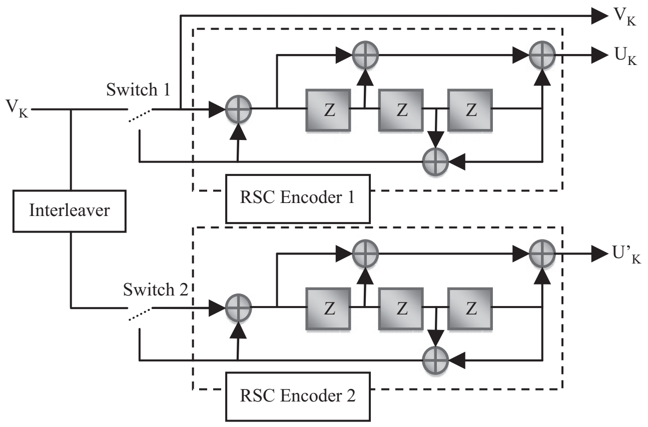

Fig. 2. Turbo encoder (CDMA2000).

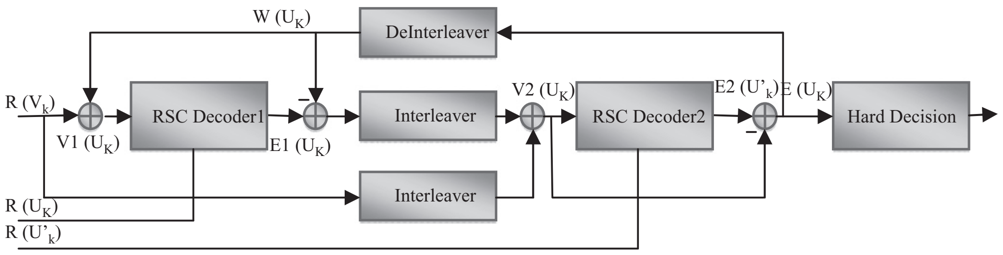

Fig. 3. Turbo decoder structure.

U'_1, V_2, U_2, U'_2, …, V_k, U_k, U'_k, where V_k, U_k and U'_k are the kth systematic (i.e., data) bit and the parity outputs which are generated from the two paralleled encoders respectively.

After all K data bits have been encoded, both encoders should be forced back to the all-zero state by the proper selection of the tail bits, preparing encoder for the next block entrance. Unlike the conventional convolutional codes, which can always be terminated with a tail of zeros, the tail bits of an RSC depend on the states of the encoder. As the states of the two RSC encoders are usually different after the data has been encoded, the tails for each encoder must be separately calculated and transmitted by throwing the two switches into the down position.

The most important feature that makes Turbo code as one of the best solution for reliable data transmission is related to its high performance decoding structure which comes from its recursive nature. Decoding in turbo codes is mostly based on the iterative MAP (Maximum A Posterior) procedure. A high-level block diagram of a turbo decoder is shown in Fig. 3.

For an AWGN[1] channel with variance σ^2 and BPSK[2] modulation the input to the decoder is assumed to be in Log-Likelihood Ratio (LLR) form, which assures that the

[1]Additive white gaussian noise.
[2]Binary phase-shift keying.

noise variance have been properly taken into account. Thus, the input to the decoder is in the form of [6]

$$R(S_k) = \ln\left(\frac{fy(W_k|S_k = \sqrt{E_c} = \sqrt{1/2})}{fy(W_k|S_k = \sqrt{E_c} = -\sqrt{1/2})}\right) \tag{1}$$

where W_k is the received signal from the noisy channel corresponding to the S_k and $fy(W_k|S_k)$ is the conditional Probability Density Function (pdf) of W_k given S_k, which is Gaussian with mean 0 and variance σ^2 and S_k is modulated data bits in transmitter that have bipolar values.

RSC Decoder1 receives LLRs of systematic and coded bits of Encoder1 as its inputs. The LLR of a systematic component is formed from channel LLRs and the deinterleaved output of RSC Decoder2 (this component is 0 in the first iteration). The value $w(U_k)$, is the extrinsic information produced by decoder number 2 and introduced to the input of decoder number 1. The values of $w(U_k)$ will be updated to reflect beliefs regarding the data propagated from decoder number 2, back to decoder number 1. The output of RSC1 (referred to as refined LLR) is interleaved and used as systematic input into RSC2, along with coded input $R(V'_k)$. As Fig. 3 represents the output of RSC2 is deinterleaved and fed back as one of the systematic components for RSC1. Due to this feedback path, it is possible to repeat the decoding process (on the same frame of data) iteratively. After the last iteration, all LLRs related to the systematic component are added and sent to the hard decision module, which transforms the LLR sequence into a binary sequence, representing the decoded version of the original sequence, U_k. The hard decision module performs sign detection on the LLR and selects 1 when the LLR is greater than 0, or selects 0 in all other cases.

The main problem in implementing this decoding structure is that it is too complex so it consumes relatively long time for completing its decoding process on the receiver side.

After proposing Turbo code, several attempts have been done [7–10] to improve this defect that have led to some lighter algorithms that try to simplify the decoding procedure by applying their estimating function to Turbo decoder's origin decoding form. The most famous algorithms that deal with this simplification are: Max-log Map which has the highest BER (Bit Error Rate) and lowest computation, Linear-log Map with a better BER with an increased complexity and finally the Constant-log Map whose BER and computation features are between those of Linear and Max-log Map.

They differ by the way they estimate the correction term ($f(\delta)$) in the Jocobian function which is the most time consuming part of the Turbo decoding algorithm:

$$\begin{aligned}\ln(e^x + e^y) &= \max(x,y) + \ln(1 + e^{-|x-y|}) \\ &= \max(x,y) + f(|x-y|) \\ &= \max(x,y) + f(\delta) \end{aligned} \tag{2}$$

This term, in addition has huge effect on the output data in terms of its BER. Each one of these algorithms tries to simplify this term to reduce the amount of computation needed to decode the input data but they usually loose the precision of the original algorithm which is critical for proper implementation of Turbo code. So in order to improve, this defect in these present algorithms, a new method that considers both parameters in proper implementation without having that limit, seems to be necessary in Turbo code decoding procedure.

To meet this requirement a new approximation algorithm is presented for the Jacobi function that has either an advantage of the precise computation and the simplification that is made by Linear-log Map with applying a linear approximation to the Jacobi logarithm.

The following sections of this paper are organized as follow: in the next section a short overview of an operator so-called max* operator has been provided, Section 3, describes the proposed new approximation technique to improve correction function (max*). Simulation results are given in Section 4, and finally Section 5, concludes the paper.

2. The max* operator

First presented in 1974 [11], the Bahl, Cocke, Jelinek, and Raviv (BCJR) algorithm, which exploits the trellis structure of a convolutional code, is one of the most frequently used MAP decoding algorithms. While this algorithm has been known for a long time, it was not extensively used for decoding of convolutional codes due to the availability of a less complex algorithm, Viterbi. The Viterbi algorithm, however, delivers only hard decisions and not probability distributions (or LLRs), and therefore cannot be used directly in turbo decoding. The BCJR algorithm for MAP decoding of convolutional codes consists of the following computational steps: branch metrics γ, forward state metrics α, backward state metrics β, and extrinsic LLR, λ_e. So as a result, the RSC decoders in Fig. 3 are each executed using a version of the classic MAP algorithm[12], implemented in the log-domain[13]:

$$\Gamma_t(p,q) = \ln\gamma_t(p,q) = R(U_k)\frac{(2U_k-1)}{2} + R(V_k)\frac{(2V_k-1)}{2} \tag{3}$$

$$A_t(q) = \ln\alpha_t(q) = \ln\left(\sum_{p=0}\alpha_{t-1}(p)\gamma_t(p,q)\right) = \ln\left(\sum_{p=0} e^{[A_{t-1}(p)+\Gamma_t(p,q)]}\right) \tag{4}$$

$$B_t(p) = \ln\beta_t(p) = \ln\left(\sum_{q=0}\beta_{t+1}(q)\gamma_t(p,q)\right) = \ln\left(\sum_{p=0} e^{[B_{t+1}(q)+\Gamma_t(p,q)]}\right), \tag{5}$$

$$\begin{aligned}\lambda_{e,t} &= \ln\frac{\sum_{(p,q)\in S_1}\alpha_t(pf_1(\delta))\gamma_t(p,q)\beta_{t+1}(q)}{\sum_{(p,q)\in S_0}\alpha_t(p)\gamma_t(p,q)\beta_{t+1}(q)}\\ &= \ln\frac{\sum_{(p,q)\in S_1} e^{[A_t(p)+\Gamma_t(p,q)+B_{t+1}(q)]} = \text{extrinsic}-\text{numerator}}{\sum_{(p,q)\in S_0} e^{[A_t(p)+\Gamma_t(p,q)+B_{t+1}(q)]} = \text{extrinsic}-\text{denominator}}.\end{aligned} \tag{6}$$

As shown in Eqs. (4)–(6), most of computations are based on the logarithm of a sum of exponentials. Such an expression can be exactly performed, two terms at the time, using the Jacobian logarithm [13], as shown in Eq. (7):

$$\begin{aligned}\max{}^* &= \ln(e^x + e^y) = \max(x,y) + \ln(1 + e^{-|x-y|})\\ &= \max(x,y) + f(|x-y|) = \max(x,y) + f(\delta)\end{aligned} \tag{7}$$

which is the maximum of the two arguments plus a nonlinear term, known as correction function ($f(\delta)$). That is only a function of the absolute difference between the two arguments represented as δ. It should be said that as it can be perceived, δ refers to the difference between the powers of the exponential terms in Eqs. (4)–(6).

Since the function ($f(\delta)$) depends upon only a single variable, it is straightforward to pre compute it and determine its values in use by lookup table or using the log and exp functions in software implementation.

Because the max* operator must be executed twice for each node in the trellis during each half-iteration, it constitutes a significant, and sometimes dominant, portion of the overall decoder complexity. The manner that max* is implemented is critical to the performance and complexity of the decoder, and several methods have been proposed for performing its computation. Below, we consider three versions of these algorithms: Max-log Map, Constant-log Map, and Linear-log Map. The correction function used by the Jacobi-log Map algorithm is illustrated in Fig. 4, along with the correction functions used by the Constant-log Map and Linear-log Map algorithms. Each one of these algorithm uses a simplified version of the original correction function.

2.1. *Max-log-map algorithm*

For the first attempt in simplifying the complexity of the decoding computation, the second term of the max* altogether, can be simply ignored and just the max instruction should be considered. This reduces the amount of instructions required but also increases the BER and inversely reduces the performance of the decoder block, since the computations are not accurate:

$$\max^* \approx \max(x,y). \tag{8}$$

2.2. *Constant-log-map algorithm*

The Constant-log Map algorithm, first introduced in [10] approximates the Jacobi logarithm using:

$$\max^* \approx \begin{cases} \max(x,y) & \text{for } |x-y| > 1.5 \\ \max(x,y) + Q & \text{for } |x-y| \leq 1.5 \end{cases}. \tag{9}$$

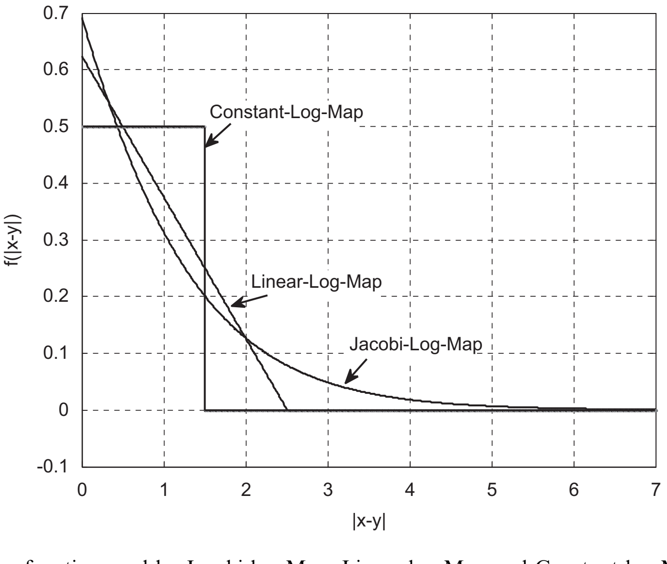

Fig. 4. Correction function used by Jacobi-log Map, Linear-log Map and Constant-log Map algorithms.

It is shown in [7] that the best value for the UMTS turbo code is $Q=0.5$. This algorithm is equivalent to the log-MAP algorithm with the correction function that is implemented by a 2-element look-up table. The performance and complexity of this implementation is between that of the Jacobi-log Map and the Max-log Map algorithms.

2.3. Linear-log-map algorithm

The Linear-log Map algorithm, first introduced in [8], uses the following linear approximation to the Jacobi logarithm:

$$\max{}^* \approx \begin{cases} \max(x,y) & \text{for } |x-y|>w \\ \max(x,y)+[k(|x-y|-w)] & \text{for } |x-y|\leq w \end{cases}. \tag{10}$$

In [9], the values of the parameters k and w, were calculated by assumption that a floating-point processor is available, so by minimizing the total squared error between the exact correction function and its linear approximation yields $k=-0.24904$ and $w=2.5068$. The Linear-log Map algorithm offers performance and complexity between that of the Jacobi-log Map and Constant-log Map algorithms.

3. Linear-constant, a new approximation to the correction function

This section proposes a new approximation algorithm to the max* that has either an advantage of the precise computation of max* which is used by Jacobi Algorithm and a simplification that is made by Linear-log Map with applying a linear approximation to the Jacobi logarithm.

As simulation results have been shown in the next section this new approximation so-called Linear-Constant-log Map algorithm has the BER performance output much like to that of the Jacobi-log Map algorithm but with applying a new simplification method to the precise Jacobi function, a software implementation can be proposed that is six times faster than the last one (means log-map algorithm). So it can be claimed that this Linear-Constant algorithm has the best performance in compare with other algorithms that were discussed in the previous section.

By looking at Fig. 4, which has shown the approximation that Linear-log Map applies to the Jacobi correction function, it can be easily perceived that as a result of using only one approximating function for minimizing log and exp functions in the first critical section (i.e. values between 0 and $w=2.5068$ for $|x-y|$), a much more deviation number will be resulted, in compare with the precise Jacobi values in that section. By considering this flaw in the Linear-log Map algorithm, a new one has been proposed ($f_1(\delta)$), that uses the advantage of two separated section, for attaining closer values to that of the Jacobi correction function, or in other words, less deviation or Manhattan Norm defined as $\|f_1(\delta)-f(\delta)\|_1 := \int_{\delta=0}^{\infty} |f_1(\delta)-f(\delta)|$. To access this point, all the attempts should be focused on the section that the values of the correction function $f(\delta)$ have more influence on the final computations precision. It can be easily seen that when δ exceeds from 3.5 (marked as C in Fig. 5), $f(\delta)$ values are converging into zero, so the section in restriction between $0\leq\delta\leq3.5$ can be supposed as our critical section. This means that, the new approximated correction function can still have a proper precision if the correction values of $f(\delta)$, are ignored and given equal to zero for $\delta>3.5$.

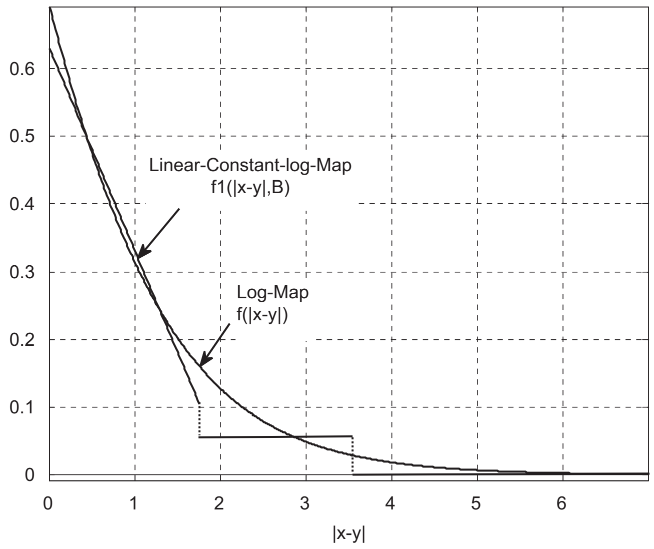

Fig. 5. Correction function used by log-Map and Linear-constant log-Map.

In this proposed simplified algorithm, in contrast with the simple Linear-log Map method, that uses one estimation function for minimizing the amount of calculation needed for decoding, two different functions have been considered as approximations for $f(\delta)$ which have been marked as [A–B] and [B–C] with linear and constant estimation respectively in Fig. 5. Point B as it has been illustrated in Fig. 5 is the border point of the two mentioned sections known as linear and constant approximated section and obviously that is one of those important points that are necessary to propose for this new algorithm. For this, the Distribution functions of δ, in region between [A–C] should be calculated and then by attending that, the best point for B that can result the best performance in BER output can be determined.

3.1. Point B value

δ's distribution pattern, can be achieved by implementing Turbo decoding algorithm in Matlab, so it can be drawn in a large scope within [0–12] for δ values. A wide range that can provide a good sketch for the desirable parameter, is the main reason for attending that wide area.

It should be reminded from Section 2 that Turbo decoding calculates four essential values named as forward state metric, backward state metric and extrinsic LLR which has two separate sections, numerator part and denominator part in each Turbo trellis node that have been calculated based on δ. By attending δ values resulted from executing 80000 nodes, Fig. 6 depicts the discrete distribution pattern of δ in percentage form, which have been resulted from four different parts of Turbo decoding algorithm in each trellis node, mentioned in Section 2, Eqs. (4)–(6).

Fig. 6 provides an explicit look around the distribution pattern of δs which asserts the Gaussian distribution form of these values for SNR = 0.25 dB in that section. So it can be inferred that this form could have a relationship with the noise variance of the AWGN

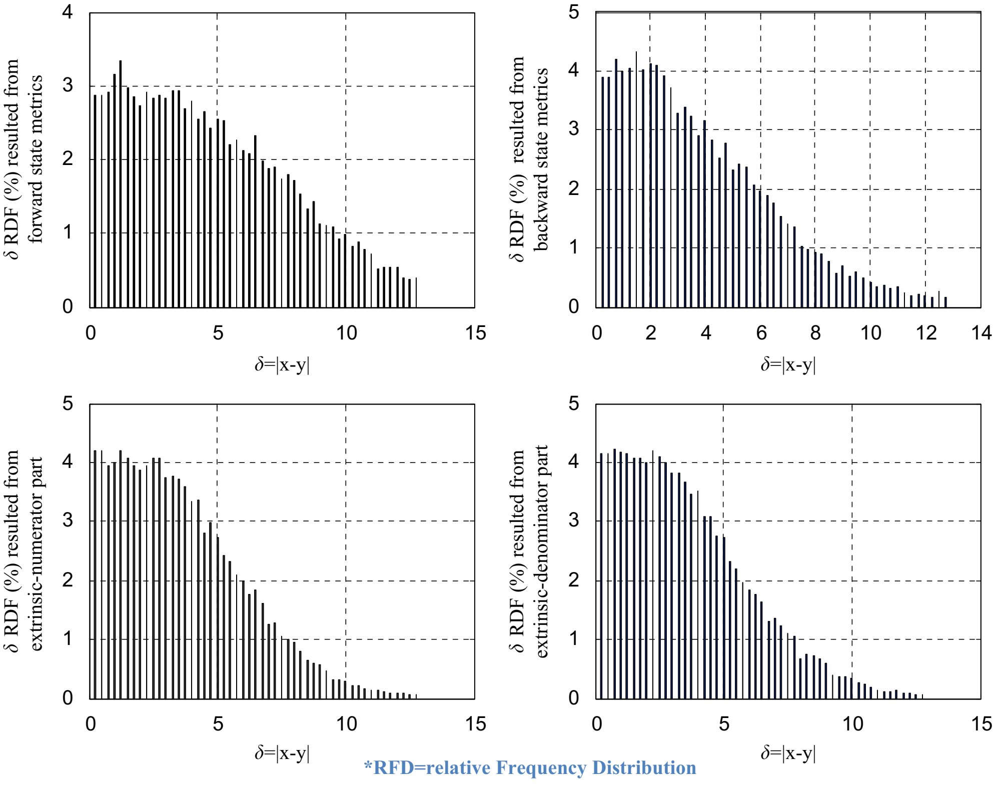

Fig. 6. Discrete distribution pattern for δ with SNR(dB)=0.25.

channel (σ^2) which the data passed through it. It can be supposed that the channel variance changes as [0.24–0.94] when the SNR(dB) changes from SNR(dB)=0.25–6 which is the customary communicational transmission variance range. With this assumption, Fig. 7 depicts the same pattern in line plot type for δs in a noisy channel when Signal to Noise Ratio increases from 0.25 to 6.

As Fig. 7 shows, δ's Distribution pattern for different practical variances has an univocal form in δ restriction presumed as $0 \le \delta \le 3.5$. Now by attending this feature, it can be concluded that no specific region in this area has special advantage over other regions. So point B can be determined only by calculating deviation or Manhattan norm which is define as $\left\| f_1(\delta,B) - f(\delta) \right\|_1 := \int_{\delta=0}^{3.5} |f_1(\delta,B) - f(\delta)|$ for two approximated sections referred as Linear and Constant that are resulted when point B slides in the critical section from 0 to 3.5. For example when border point $B_1 = 1.5$ is chosen as B and the Linear and Constant estimation related to each zone on the left and right hand of this point is calculated then the Manhattan norm computed in relation to the origin $f(\delta)$, for the Linear section will be 0.018 and in the same way, 0.048 for the Constant section. Fig. 8 shows the norm of Linear and Constant sections considering sliding B point as well as the summation of these two values (norm of Linear and Constant sections) in each point.

Fig. 8 shows this deviation from the Jacobi correction function, $f(\delta)$, for two estimated sections, when B point changes within the specific range $0 \le \delta \le 3.5$. By adding these two

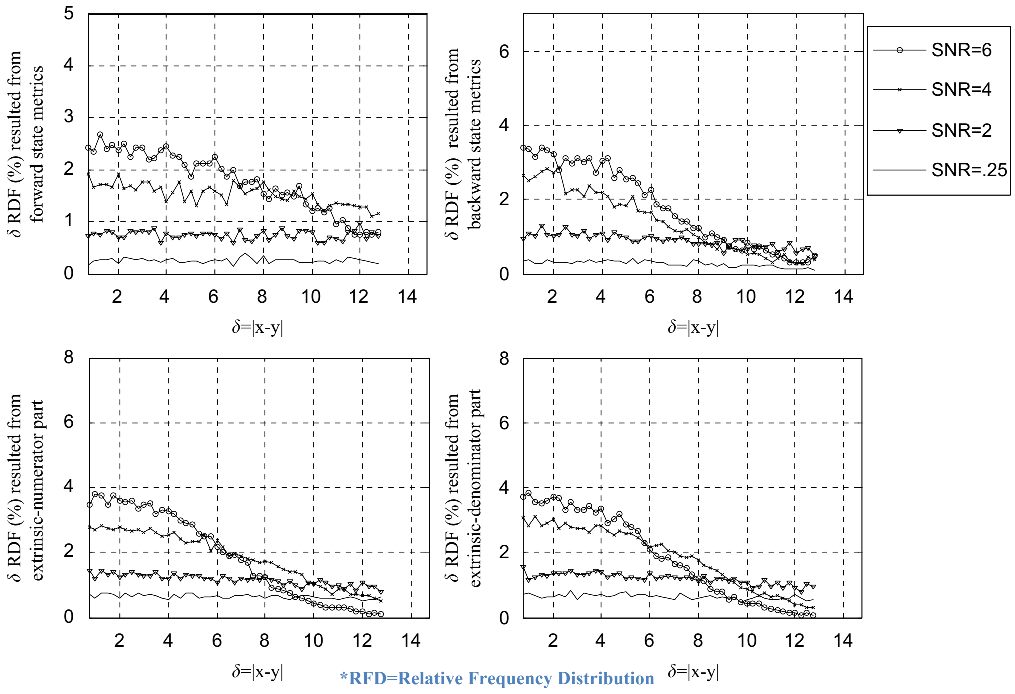

Fig. 7. Discrete distribution pattern for δ with SNR(dB)=[0.25–6].

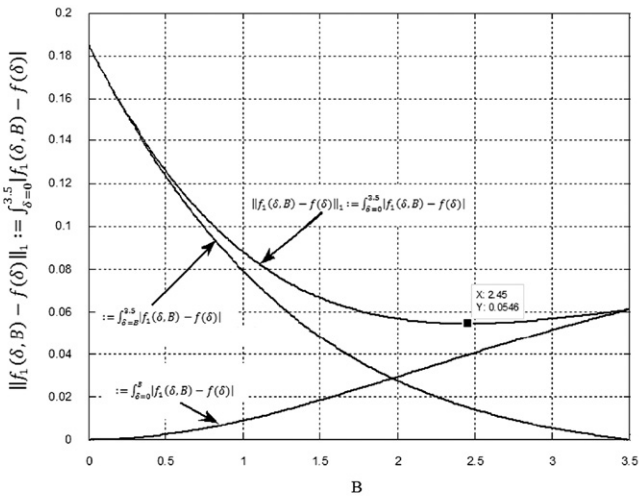

Fig. 8. Deviation value for two approximated sections which are resulted from sliding B.

curves and showing in another curve labeled as addition of two deviation values, an important result can be achieved and that is, the best point for B is the point where this summation becomes minimum, which has been shown in above figure as $x=B=2.45$.

3.2. New estimated correction function equation

By specifying the proposed limits to the new estimating method, two sections' approximated functions can be achieved. To find out that functions, the function so-called as Polyfit in Matlab software could be used, which finds coefficients of the polynomial $p(x)$ of degree n where $p(x)$ is the linear form of the output $y(x)$. In this case x and $y(x)$ are δ and correction function outcome respectively. The computations have resulted in the following equations:

$$f_1(\delta)=\begin{cases} y=-0.24x+0.596 & x\le 2.45 \\ y=0.048 & 2.45<x\le 3.5 \\ y=0 & x>3.5 \end{cases}. \tag{11}$$

4. Simulation results

In this section, simulations were run to illustrate the performance of the new proposed algorithm in compare with other four variants of the turbo decoding algorithm. Two representative, frame/interleaver sizes were used, $K=2500$ and $K=3000$ bits. For the smaller interleaver, up to 3 decoding iterations were performed, while for the larger interleaver, up to 2 decoding iterations were applied. In order to have a fair comparison, all five algorithms decoded the same received code words, and thus the data and noise, keep the same for each family of five curves.

The Bit Error Rate (BER) is shown for the $K=3000$ bit UMTS turbo code in Fig. 9 and for the $K=2500$ bit code in Fig. 10. In each case, the performance of Max-log Map is significantly worse than the other algorithms. As it can be seen, the performance of the Linear-Constant-log Map algorithm is nearly close to that of the exact computation of the log-MAP algorithm in Jacobi-log Map.

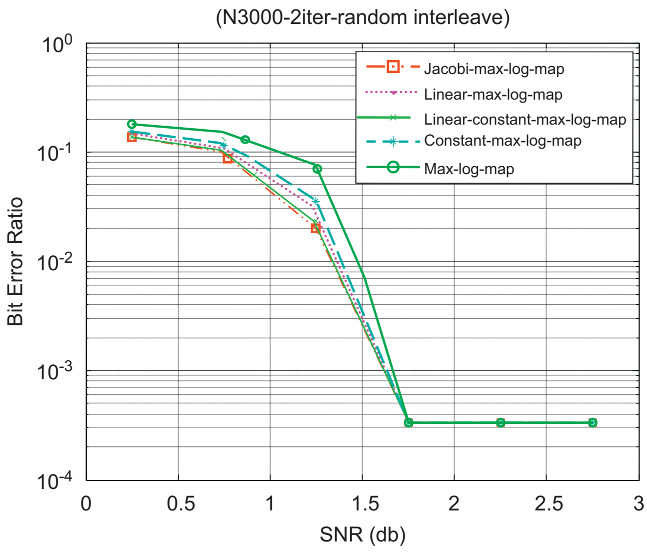

Fig. 9. BER of $K=2500$ turbo code after 2 iteration.

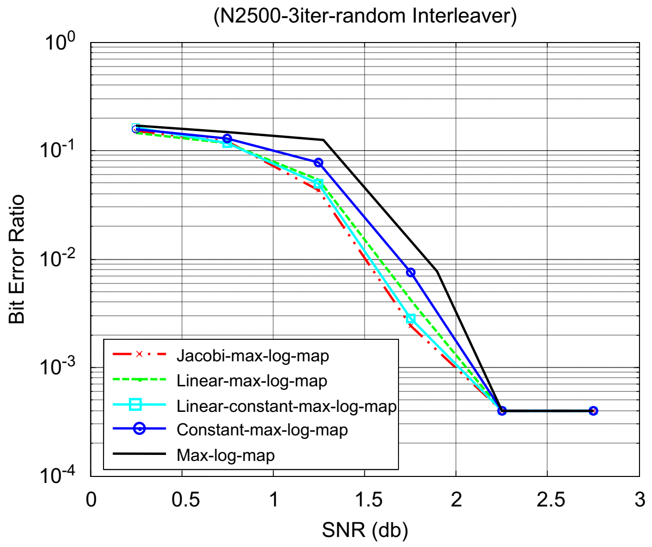

Fig. 10. BER of $K=3000$ turbo code after 3 iteration.

Table 1
Data throughput for different algorithm and different iteration number.

$N=900$ at 200 MHz	1 Iteration (Kbps)	4 Iteration (Kbps)	8 Iteration (Kbps)
Jacobi-log Map	9	2.24	1.12
Linear-constant-log Map	54	13.2	6.6
Linear-log Map	61	15	7.6
Constant-log Map	102	26	13
Max-log Map	786	229	118

In the next step for analyzing our turbo decoding algorithm, turbo code was implemented on TMS320C6713 DSP family for all five estimated algorithm. In this case a frame with 900 bit has been supposed for having a proper comparison between CPU clock cycles required in those estimated algorithms. The maximum data throughput that can be achieved in each method is tabulated in Table 1 for different decoding iterations. As Table 1 shows Linear-Constant-log Map approximation has a little increase in CPU process when it is compared with that of the Linear-log Map method. But it still has the benefit that its output BER is very close to its exact correction function referred as Jacobi-log Map algorithm BER while in the same condition its processing speed is six times faster as the Fig. 11 shows for the first iteration.

5. Conclusions

This paper has discussed about turbo decoding structure algorithm and some estimated methods that can be served as simplifying algorithms for turbo decoding implementation in software. In addition a new much more efficient approximating method has been proposed, and compared with other present algorithms. All steps in designing this new trimmed algorithm in turbo decoding were discussed and shown that this new represented

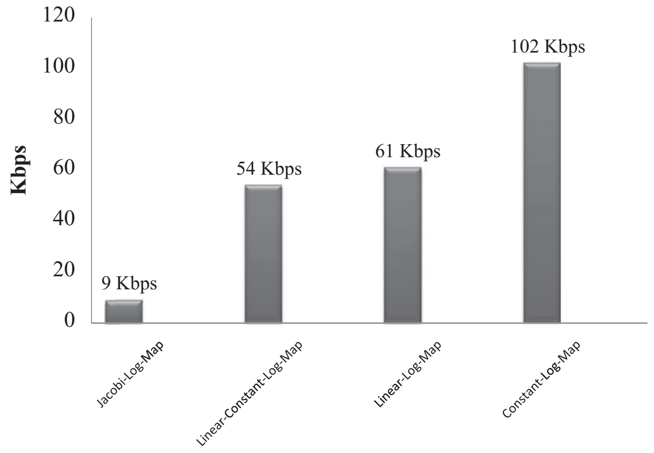

Fig. 11. Data throughput of different algorithms for one decoding iteration.

method can compete with Jacobi-log Map algorithm in output BER, which has the profit of the precise algorithm for output bit calculation, in addition to have the advantage of having a six times faster processing algorithm than Jacobi-log Map which make it a reliable and a high performance method in turbo decoding process.

References

[1] C. Berrou, A. Glavieux, P. Thitimasjshima, Near Shannon limit error-correcting coding and decoding: Turbo-codes(1), Proceedings of IEEE International Conference on Communications (Geneva, Switzerland), pp. 1064–1070, May 1993.

[2] A. Abedi, A.K. Khandani, A new method for performance evaluation of bit decoding algorithms using statistics of the log likelihood ratio, in: ELSEVIER, Journal of the Franklin Institute, June 2007.

[3] Xie Wei, Hu Guijun, Deng Qing, Application of Turbo codes in optical OFDM multimode fiber communication system, in: ELSEVIER, Journal of Optics Communications, October 2007.

[4] Abdesselam Bassou, Ali Djebbari, Efficient turbo encoding scheme using unpunctured trellis-coded modulation codes, in: ELSEVIER, International Journal of Electronics and Communications, May 2006.

[5] European Telecommunications Standards Institute, Universal mobile telecommunications system (UMTS): Multiplexing and channel coding (FDD), 3GPP TS 125.212 version 3.4.0, pp. 14–20, September 23, 2000.

[6] T.K. Moon, Error correction coding: mathematical methods and algorithms, John Wiley & Sons, Inc., Hoboken, New Jersey, 2005, pp. 605 (©.

[7] B. Classon, K. Blankenship, V. Desai, Turbo decoding with the constant-log-MAP algorithm, in: Proceedings of Second International Symposium. Turbo Codes and Related Applications (Brest, France), pp. 467–470, September 2000.

[8] J.-F. Cheng, T. Ottosson, Linearly approximated log-MAP algorithms for turbo coding. Proceedings of IEEE Vehicular Technology Conference (VTC) (Houston, TX), May 2000.

[9] M.C. Valenti, J. Sun, The UMTS turbo code and an efficient decoder implementation suitable for software-defined radios, International Journal of Wireless Information Networks 8 (4) (2002) October 2001 (©.

[10] W.J. Gross, P.G. Gulak, Simplified MAP algorithm suitable for implementation of turbo decoders, Electronics Letters 34 (August 6, 1998) 1577–1578.

[11] L.R. Bahl, et al., Optimal decoding of linear codes for minimizing symbol error rate, IEEE Transactions on Information Theory (March 1974) 284–287.

[12] L.R. Bahl, J. Cocke, F. Jelinek, J. Raviv, Optimal decoding of linear codes for minimizing symbol error rate, IEEE Transactions on Information Theory 20 (Mar. 1974) 284–287.

[13] P. Robertson, P. Hoeher, E. Villebrun, Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding, European Transactions on Telecommunications 8 (March/April 1997) 119–125.